Fine-Grained Proof Complexity for Elementary Arithmetic in Extended Frege*

Andrew Krapivin
CMU

Jiatu Li
MIT

Elaine Shi
CMU

Abstract

Proof complexity has historically focused on *qualitative* results, such as whether natural mathematical statements (e.g. the Pigeonhole Principle) admit polynomial-size proofs in certain proof systems (e.g. Resolution or Frege). In contrast to the rich theory of algorithm design, there is little understanding of how to design *quantitatively* efficient proofs. We initiate the study of fine-grained propositional proof complexity. Our motivation comes from an emerging line of cryptographic constructions such as the elegant work of Jain and Jin (2022), in which the efficiency of the resulting scheme is governed by the size of the underlying Extended Frege proofs.

Our main result is that the basic laws of integer arithmetic — commutativity and associativity of addition and of multiplication, together with distributivity — admit Extended Frege proofs of quasi-linear size, when addition is implemented by the schoolbook algorithm and multiplication by an FFT-based algorithm. This seemingly elementary result turns out to be surprisingly non-trivial: the natural paper-and-pencil arguments for distributivity and for associativity of multiplication translate into Extended Frege proofs of quadratic size. To reach quasi-linear size, we develop a technique we call *index generalization*, which we believe will find wide application in the design of efficient propositional proofs.

*A slightly differently formatted version of the manuscript has been posted on the Cryptology ePrint archive at the time of submission. This paper results from a long-running collaboration between human and AI. Detailed AI methodology is provided right before the bibliography section at the end.

Contents

1 Introduction **1**
- 1.1 Our Results . . . 3
- 1.2 Conceptual Contributions and Open Questions . . . 3

2 Definitions and Notations for Extended Frege Proofs **4**
- 2.1 Extended Frege Proofs . . . 4
- 2.2 Big-Step Rules: New Language for Modular Construction of EF Proofs . . . 5

3 New Proof Paradigm: Index-Generalization **8**
- 3.1 Building Blocks . . . 9
- 3.2 The Index-Generalization Lemma . . . 11
- 3.3 Proof of Index-Generalization . . . 11

4 Basic Warmup **15**
- 4.1 A First Example: Commutativity of Addition . . . 15
- 4.2 Reordering Summation . . . 16
- 4.3 Carry Resolution . . . 18
- 4.4 Other Reusable Inference Rules . . . 20

5 FFT Warmup **23**
- 5.1 Simplifying (False) Assumptions . . . 23
- 5.2 The FFT Networks . . . 23
- 5.3 A Selection of FFT Identities . . . 24

6 Main Example: Distributivity of Schoolbook Adder and FFT Multiplier **26**
- 6.1 FFT Multiplier and Ordinary Convolution . . . 26
- 6.2 A Naïve Quadratic EF Derivation . . . 26

7 Compressing the Derivation to Quasilinear Size **29**
- 7.1 Additional Circuits and Inference Rules for a Variable Index . . . 30
- 7.2 Applying the Index-Generalization Paradigm . . . 34
- 7.3 Putting Everything Together . . . 40
- 7.4 Associativity and Other Applications . . . 41
- 7.5 Removing the Simplifying Assumptions . . . 41

8 Additional Related Work **41**

A The Implementation Circuits **43**
- A.1 Integer Circuit Primitives . . . 43
- A.2 Scaled Real Arithmetic Circuits . . . 45
- A.3 Scaled Complex Arithmetic Circuits . . . 47
- A.4 FFT Transform Parameters and Complex Constants . . . 48
- A.5 The Binary Integer Multiplier . . . 50
  - A.5.1 The Forward FFT . . . 51
  - A.5.2 Pointwise Product . . . 52
  - A.5.3 The Inverse FFT . . . 52
  - A.5.4 Circuit Size Accounting . . . 53

A.6 EF Statements of Commutativity, Associativity, and Distributivity for Integer Arithmetic . . 53

B Reusable Big-Step Inference Rules **54**

C Short-Word Library **60**
C.1 Auxiliary Circuitry 60
C.2 Short-Word Inference Rules 63

D Inference Rules for Permutations, Routing, and Reordering **68**
D.1 Routing Identities 68
D.2 Reordering Sums and Products 73
D.3 Pairing Identities 79

E EF Verification of FFT Constants **82**
E.1 Twiddle Computation Circuit 82
E.2 Constant Certificates 83
E.3 Proof of Lemma E.3 84

F EF Verification of FFT Network Identities **98**
F.1 A Row Identity with a Variable Row Number 98
F.2 Exact Linearity and Transpose Rules 105
F.3 Network Bounds and Cancellation Identities 113

G The Rounded Coefficient Theorem **123**

H Vector Distributivity, Shift Compatibility, and Associativity **126**

I Carry Resolution **130**

J Proof of Main Theorem **133**
J.1 Bounds and Padding 133
J.2 Proof for Unsigned Inputs 136
J.3 Proof for Signed Integers 138

1 Introduction

Proof complexity studies the complexity (in certain measure such as length, steps, or width) of formal proofs (in certain proof systems) as a function of the size of the statements they establish [Kra95, CN10, Kra19]. A fundamental motivation comes from Cook and Reckhow's characterization: $\mathsf{NP} = \mathsf{coNP}$ if and only if there is a Propositional Proof System (PPS) in which every tautology has a polynomial-size proof [CR79]. Understanding the power and limitations of concrete proof systems is therefore a central research program in complexity theory. Among others, Extended Frege (EF) [CR79] is an important system in this program. It augments ordinary propositional reasoning, as formalized by Frege systems, with definitions that name intermediate formulas [Tse70], allowing later steps to reuse these formulas without writing them out again.

There is a connection between Extended Frege and another line of work, known as bounded arithmetic, that aims to formalize the concept of "*feasible mathematical reasoning*" [Kra95, CN10, Kra19, Li25]; for instance, Cook's theory PV [Coo75] formalizes reasoning about polynomial-time computations. Cook discovered a connection between PV and EF, known as Cook's propositional translation: every fixed PV proof yields a family of polynomial-size EF proofs of the propositional instances of the proved statement [Coo75] (see also [Bus97, Li25]). This translation provides a systematic way to construct short propositional proofs from uniform mathematical arguments. It also makes EF a natural setting for studying the proof complexity of mathematical properties of algorithms and circuits.

With a few exceptions (see, e.g., [BB93, Bus95]), both lines of work are concerned with the distinction between polynomial and superpolynomial proof size. To our knowledge, the precise size of EF proofs of natural statements, e.g., whether it is quasi-linear, quadratic, or cubic in the size of the statements, has not been studied systematically.

One might hope to obtain finer proof-size bounds through PV. However, the degree of the polynomial in Cook's translation depends on the PV proof and on the polynomial-time functions it uses [Li25, Chapter 5]. The translations of other theories, such as VTC^0 into TC^0-Frege and VNC^1 into Frege, likewise guarantee only polynomial-size proofs of the translated theorems [CN10]. Fine-grained bounds would require a more precise analysis of the proofs and their translations.

Cryptographic applications. Polynomial bounds, however, are no longer sufficient for emerging applications of EF in cryptography. Jain and Jin [JJ22] showed how to obfuscate circuits whose functional equivalence has short EF proofs, with a security loss independent of the input length, and a growing line of work builds obfuscation and succinct arguments from propositional proofs [JKLV24, JKLM25, JJMP25, HJLM26, JLP26, DHJQ26]. In these constructions, the size of the EF proofs affects the efficiency of the schemes. For example, in the universal SNARGs of [JKLM25], the length of the common reference string and the running time of the prover grow with the length of an EF proof of completeness. Ma, Dai, and Shi [MDS25] constructed indistinguishability obfuscation (iO) whose obfuscation time and evaluation time are $\widetilde{O}_\lambda(N_{\mathrm{circ}} + N_{\mathrm{proof}})$, where N_{circ} bounds the size of the circuits and N_{proof} bounds the size of an EF proof of the equivalence of any two circuits in the family. The overhead of this obfuscator is thus *governed* by the size of the EF proofs.

Moreover, improving the size of EF proofs also has applications to efficient zero-knowledge (ZK) proofs [GMR89] (see [Tha22] for a survey). General-purpose zero-knowledge proofs and SNARKs [Kil92, Mic00, BCCT12, GGPR13, PHGR13, Gro16, GWC19, BSBHR19, Set20] can turn a proof in a formal proof system into a zero-knowledge one, which convinces a verifier of the proved statement without revealing the inference steps of the proof or any private data that the statement refers to, such as a committed circuit. For example, recent work has shown how to compile proofs written in resolution [BN21], SMT, or Lean into zero-knowledge proofs [LAH+22, LOB24, LKA+24]. Using modern SNARK implementations [Tha22], the running time of the ZK prover is roughly proportional to the time it takes to check the validity of the (resolution/SMT/Lean) proof on a Random Access Machine (RAM). For proving facts stated as circuits (e.g.,

that a circuit has certain claimed functionalities), using EF as the formal proof system is a natural choice. In this case, the cost of the ZK prover is roughly proportional to the EF proof size.

Motivated by emerging applications in cryptography, we initiate the study of fine-grained proof complexity for EF. We ask the following fundamental question:

> *For natural mathematical theorems that are known to have polynomial-sized EF proofs, what is the minimal size (e.g. linear, quadratic, or cubic) required to prove them in EF?*

EF, Circuits, and Optimal Size. It is convenient to view EF as a way of writing proofs in the circuit model. An EF proof first uses circuitry to construct all the variables it needs: each extension defines a fresh variable as a constant-size Boolean function of existing variables, that is, as the output of a gate, so every wire of the resulting circuit is a variable. The proof then derives assertions about these wires one at a time. Each new assertion is obtained by applying a constant-size inference rule, substituting wires for the placeholder letters of the rule. Some rules are axioms, which have no premises, such as $P \to (Q \to P)$. Other rules require premises, that is, assertions derived earlier. An example is modus ponens: $P,\ P \to Q \vdash Q$. For instance, suppose that the circuit contains a gate $e := u \vee v$ and that the assertion u has already been derived. The definition of the gate yields $u \to e$ in a constant number of steps, and modus ponens with the substitution $P \mapsto u$ and $Q \mapsto e$ then derives e. The size of an EF proof is the number of its gates and inference steps; Section 2 gives the details.

Optimal proof size. Consider an EF proof of a statement about one or more circuits, expressed as the assertion that a check circuit outputs one on every input. Our hope is to obtain an EF derivation whose size closely matches the circuit size, meaning the total size of these circuits with the check included, say, quasi-linear in it. For a wide class of circuit statements, a derivation of this size is also nearly optimal: consider circuits in which every gate is *pivotal*, that is, for each gate there is an input on which flipping that gate's value, leaving the rest of the circuit unchanged, makes the check output zero. Checking the equivalence of two circuit implementations and checking that a sorting circuit is correct are natural examples of such circuits. In such cases, an EF derivation must use the definition of every gate. Since each inference step uses only a constant number of gate definitions, the derivation therefore has size at least proportional to the circuit size.

As mentioned above, obtaining an EF derivation that is optimal in size would also make the cryptographic applications above optimal up to polylogarithmic factors, whatever statement about their circuits they need to prove, such as the equivalence of two circuits in a family or a property of a committed implementation. In quasi-linear iO [MDS25], for instance, the obfuscation time and the evaluation time would then be optimal up to polylogarithmic factors.

A "hello world" case study. Our community has developed strong intuitions about the circuit complexity of computational tasks. Surprisingly, we have very little intuition about writing proofs in the circuit model with EF. Our paper lays the groundwork for developing such intuition. We give a "hello world" case study, proving elementary school math in EF, and we find that even for statements that have long been part of our second nature, obtaining EF proofs of optimal size is highly non-trivial.

More specifically, consider circuit implementations of the addition and multiplication of n-bit integers, and the basic laws of arithmetic, such as distributivity and associativity. Our question is the following.

> *Do the basic laws of arithmetic have EF proofs whose size matches that of the circuit implementations, up to polylogarithmic factors?*

Note that integer addition and multiplication can be implemented by circuits of quasi-linear size: schoolbook addition uses $O(n)$ gates, and multiplication based on the fast Fourier transform (FFT) [CT65, SS71] uses

$n \log^{O(1)} n$ gates (see [Für09, HvdH21] for faster circuits). Therefore, the question is whether the basic laws of arithmetic for these circuits have EF proofs of size $n \log^{O(1)} n$.

Proof complexity has studied proofs of the laws of arithmetic before. For example, Bonet, Pitassi, and Raz [BPR00] give polynomial-size TC^0-Frege proofs of elementary facts of arithmetic, and Beame and Liew [BL19] give polynomial-size resolution proofs of identities such as commutativity and distributivity for several multiplier circuits. Unlike our question, these works concern polynomial size bounds: they do not consider the fine-grained question of whether the proofs can be quasi-linear in the size of the circuits.

The question may seem simple, but answering it turns out to be highly non-trivial (which partly shows how little we know about fine-grained proof complexity in EF). To obtain optimally sized proofs, we introduce a reusable proof paradigm, which we call *index-generalization*: it turns one derivation about the entry of an array selected by a variable index into assertions about all entries of the array, at the cost of a single copy of the derivation. Below, we state our main theorem; Section 3 presents the paradigm in detail.

1.1 Our Results

Below let ADD represent the schoolbook ripple-carry adder circuit, of size $O(n)$ for n-bit inputs. Let FFTMul represent the textbook FFT multiplier implemented using complex numbers with finite precision, whose size is $n \log^{O(1)} n$ for n-bit inputs. We state our main theorem below.

Theorem 1.1 (Main theorem, informal version of Theorem A.2)**.** *For every $n \geq 1$ and free n-bit inputs x, y, z, the identities*

$$\mathsf{FFTMul}\big(x, \mathsf{ADD}(y,z)\big) = \mathsf{ADD}\big(\mathsf{FFTMul}(x,y), \mathsf{FFTMul}(x,z)\big),$$
$$\mathsf{FFTMul}\big(\mathsf{FFTMul}(x,y), z\big) = \mathsf{FFTMul}\big(x, \mathsf{FFTMul}(y,z)\big),$$
$$\mathsf{FFTMul}(x,y) = \mathsf{FFTMul}(y,x)$$

have EF proofs of size $n \log^{O(1)} n$, and the identities

$$\mathsf{ADD}\big(\mathsf{ADD}(x,y), z\big) = \mathsf{ADD}\big(x, \mathsf{ADD}(y,z)\big),$$
$$\mathsf{ADD}(x,y) = \mathsf{ADD}(y,x)$$

have EF proofs of size $O(n)$.

Thus the size of the EF derivations closely matches the size of the adder and FFT-multiplier circuits: for the two identities of the adder alone the match is up to a constant factor, and for the three identities involving the multiplier it is up to polylogarithmic factors.

1.2 Conceptual Contributions and Open Questions

Fine-grained proof complexity. Our work initiates the study of fine-grained proof complexity for EF, and it opens up many new questions. Can we design optimally sized EF proofs for the correctness of other common computational tasks, for example, sorting networks, matrix multiplication, or cryptographic primitives? Is there a fine-grained counterpart of the propositional translation of PV, that is, a uniform theory whose proofs translate into EF proofs of quasi-linear size? Conversely, can we prove fine-grained lower bounds, showing that some natural statement about circuits of size s requires EF proofs of size much larger than s? Such lower bounds would be interesting even if they are far from superpolynomial.

A language for fine-grained EF proofs. We also introduce a notation for writing EF proofs and accounting for their size (Section 2), which we view as necessary groundwork for this new area. Such a language is timely. Recent cryptographic works rely on EF proofs whose size determines the efficiency of their constructions [JJ22, JKLV24, JKLM25, JJMP25, MDS25], and they expect, for example, that the circuits arising in applications have EF proofs of equivalence whose size is close to the circuit size [MDS25]. As Jin, Kalai, Lombardi, and Mathialagan [JKLM25] note, the expectation that natural cryptographic constructions have EF proofs of correctness must be verified case by case. These works typically describe EF derivations and account for their size in prose, often together with Cook's translation of PV proofs [JKLV24, MDS25]. Prose accounting can be hard to follow, and it does not scale to large proofs, which combine many derivations and reuse their parts; Cook's translation, moreover, gives only polynomial bounds.

Our notation gives a language in which claims about the size of EF proofs can be stated precisely and proven. It is built on *big-step inference rules*, which let us write EF proofs and account for their cost modularly, so that the accounting scales to large proofs. We hope that it can also serve as a bridge to formalizing EF proofs in a programming language or a formal verification tool, where a big-step rule would become a procedure that generates an EF derivation, with its cost as a bound on the size of the output.

2 Definitions and Notations for Extended Frege Proofs

2.1 Extended Frege Proofs

Extended Frege (EF) [CR79] is a propositional proof system that enhances the Frege system (the textbook proof system) with a rule called *extension*. For clarity, we directly define EF, and we defer a more detailed comparison of EF and Frege (and other systems) to Section 8.

Small-step rules: axioms and modus ponens. We assume that formulas are built from Boolean variables using arbitrary unary or binary Boolean operators[1]. We fix a sound and implicationally complete finite set of rules. For example, one may use the constant-size axiom schemes

$$P \to (Q \to P),$$
$$(P \to (Q \to R)) \to ((P \to Q) \to (P \to R)),$$
$$(\neg Q \to \neg P) \to (P \to Q),$$

together with *modus ponens*,

$$P,\ P \to Q \vdash Q.$$

In an Extended Frege (EF) proof, each inference step works by applying one of these small-step inference rules, where the nominal variables P, Q, and R are substituted with concrete instantiations of variables[2] that are either part of the statement to be proven, or derived using an *extension rule* that we will describe below.

We introduce the new terminology *small-step inference rule* to refer to each of the axioms and modus ponens, primarily to distinguish them from the *big-step inference rule* notation to be introduced in Section 2.2, which allows for modular construction of EF derivations, and reusability of proof gadgets already developed.

[1] For the Extended Frege system, the choice of the Boolean operators and the finite set of axiom schemes does not affect asymptotic proof complexity; see, e.g., [Kra19, Lemma 2.4.5].

[2] Extended Frege is often defined to allow substitution of variables in axiom schemes and rules with any formula (rather than only variables). Our definition is more restricted, which is fine as we are proving upper bounds.

Extensions construct circuits. Extended Frege allows the *extension* rule, which lets a proof define a fresh variable as an abbreviation for a formula and then reason with that abbreviation in place of the formula itself. Specifically, for any constant-sized[3] formula $\phi(u_1, \ldots, u_k)$, we may include a proof line of the form

$$e \leftrightarrow \phi(u_1, \ldots, u_k), \tag{1}$$

to name a fresh variable e that has not appeared in previous lines of proof or in ϕ.

While the standard proof complexity literature uses formulas and extension lines to describe variables in EF proofs, in this work, it helps to switch to a *circuit-centered* terminology. We can imagine that every extension rule $e \leftrightarrow \phi$ introduces a new Boolean gate in a circuit. For example, for the extension (1), we may construct a gate whose input wires are $u_1, \ldots, u_k$, whose output wire is named e, and whose gate logic implements the function $\phi(u_1, \ldots, u_k)$. We allow each gate to have *constant fan-in* and *unbounded fan-out*, reflecting the conventions that every extension rule in EF must be constant in size, and that every named variable can be referenced an arbitrary number of times in the proof lines. The constant fan-in also implies that the formula computed by the gate must be constant in size too. With this one-to-one correspondence, the number of gates in the "proof circuit" corresponds to the number of extension rules in the EF proof.

2.2 Big-Step Rules: New Language for Modular Construction of EF Proofs

To support modular construction of EF proofs and reusability of proof gadgets, we introduce a new language for describing EF proofs. This new language can also potentially serve as a bridge if one chooses to encode and mechanically verify EF proofs using formal methods.

Overview of the proof growth process. As shown in Figure 1, developing an EF proof can be thought of as iteratively applying the following process. In each step of the proof, our starting point is a *context* (shown as the expanded context $\widehat{\Gamma}_\phi$ in Figure 1), which includes

1. circuitry that either computes the conclusion variables, or was introduced in previous proof steps, along with all wires defined by any such circuitry and all raw input wires to the circuitry; and
2. a list of *assertions* of wires (or variables) that are already known to be true. Each assertion can either be previously derived or can be a supplied hypothesis (assumption).

Under such a context, each proof step can take one of two actions, and each action will grow the context correspondingly:

1. *Defining a new gate and output variables:* construct a new gate whose inputs are wires that already exist in the context, and explicitly name its outputs; or
2. *Small-step inference:* append a new assertion to the list of assertions, by applying one of the small-step inference rules, and substituting concrete wires from the context for the nominal variables in the rule.

Under this view, the number of proof steps is equivalent to the proof size (up to a constant factor).

[3] In this paper, we focus on *constant-width* EF, where extension lines are constant in size. Ordinary EF does not impose this constant-width restriction, and correspondingly, the length of each proof line must be charged to the proof size. Considering constant-width EF is without loss of generality since we focus on showing upper bounds. Furthermore, for known cryptography applications of EF, assuming constant-width EF is also without loss of generality [JJ22, JKLV24, HJLM26, MDS25].

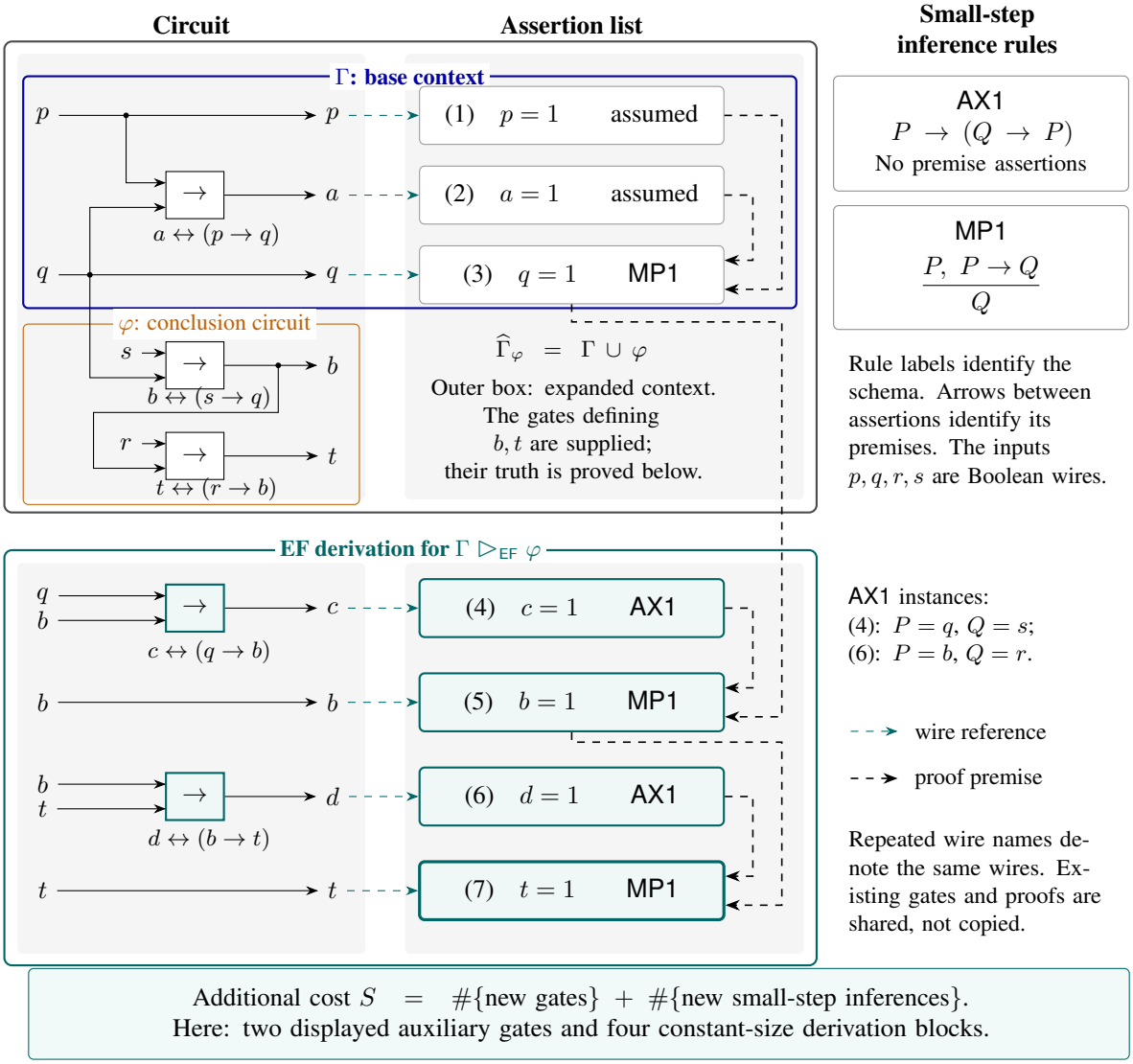


Figure 1: An EF derivation of $\Gamma \rhd_{\mathsf{EF}}^{S} \varphi$, meaning that the output t of the conclusion circuit φ is proved to be 1. The blue box marks the base context Γ: the gate defining a and assertions (1)–(3), together with the declared inputs p, q, r, s. The orange box is φ, the two-gate circuit defining the target assertion t computed as $r \to (s \to q)$. The outer upper box is the expanded context $\widehat{\Gamma}_\varphi = \Gamma \cup \varphi$ (Section 2.2). The lower part adds gates and assertions to derive $t = 1$, where t is the target assertion $r \to (s \to q)$. Labels name the underlying small-step inference rules; arrows give their premises.

Big-step inference rule, base and expanded contexts. We now introduce a big-step inference rule which embodies how we want to support modular construction of EF proofs. At a very high level, a big-step inference rule allows us to apply many small proof steps as a bundle in one shot, where each proof step either introduces new variables through a new gate definition, or applies a small-step inference rule. The big-step notation therefore supports the description of a reusable proof gadget, and the exact conditions under which the proof gadget can be applied.

Concretely, let $\Phi = [\varphi_j]_{j<r}$ denote a finite list of single-output Boolean circuits (called *conclusion*

circuits). A *big-step inference rule* has the form

$$\Gamma : \underbrace{\mathcal{R}}_{\text{optional}} \ \rhd^{S}_{\mathsf{EF}} \ [\varphi_j]_{j<r}.$$

To understand what it means, first, ignore the optional suffix $\mathcal{R}$ and consider the basic form $\rhd^{S}_{\mathsf{EF}} \ [\varphi_j]_{j<r}$. The rule should be read as follows:

- **Base and expanded contexts.** Here the full context $\widehat{\Gamma}_\Phi := \Gamma \cup \Phi$, henceforth called the *expanded context*, consists of the *base context* Γ plus the additional variables introduced by the conclusion circuits $[\varphi_j]_{j<r}$. Differentiating between the base context Γ and the expanded context $\widehat{\Gamma}_\Phi$ is for notational convenience and clarity: it allows the base context to not contain the new variables introduced by the conclusion circuits Φ when writing big-step rules.
- **Semantics of the rule, and additive cost accounting.** The rule states that starting from the *expanded context*, we can apply S steps of reasoning to obtain the assertions described by Φ, that is, every circuit in Φ outputs 1. As mentioned, each step of reasoning either introduces new variables through a new gate definition, or applies a small-step inference rule. In other words, the bound S records the total additive cost, measured as below:

  $$\#\{\text{new gate definitions}\} + \#\{\text{new small-step inferences}\} \leq S. \tag{2}$$

  The accounting is therefore additive: nothing already in the expanded context is charged to S, including the conclusion circuits.

Since our formulation is equivalent to constant-width EF in which all proof lines are constant in size, it follows that our cost accounting is asymptotically equivalent to that of EF — the latter charges the total length of all proof lines.

Optional designated assertions. Finally, the optional suffix $: \mathcal{R}$ highlights a finite list of assertions already available in Γ. Each assertion may be previously derived, with its derivation recorded in Γ, or an externally supplied assumption. The colon explicitly designates the assertions that will be used as premises to apply this big-step rule. Other assertions may also exist in the context Γ which are not explicitly listed. An empty list $\mathcal{R}$ means that the derivation needs no assumption beyond the circuit definitions; we omit "$: \mathcal{R}$" when no assertions are needed.

Applying a big-step inference rule. A big-step inference rule refers to abstract variables and an abstract context containing these variables. The EF proof of a big-step inference rule also refers to these abstract variables, and as mentioned, such a proof contains auxiliary circuitry that introduces new variables as well as applications of small-step inference rules that introduce new assertions.

During a concrete application of the big-step rule, the abstract variables are substituted with actual variables in the concrete base context. The application proceeds in the following two steps, growing the context in the process:

1. *Prepare the expanded context.* Grow the base context with the conclusion circuitry (with proper substitutions), as well as any new variable introduced by the conclusion circuitry. Recall that the cost of this step is not included in the displayed cost of the rule itself; it should be charged to the caller who invokes the big-step rule.

2. *Copy proof steps with substitution.* Copy all abstract proof steps from the proof of the big-step rule itself, where abstract variables explicitly referred to in the statement of the rule must also go through the same substitution, and any auxiliary new variable introduced during these proof steps can acquire an arbitrary unique name when transferred to the concrete context. These proof steps grow the concrete context with new circuitry, variables, and assertions that result from the application of small-step inference rules.

If the big-step rule being applied has explicit requirements on the base context Γ or requires some explicitly stated assumption set $\mathcal{R}$, then the application is only possible if the concrete context satisfies these requirements after the variable substitution. Specifically, for an assumption set $\mathcal{R}$, the concrete context must already contain the circuitry that outputs the variables asserted to be true by $\mathcal{R}$, and the corresponding assertions must already exist in the list of assertions.

Conventions for writing big-step rules. When stating a big-step inference rule, unless explicit requirements on the base context Γ are declared, by default, we only require that the variables supplied as inputs to the conclusion circuits be wires in the context Γ. Beyond these variables, Γ may contain arbitrary unrelated circuitry and assertions, and the variables (explicitly named in the rule or not) may have arbitrary relationships between them — the big-step inference rule holds regardless. When the big-step inference rule holds under explicit conditions or requirements, we always state these requirements explicitly, either through the designated premises $\mathcal{R}$ or in the surrounding text introducing the rule.

Families of big-step inference rules. We often use this notation for a family of big-step inference rules parameterized by an input length. For example, for each fixed width $w \geq 1$, let x, y, z be integers each represented by w bits. An integer distributivity rule has the form

$$\Gamma \ \rhd_{\mathsf{EF}}^{S(w)} \ x(y+z) = xy + xz.$$

The circuits implementing the arithmetic operations and equality checks are defined in Appendix A. Importantly, the circuitry depends on w, as does the additional derivation cost $S(w)$. The width w is a fixed parameter of the circuit instance.

3 New Proof Paradigm: Index-Generalization

In this section, we present index-generalization, the novel proof paradigm behind our quasi-linear-size EF proofs. We use distributivity as an example to illustrate the capabilities of this technique.

The natural proof is quadratic. Recall that we want to prove the distributivity rule $\mathsf{FFTMul}\big(x, \mathsf{ADD}(y,z)\big) = \mathsf{ADD}\big(\mathsf{FFTMul}(x,y), \mathsf{FFTMul}(x,z)\big)$, where ADD is the schoolbook adder, and FFTMul is the FFT-multiplier. Specifically, $\mathsf{FFTMul}(x,y)$ works by first applying a forward FFT on the two integer vectors representing the digits of x and y respectively, then performing a coordinate-wise multiplication, and then applying the inverse FFT transform.

Unfortunately, the most natural way to prove distributivity incurs at least quadratic cost. To convey the intuition, we briefly describe one natural approach below while deferring the details of the naïve proof to Section 6.2. The most natural way to prove distributivity is to first prove the following convolution theorem:

After rounding, the k-th output of the FFT convolution of two integer vectors a, b is equal to $\sum_j a_j b_{k-j}$.

With some effort, one can show that for each *fixed* coordinate k, the convolution theorem has an EF proof of size $N \log^{O(1)} N$. In fact, since the k-th output depends on all N inputs, the EF proof size for a fixed k must

be at least linear in N. Therefore, proving the convolution theorem separately for each of the N coordinates yields a proof of size $N^2 \log^{O(1)} N$.

Indeed, not only for proving the convolution theorem, the naïve approach of repeating an EF proof separately for each coordinate would encounter a similar quadratic curse when proving other key identities between arrays.

The index-generalization paradigm. It would be ideal if we could unify the EF proofs for all N coordinates into one, as the N coordinatewise proofs are essentially identical: they differ only in the hardwired constant k. However, the problem is that a different choice of k leads to different wiring in the proof circuitry. With EF, a derivation proves its conclusion only about the specific wires it mentions, and the choice of those wires needs to be predetermined. Thus, obtaining the same proof at another coordinate requires instantiating it again on other wires, and its cost is charged again.

To avoid making N copies, our idea is to write the EF derivation once, at a *variable* coordinate. For example, given a vector $\boldsymbol{A} = (A_0, \ldots, A_{N-1})$, suppose we want to check that some predicate $P(A_i)$ holds for every $i < N$. To do so, we first prove the following statement which checks the predicate to be true at a variable index $i < N$:

$$\text{[Selector form:]} \qquad P(\mathsf{Sel}(\boldsymbol{A}, i)) \qquad \text{where } \mathsf{Sel} \text{ is an } N\text{-way selector circuit} \tag{3}$$

Since i is a variable, if $P(\mathsf{Sel}(\boldsymbol{A}, i))$ is a valid assertion, then intuitively it should work at every fixed index i. However, the intuition does not immediately tell us a way to efficiently expand an EF proof for the assertion $P(\mathsf{Sel}(\boldsymbol{A}, i))$ into a proof for all N assertions at all indices, that is,

$$\text{[Expanded form:]} \qquad P(A_0), P(A_1), \ldots, P(A_{N-1}) \tag{4}$$

As a key technical contribution, we devise a new proof paradigm called index-generalization, which allows us to pass from the variable-selector form (3) to the expanded form (4) without having to make N copies of the proof of (3). We formally describe the index-generalization paradigm in Lemma 3.4.

A reusable paradigm. The index-generalization paradigm is very useful for writing compact EF proofs. Indeed, our proofs of basic laws of arithmetic apply index-generalization several times, and in a nested fashion. For the case of distributivity, Section 7.2 and Figure 6 of the informal technical roadmap give an overview of what each application achieves, and of the nesting structure of these applications.

The remainder of this section will formally describe the index-generalization paradigm and provide detailed proofs. Section 3.1 describes the basic circuitry needed and states some basic inference rules. Section 3.2 then states index-generalization and discusses its premise and its use, and Section 3.3 proves it.

3.1 Building Blocks

We first introduce some basic circuit constructions and big-step inference rules. Here we stick only to the preliminaries that are necessary for describing and proving the index-generalization paradigm. The subsequent informal technical roadmap and formal technical sections will introduce additional circuitry and big-step inference rules that are necessary for proving our main objective, basic laws of arithmetic.

Boolean trees. Let m be a power of 2, and fix $m \geq 2$ Boolean wires $P_0, \ldots, P_{m-1}$. Throughout the paper, we use $\bigwedge_{j<m} P_j$ to denote the outcome of an AND-tree whose leaves are $P_0, \ldots, P_{m-1}$. Specifically, build a balanced binary tree of height $O(\log m)$ with $P_0, \ldots, P_{m-1}$ as leaves, and each non-leaf node computes the AND of its two children. The notation $\bigvee_{j<m} P_j$ is defined similarly, with the AND-tree replaced by an OR-tree.

Unfolding of an OR-tree starts with the assertion that $\neg \bigvee_{j<m} P_j$, and derives in EF the m separate assertions $\neg P_j$ for every $j < m$. We state the big-step rule for unfolding of an OR-tree below, and defer its full proof to Lemma B.3 of Appendix B. Unfolding of an AND-tree can also be similarly defined, but we omit it in this section since it is not needed for proving index-generalization.

Lemma 3.1 (Unfolding an OR tree)**.** *Let Γ contain the gate definitions of the specified OR tree. Then*

$$\Gamma : \neg \left(\bigvee_{j<m} P_j \right) \rhd_{\mathsf{EF}}^{O(m)} [\neg P_j]_{j<m}.$$

Following the convention in Section 2.2, the premise is a circuit in Γ (implemented with an OR-tree and a top negation gate), and the conclusion circuit is supplied in the implied expanded context. The big-step inference rule states that in the expanded context, we can prove $\neg P_0, \ldots, \neg P_{m-1}$ (i.e. the output wires of the circuits) by writing $O(m)$ additional EF proof lines.

Equality propagation. A commonly used big-step rule is called equality propagation. Intuitively, it says that feeding equal inputs to the same circuit produces equal outputs, and this can be proven in EF incurring cost proportional to the circuit size (see also [Li25, Remark 5.3.1]). The proof of index-generalization needs a conditional form of this rule, in which the inputs are equal bit by bit under an antecedent H, and this conditional version is stated in the following lemma:

Lemma 3.2 (Conditional equality propagation)**.** *Let x, y be w-bit strings, let H be a Boolean wire, and let C be a Boolean circuit with one output bit and $|C|$ gates. The copies $C(x, z)$ and $C(y, z)$ have the same gate structure and share all other input wires z. Then*

$$\Gamma : \big[H \to (x_t \leftrightarrow y_t)\big]_{t<w} \rhd_{\mathsf{EF}}^{O(|C|)} H \to \big(C(x, z) \leftrightarrow C(y, z)\big).$$

We defer the proof of Lemma 3.2 to Lemma B.4 of Appendix B, which proves the general form (54) that this lemma specializes.

Array selectors. A word multiplexer $\mathsf{Mux}(c, x, y)$ selects x when $c = 1$ and y when $c = 0$: its output bit j is $(c \wedge x_j) \vee (\neg c \wedge y_j)$, so it has $O(w)$ gates on w-bit words. For an array $\boldsymbol{A} = (A_0, \ldots, A_{m-1})$ of $m = 2^s$ words of width $w \geq 1$ and an s-bit address i, let $\mathsf{Sel}_{m,w}(\boldsymbol{A}, i)$ select entry A_i. It is implemented as a balanced binary tree of word multiplexers. The most significant address bit selects the lower half when it is zero and the upper half when it is one; the remaining bits select recursively within that half. The circuit has s multiplexer levels and $O(mw)$ gates. For $m = 1$, its output is the word A_0 itself. Bit t of $\mathsf{Sel}_{m,w}(\boldsymbol{A}, i)$ is computed by a one-bit selector $\mathsf{Sel}_{m,1}$ on the t-th bits of the entries, and these w one-bit selectors share the address bits. We omit the subscripts when the array length and entry width are clear.

Applying a check to every entry of an array and then selecting one of the results gives the same outcome as selecting the entry first and then applying the check. The proof of index-generalization uses this fact at an address that is computed in the proof.

Lemma 3.3 (Selection commutes with a check)**.** *Let $\boldsymbol{A} = (A_0, \ldots, A_{m-1})$ be an array of $m = 2^s$ words of width w, let K be an s-bit address, and let P be a single-output Boolean check on w-bit words, of circuit size $|P|$. Put $P_j := P(A_j)$. Then*

$$\Gamma \rhd_{\mathsf{EF}}^{O(m(|P|+w))} P(\mathsf{Sel}(\boldsymbol{A}, K)) = \mathsf{Sel}\big((P_j)_{j<m}, K\big).$$

The above lemma is later further generalized to allow multiple input arrays — see Lemma B.7 of Appendix B. We therefore defer its full proof to Appendix B.

3.2 The Index-Generalization Lemma

A context Γ' *extends* Γ, written $\Gamma' \supseteq \Gamma$, if it contains all declared inputs, circuits, and assertions of Γ, possibly together with further ones.

Lemma 3.4 (Index-generalization)**.** *Let $\boldsymbol{A} = (A_0, \ldots, A_{m-1})$ be an array of $m = 2^s$ words of width w available in the context Γ. Let P be a single-output Boolean check on w-bit words, of circuit size $|P|$, and put $P_j := P(A_j)$. If for every context $\Gamma' \supseteq \Gamma$ and every ordered tuple of s wires in Γ', read as an s-bit index and denoted i,*

$$\Gamma' \ \rhd^{S}_{\mathsf{EF}} \ P(\mathsf{Sel}(\boldsymbol{A}, i)) = 1, \tag{5}$$

then we have that

$$\Gamma \ \rhd^{S+O(m(|P|+w+s))}_{\mathsf{EF}} \ [P_j = 1]_{j<m}. \tag{6}$$

In the above lemma, because Γ' can be an arbitrary extension of the context Γ, and i can be an arbitrary ordered tuple of s wires in Γ', essentially $P(\mathsf{Sel}(\boldsymbol{A}, i))$ must be true for an i that can take on any arbitrary value. Therefore we can derive all of the m assertions $P_j := P(A_j)$ where $j < m$. The proof size, however, is nontrivial: the lemma states that simultaneously deriving in EF all of the m assertions $P_j := P(A_j)$ where $j < m$ incurs only the cost S corresponding to a single copy of the proof, plus some additive terms that do not depend on S.

Note also that the universal quantification over the extended context Γ' and the choice of the index wires i is important for Lemma 3.4: for a specific $\Gamma' \supseteq \Gamma$, and a specific choice of the i wires, it could be that Γ' imposes some restriction on i's value, e.g., an already-proven assertion can require i's value to be 0. The index-generalization does not apply when such restrictions are imposed on i.

3.3 Proof of Index-Generalization

Proof idea. Instead of applying the premise at each constant index j, we apply it once, at the outcome of a single circuit K that computes a potential counterexample called a witness, an index that points to a failing entry whenever one exists. The proof works as follows.

1. *The witness property.* Let $\beta_j := \neg P_j$ (called failure flags) and their disjunction $B := \bigvee_{j<m} \beta_j$. We construct a circuit K (whose input is wired to $\beta_0, \ldots, \beta_{m-1}$) and derive $B \to \neg P(\mathsf{Sel}(\boldsymbol{A}, K))$: if some entry fails, then the entry selected by K fails.

2. *One application of the premise.* The premise holds at every index of every extension of Γ, in particular at K (in the expanded context Γ'' that contains K and B). This gives a derivation of $P(\mathsf{Sel}(\boldsymbol{A}, K)) = 1$ of size at most S.

3. *The conclusion.* These two parts conclude $\neg B$; unfolding it gives $P_j = 1$ for every j.

We stress that K is computed dynamically on the fly, so it is not known in advance at the time when the EF proof is written down.

Proof of Lemma 3.4. At a high level, we compute K along a balanced binary tree over the positions: every internal node passes up the witness of its left subtree if that subtree contains a failure, and the witness of its right subtree otherwise (Figure 2). Correspondingly, we derive the witness property by a structural induction over the tree: at each node, we derive that if the node's subtree contains a failure, then the node's witness points to one (Figure 3).[4]

More concretely, write $\varphi_i := P(\mathsf{Sel}(\boldsymbol{A}, i))$ for the conclusion circuit of the premise, including its selector; it has $O(mw + |P|)$ gates. The coordinate checks P_j belong to the expanded context, so they are not charged.

[4]Note that the induction happens in *meta-theory*; that is, we construct a certain circuit or EF proof inductively. The EF proof can be viewed as the "unrolled" version of an inductive mathematical proof.

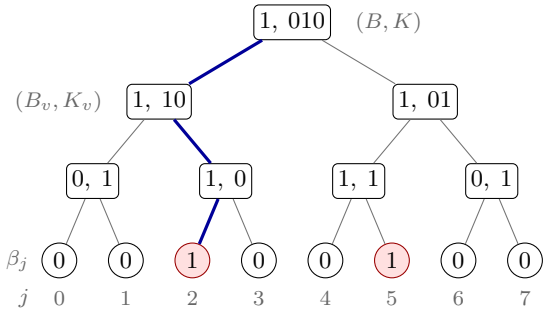


Figure 2: The witness circuit for $m = 8$ entries, of which entries 2 and 5 fail (shaded leaves, $\beta_j = 1$). Each internal node v shows the pair (B_v, K_v) of its failure bit and its local witness, an index into the interval of v in binary; at the root this pair is written (B, K). A node passes up the witness of its left child, prefixed by 0, if the left half of its interval contains a failure, and the witness of its right child, prefixed by 1, otherwise. The root's witness $K = 010$ points to entry 2, along the thick path.

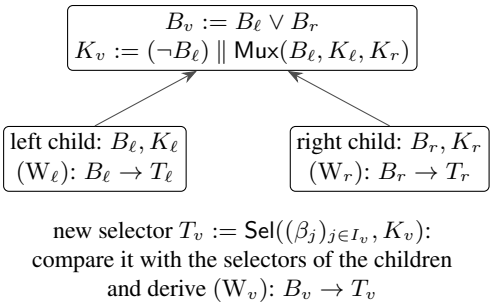


Figure 3: One inductive step of Step 2 of the proof, at a node v. The arrows carry the failure bits and local witnesses of the children, whose claims (W_ℓ) and (W_r) are the induction hypotheses. The new selector T_v checks the witness for the whole interval of v, and its comparison with the selectors of the children has cost proportional to the size of that interval.

Step 1: the witness circuit. Introduce the failure flags $\beta_j := \neg P_j$, at cost $O(m)$. Build a balanced binary tree over the leaves $\beta_0, \ldots, \beta_{m-1}$. A node v at height h covers an interval I_v of 2^h consecutive positions. For each node v, we construct a failure bit B_v, which indicates whether I_v contains a failure, and an h-bit local witness K_v, an index into I_v that is meant to point to a failure. Thus there is a separate witness at each node, and the witness at the root is the final index K. At a leaf j, let $B_v := \beta_j$, and let K_v be the empty index into the one-element interval $\{j\}$. At an internal node v with left child ℓ and right child r, let

$$B_v := B_\ell \vee B_r, \qquad k_v := \mathsf{Mux}(B_\ell, K_\ell, K_r), \qquad K_v := (\neg B_\ell) \parallel k_v,$$

where $\parallel$ places the bit $\neg B_\ell$ in the most significant position. If the left half of I_v contains a failure, then K_v points into the left half, at the left child's witness; otherwise, it points into the right half, at the right child's witness. When neither half contains a failure, the value of K_v does not matter. At the root, put $B := B_{\mathrm{root}}$ and $K := K_{\mathrm{root}}$. The bits B_v form a balanced OR tree with leaves β_j, so $B = \bigvee_{j<m} \beta_j$. A node at height h has $O(h)$ gates, so the witness circuit has $O(\sum_{h=1}^{s} (m/2^h)\, h) = O(m)$ gates.

Step 2: the witness property along the tree. For each node v at height h, introduce the one-bit output

$$T_v := \mathsf{Sel}_{2^h,1}\big((\beta_j)_{j \in I_v}, K_v\big),$$

the failure flag at the position of I_v that K_v selects. For $h > 0$, this is a new selector circuit with $O(2^h)$ gates, whose address input is the word K_v. The claim at node v is

$$B_v \to T_v, \tag{W_v}$$

which states that if I_v contains a failure, then the witness K_v points to one. At the root, $(\mathrm{W}_{\mathrm{root}})$ is $B \to T$ for $T := T_{\mathrm{root}} = \mathsf{Sel}((\beta_j)_{j<m}, K)$. We prove the claims at all nodes by induction on the height; the actual EF proof is constructed by unrolling the induction over the tree, from the leaves up: it derives the claims at the m leaves, and then, for $h = 1, \ldots, s$ in turn, performs the inductive step at every node of height h, citing the claims already derived at its two children.

Base case. At a leaf j, the size-one selector is the wire β_j itself, so $T_v = B_v = \beta_j$, and (W_v) is the tautology $\beta_j \to \beta_j$.

Inductive step. Let v be a node at height $h \geq 1$ with left child ℓ and right child r, and assume the claims at the children,

$$B_\ell \to T_\ell, \qquad B_r \to T_r,$$

which are the induction hypotheses (W_ℓ) and (W_r) (Figure 3). We first derive three facts about the circuits at v, none of which uses the induction hypotheses.

- *The top multiplexer.* The selector computing T_v consists of two half-size selectors, which select within I_ℓ and I_r using the low $h-1$ bits k_v of K_v,

  $$U_\ell := \mathsf{Sel}\big((\beta_j)_{j\in I_\ell}, k_v\big), \qquad U_r := \mathsf{Sel}\big((\beta_j)_{j\in I_r}, k_v\big),$$

  followed by a multiplexer controlled by the top bit $\neg B_\ell$ of K_v. The definitions of this multiplexer and of the negation gate give $T_v \leftrightarrow \mathsf{Mux}(B_\ell, U_\ell, U_r)$ by a constant-size derivation.

- *The left comparison.* The half-size selectors select at k_v, while the selectors T_ℓ and T_r of the children select at K_ℓ and K_r. For every bit position t, the definition of bit t of the multiplexer k_v gives $B_\ell \to \big((k_v)_t \leftrightarrow (K_\ell)_t\big)$ by a constant-size derivation. The wires U_ℓ and T_ℓ are the outputs of two copies of the same selector circuit, with the same data $(\beta_j)_{j\in I_\ell}$ but with the addresses k_v and K_ℓ. Conditional equality propagation (Lemma 3.2), applied to this selector with antecedent B_ℓ and these bitwise premises, therefore gives $B_\ell \to (U_\ell \leftrightarrow T_\ell)$.

- *The right comparison.* Symmetrically, the multiplexer gives $\neg B_\ell \to \big((k_v)_t \leftrightarrow (K_r)_t\big)$ for every t, and the same lemma, with antecedent $\neg B_\ell$, gives $\neg B_\ell \to (U_r \leftrightarrow T_r)$.

For $h = 1$, the half-size selectors are the leaf wires themselves, so $U_\ell = T_\ell$ and $U_r = T_r$, and the two comparisons hold trivially.

We now derive (W_v) from the two induction hypotheses, the three facts above, and the definition of the OR gate B_v. The conjunction of these six formulas implies (W_v):

$$\left.\begin{array}{ll} (\mathrm{W}_\ell) & B_\ell \to T_\ell \\ (\mathrm{W}_r) & B_r \to T_r \\ \text{top multiplexer} & T_v \leftrightarrow \mathsf{Mux}(B_\ell, U_\ell, U_r) \\ \text{left comparison} & B_\ell \to (U_\ell \leftrightarrow T_\ell) \\ \text{right comparison} & \neg B_\ell \to (U_r \leftrightarrow T_r) \\ \text{OR gate} & B_v \leftrightarrow (B_\ell \vee B_r) \end{array}\right\} \implies B_v \to T_v.$$

This implication is a tautology of constant size, and thus (W_v) follows from the six formulas by a constant-size derivation.

Cost. At a node of height h, the selector T_v costs $O(2^h)$, the bitwise facts about k_v cost $O(h)$, and the two comparisons cost $O(2^h)$ by Lemma 3.2; the rest of the inductive step costs $O(1)$. The step pays for selectors as large as the interval of v, and cites the claims at the two children. Summed over the $m/2^h$ nodes at each height h, Step 2 costs

$$O\Big(m + \sum_{h=1}^{s} \frac{m}{2^h}\big(2^h + h\big)\Big) = O\big(m(s+1)\big).$$

Step 3: applying the premise. For an s-bit index x, write $\varphi_x := P(\mathsf{Sel}(\boldsymbol{A}, x))$ for the conclusion circuit of the premise at x, and let $p := \varphi_K$; it has $O(mw + |P|)$ gates. Recall that the failure flags $\beta_j = \neg P_j$ are the copies of the check $\neg P$ at the entries of $\boldsymbol{A}$. Since selection commutes with this check (Lemma 3.3), we have

$$\neg p = \mathsf{Sel}\big((\beta_j)_{j<m}, K\big) = T$$

at cost $O(m(|P| + w))$, including the circuit p and its negation. This turns the claim $(\mathrm{W}_{\mathrm{root}})$, which is $B \to T$, into the witness property $B \to \neg p$.

Let Γ'' consist of Γ together with the circuits and derived assertions from above. It extends Γ, and the output wires of K form an s-bit index among its wires. So the premise, applied in Γ'' at the index K, gives

$$\Gamma'' \ \rhd^{S}_{\mathsf{EF}} \ p = 1.$$

Its conclusion circuit φ_K is the circuit p of Step 3, which is already present, so this application costs at most S. Thus the derivation at an arbitrary index is applied at the computed index K, without knowing its value, and every use of the index shares the circuit computing K.

Note that we have derived $B \to T$, $T \leftrightarrow \neg p$, and p. A constant-size derivation gives $\neg T$ and then $\neg B$. Since the bits B_v form an OR tree with output B and leaves β_j, unfolding (Lemma 3.1) gives the list $[\neg\beta_j]_{j<m}$, at cost $O(m)$. Each definition $\beta_j \leftrightarrow \neg P_j$ then gives $P_j = 1$ by a constant-size derivation. This establishes every entry of the conclusion list.

Total cost. Step 1 costs $O(m)$, Step 2 costs $O(m(s+1))$, Step 3 costs $O(m(|P| + w)) + S + O(m)$, where $O(m(|P| + w))$ is the cost to prove $B \to T$, S is the cost from the application of the premise, and the last term $O(m)$ is from unfolding the OR tree. The total is therefore $S + O(m(|P| + w + s))$. □

Informal Technical Roadmap

In the following few sections, we will give an informal overview of our techniques. Specifically, suppose that we implement addition using schoolbook addition and multiplication using FFT. Thus, addition of N-bit integers incurs $O(N)$ and multiplication of N-bit integers incurs $N \log^{O(1)} N$ circuit size. We want to prove in EF the basic laws for addition and multiplication, including associativity and distributivity, and we ask whether an EF derivation size of $N \log^{O(1)} N$ is possible. Note that the derivation size must be at least the circuit size since the EF proof must at least read the circuit implementation. Thus, if we can show an $N \log^{O(1)} N$-sized EF derivation, it must be optimal up to polylogarithmic factors.

In this roadmap, we will take distributivity as a primary example to illustrate the challenges and our new techniques.

4 Basic Warmup

Before turning to our main example, we work through EF derivations of a few selected facts. Even very simple arithmetic facts take some getting used to in EF, and the derivations below introduce patterns that recur throughout the paper. Several of them are also used directly later. At the end of the section, we state further reusable inference rules that the rest of the roadmap uses (Section 4.4).

We use the notation of Section 2. A derivation starts from a context Γ, which contains circuits, in which every wire is a variable, and a list of available assertions, each stating that some wire carries the value one. The derivation adds new gates and small-step inferences, and $\Gamma \rhd^{S}_{\mathsf{EF}} \varphi$ states that it establishes that the check circuit φ outputs one, using at most S new gates and inferences. Our derivations are written down by an algorithm, the proof generator, separately for each input length, and the arguments below describe what it writes.

4.1 A First Example: Commutativity of Addition

Let x, y be integers represented by w-bit words, and let ADD be the schoolbook adder. Starting from the carry $\gamma_0 := 0$, at column j it computes the sum bit $s_j := x_j \oplus y_j \oplus \gamma_j$ and the carry $\gamma_{j+1} := \mathrm{MAJ}(x_j, y_j, \gamma_j)$, the majority of the three bits. Its output has $w + 1$ bits: the sum bits $s_0, \ldots, s_{w-1}$ and, as the top bit, the final carry $s_w := \gamma_w$. We show

$$\Gamma \ \rhd^{O(w)}_{\mathsf{EF}} \ \mathsf{ADD}(x, y) = \mathsf{ADD}(y, x).$$

Following the convention (see Section 2.2), both adders are circuits in the context, so their wires are variables. Write s'_j and γ'_j for the sum bits and carries of $\mathsf{ADD}(y, x)$. The conclusion $\mathsf{ADD}(x, y) = \mathsf{ADD}(y, x)$ is implemented by the equality check EQ: for each output bit, one gate computes the equivalence $s_j \leftrightarrow s'_j$, and a balanced tree of AND gates combines these bits into a single output (Appendix B). The two adders output the same bits because they have the same carries, and the derivation follows this reasoning column by column. At column j, the implication

$$\gamma_j = \gamma'_j \ \rightarrow \ \big(s_j = s'_j \text{ and } \gamma_{j+1} = \gamma'_{j+1}\big) \tag{7}$$

follows from the definitions of the gates in column j by a tautology over the wires $x_j, y_j, \gamma_j, \gamma'_j$, so it has a derivation of constant size. The derivation works by unrolling an induction over the columns. Starting from $\gamma_0 = \gamma'_0 = 0$, it applies the implication for $j = 0, 1, \ldots, w - 1$ in turn, each time citing the equality $\gamma_j = \gamma'_j$ derived at the previous column, at total cost $O(w)$. Finally, we derive from the leaves up that every gate of the AND tree outputs one, so that the equality check outputs one, again at cost $O(w)$ (Lemma B.2). This

proves the commutativity-of-addition conclusion of Theorem A.2, which we will not repeat in the formal technical sections.

To prove Equation (7) in each step, we use the following property called *column equation*:

$$x_j + y_j + \gamma_j = s_j + 2\gamma_{j+1}. \tag{8}$$

For each j, the equation concerns only the wires of column j, so it has a derivation of constant size from their gate definitions available in the context.

4.2 Reordering Summation

Long sums over a list of numbers occur throughout circuits and proofs, e.g., the column sums of schoolbook multiplication and the pairings below. Let $x_0, \ldots, x_{m-1}$ be integers represented by w-bit words, with $m = 2^s$ a power of two. A sum circuit adds them along the balanced binary tree with m leaves in index order, obtained by splitting each list of leaves into consecutive halves, and we write $\sum_{j<m} x_j$ for its output. For simplicity, all additions in this subsection are performed at the common width $w + s$, at which no partial sum overflows. Also, it is well-known that basic arithmetic rules on w-bit numbers can be proved in $w^{O(1)}$ size (see, e.g., [Bus86, Kra95]). When the positions $a, \ldots, b$ are the leaves of a subtree, we write $\Sigma[a, b]$ for the output of that subtree; for example, $\Sigma[4, 5] = x_4 + x_5$ is the output of one of its nodes, and $\Sigma[j, j] = x_j$.

We show that there is a short EF proof that a permutation does not change the summation. Formally, for an arbitrary fixed permutation π of $\{0, \ldots, m-1\}$,

$$\Gamma \;\rhd^{m \log^{O(1)}(m+1)\, w^{O(1)}}_{\mathsf{EF}} \; \sum_{j<m} x_{\pi(j)} = \sum_{j<m} x_j. \tag{9}$$

This rule is what lets a derivation split a sum into two half-sums, or merge two half-sums into one. Since the balanced tree on $2k$ leaves is the addition above the balanced trees on the two halves of its leaf list, the sum $\sum_{j<k} x_j + \sum_{k\le j<2k} x_j$ is itself the balanced tree on those $2k$ leaves. A split such as

$$\sum_{j<m} x_j = \sum_{j<m/2} x_{2j} + \sum_{j<m/2} x_{2j+1}$$

is therefore the instance of (9) for the permutation that lists the even positions before the odd ones, and every application of the rule in this roadmap is of this form.

The textbook proof of the fact, which decomposes the permutation using swaps of adjacent terms and then applies associativity and commutativity, is not efficient enough. A permutation is a product of such swaps, but it may need $\Theta(m^2)$ of them. Moreover, a swap takes more than one step in EF: the two terms must first be brought together by associativity, and the equality that each move creates at one node must then be carried to the root through every ancestor, one adder at a time.

The construction below instead exchanges two arbitrary terms directly, so that $m - 1$ exchanges suffice, and it realizes each exchange by $O(\log m)$ moves along the two paths from those terms to their lowest common ancestor.

Exchange two leaves of the balanced tree. A permutation is a product of at most $m - 1$ transpositions, so it suffices to swap two leaves. A swap changes only the two paths from these leaves to their lowest common ancestor, which have length $O(\log m)$, and it proceeds in three phases (Figure 4). First, it exposes each leaf: along the leaf's path, associativity and commutativity move the leaf to the front of each sum, until the child of the lowest common ancestor on that side adds the leaf to the sum of all the subtrees beside the path. Second,

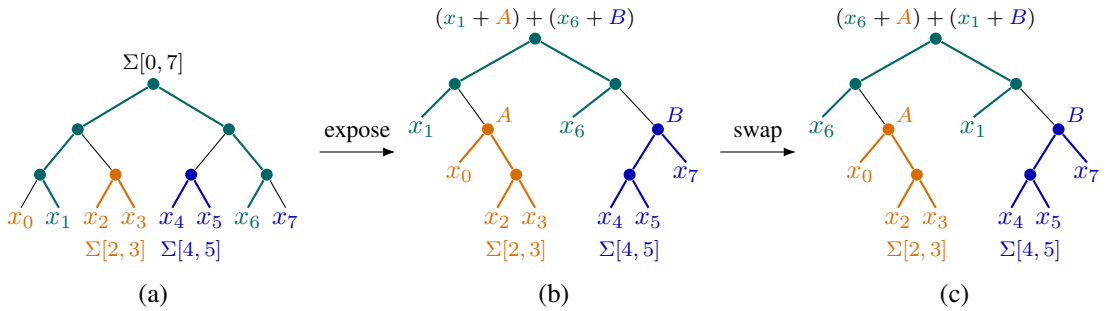


Figure 4: Swapping the leaves x_1 and x_6 of the balanced tree on $m = 8$ terms. (a) The swap changes only the two teal paths from the leaves to their lowest common ancestor, here the root. The orange subtrees lie beside the path of x_1, and the blue subtrees beside the path of x_6. (b) Exposing the two leaves turns the children of the root into $x_1 + A$ and $x_6 + B$, where $A = x_0 + \Sigma[2, 3]$ adds the orange subtrees and $B = \Sigma[4, 5] + x_7$ adds the blue subtrees. (c) A local identity at the root exchanges x_1 and x_6 and leaves A and B in place. Applying the exposing steps in reverse order then gives the balanced tree with x_1 and x_6 exchanged.

a local identity at the lowest common ancestor exchanges the two leaves. Third, it applies the exposing steps in reverse order. For example, the swap of x_1 and x_6, whose lowest common ancestor is the root, is the chain

$$
\begin{aligned}
\Sigma[0,7] &= \big((x_0 + x_1) + \Sigma[2,3]\big) + \big(\Sigma[4,5] + (x_6 + x_7)\big) && \text{balanced tree} \\
&= \big((x_1 + x_0) + \Sigma[2,3]\big) + \big(\Sigma[4,5] + (x_6 + x_7)\big) && \text{commutativity} \\
&= \big(x_1 + \underbrace{(x_0 + \Sigma[2,3])}_{=:A}\big) + \big(\Sigma[4,5] + (x_6 + x_7)\big) && \text{associativity} \\
&= (x_1 + A) + \big(x_6 + \underbrace{(\Sigma[4,5] + x_7)}_{=:B}\big) && P + (Q + R) = Q + (P + R) \\
&= (x_6 + A) + (x_1 + B) && \text{swap at the root} \\
&= \big((x_0 + x_6) + \Sigma[2,3]\big) + \big(\Sigma[4,5] + (x_1 + x_7)\big) && \text{above exposing steps in reverse.}
\end{aligned}
$$

The first transition writes out the balanced tree along the two paths (Figure 4(a)). The second and third transitions expose x_1, at the nodes $\Sigma[0, 1]$ and $\Sigma[0, 3]$, and thereby form the sum A of the orange subtrees beside its path. The fourth exposes x_6 at the node $\Sigma[4, 7]$ and forms the sum B of the blue subtrees beside its path; at the parent $\Sigma[6, 7]$ of x_6, no step is needed, since x_6 already comes first in $x_6 + x_7$. The root now adds $x_1 + A$ and $x_6 + B$ (Figure 4(b)). The fifth transition is the swap at the root, $(x_1 + A) + (x_6 + B) = (x_6 + A) + (x_1 + B)$, a basic law on four words: it exchanges x_1 and x_6 and leaves A and B in place (Figure 4(c)). The last transition reverses the exposing transitions with x_1 and x_6 exchanged, and its right-hand side is the output of the balanced tree on the terms $x_0, x_6, x_2, x_3, x_4, x_5, x_1, x_7$. Each identity in the chain changes the sum at one node of the two paths, and its equality is carried to the root through the $O(\log m)$ additions above that node. A swap applies at most two identities at each node of the two paths and one at the lowest common ancestor, so it costs $\log^{O(1)} m \cdot w^{O(1)}$, and the $m - 1$ swaps together cost $m \log^{O(1)}(m + 1) \cdot w^{O(1)}$, which proves (9).

Later, in the formal technical sections, Lemma D.6 restates this rule and proves it, for real and complex short words whose scales and component widths may differ. Lemma D.8 generalizes it to sum circuits built on arbitrary, possibly deep, binary trees. The sums of this roadmap are all balanced; the product form of that generalization, Lemma D.9, is used for the short left-to-right products of Appendix E.

4.3 Carry Resolution

Notational convention. Throughout the paper, a boldface letter denotes a vector and the plain letter with a subscript denotes its entries, as with $\boldsymbol{a}$ and a_j below. In particular, for an unsigned integer x carried by N bits, write $\mathsf{bits}(x) \in \{0,1\}^N$ for its vector of bits, whose entry at position j is the bit x_j, so that $x = \sum_{j<N} x_j 2^j$ and every bit of x is one coefficient. We write

$$\boldsymbol{x} := \mathsf{bits}(x), \quad \boldsymbol{y} := \mathsf{bits}(y), \quad \boldsymbol{z} := \mathsf{bits}(z).$$

Henceforth in the roadmap, given two bit vectors $\boldsymbol{v}, \boldsymbol{v}' \in \{0,1\}^N$, we use $\mathsf{ADD}(\boldsymbol{v}, \boldsymbol{v}')$ to mean treating each input as the integer $\sum_j v_j 2^j$ and applying the schoolbook adder to the two integers, which propagates carries across all positions. Its output has $N+1$ bits, the last of which is its final carry. When we equate it with an N-bit string, such as an output of the circuit Carry below, we pad that string with a zero top bit. Given two vectors $\boldsymbol{v}$ and $\boldsymbol{v}'$ of the same length, we use $\boldsymbol{v} + \boldsymbol{v}'$ to mean coordinatewise addition, with no carries between positions. For scalars, such as two integers represented by w-bit words, the two notations coincide, and we use ADD and $+$ interchangeably. For example, the coordinatewise sum $\boldsymbol{y} + \boldsymbol{z}$ has entries in $\{0,1,2\}$, and if $y + z < 2^N$, then $\mathsf{Carry}(\boldsymbol{y} + \boldsymbol{z})$ is the binary representation of $y + z$.

Carry resolution. The multiplier computes a vector of nonnegative integer coefficients and then converts it into binary. For a vector $\boldsymbol{a} = (a_0, \ldots, a_{N-1})$ of nonnegative integers, each represented by an unsigned w-bit word, the carry-resolution circuit $\mathsf{Carry}(\boldsymbol{a})$ processes the positions from low to high. Starting from the carry $\kappa_0 := 0$, at position j it forms the column sum $a_j + \kappa_j$, outputs its parity b_j, and passes on the carry $\kappa_{j+1} := \lfloor (a_j + \kappa_j)/2 \rfloor$. Thus, as for the adder (see Equation (8)), every position satisfies the following *column equation*, which has an EF derivation of size $w^{O(1)}$:

$$a_j + \kappa_j = b_j + 2\kappa_{j+1}, \qquad b_j \in \{0,1\}. \tag{10}$$

In the roadmap, we apply Carry only to such nonnegative vectors, so all carries are nonnegative as well. The circuit outputs the N-bit string $\boldsymbol{b} := \mathsf{Carry}(\boldsymbol{a}) = (b_0, \ldots, b_{N-1})$. Unlike the final carry γ_N of the adder, which is a bit, the final carry κ_N is a word, and it is not part of the output. If κ_N vanishes, then $\mathsf{Carry}(\boldsymbol{a})$ is the binary representation of $\sum_{j<N} a_j 2^j$.

The rest of the roadmap relies on the following four inference rules, which relate the two kinds of addition (Lemma I.1). For vectors $\boldsymbol{a}, \boldsymbol{a}', \boldsymbol{V}$ of length N,

$$\Gamma : \mathcal{R} \;\rhd^{N\, w^{O(1)}}_{\mathsf{EF}}\; \mathsf{ADD}(y,z) = \mathsf{Carry}(\boldsymbol{y} + \boldsymbol{z}), \tag{11}$$

$$\Gamma : \mathcal{R} \;\rhd^{N\, w^{O(1)}}_{\mathsf{EF}}\; \mathsf{Carry}(\boldsymbol{a} + \boldsymbol{a}') = \mathsf{ADD}\big(\mathsf{Carry}(\boldsymbol{a}), \mathsf{Carry}(\boldsymbol{a}')\big), \tag{12}$$

$$\Gamma \;\rhd^{N\, w^{O(1)}}_{\mathsf{EF}}\; \boldsymbol{a} - \mathsf{Carry}(\boldsymbol{a}) = 2\boldsymbol{u} - \mathsf{Shift}\,\boldsymbol{u}, \tag{13}$$

$$\Gamma : V_{N-1} = 0 \;\rhd^{N\, w^{O(1)}}_{\mathsf{EF}}\; \mathsf{Carry}(\boldsymbol{a} + 2\boldsymbol{V} - \mathsf{Shift}\,\boldsymbol{V}) = \mathsf{Carry}(\boldsymbol{a}). \tag{14}$$

- In (11) and (12), the premise $\mathcal{R}$ states that the final carry of the carry-resolution circuit vanishes: for $\mathsf{Carry}(\boldsymbol{y} + \boldsymbol{z})$ in (11), this means $y + z < 2^N$, and for $\mathsf{Carry}(\boldsymbol{a} + \boldsymbol{a}')$ in (12), it means $\sum_{j<N}(a_j + a'_j)2^j < 2^N$. The premise ensures that the top output bit of the adder is zero, matching the zero padding of the other side.

- In (13), the vector $\boldsymbol{u}$, with $u_j := \kappa_{j+1}$, collects the carries out of the positions, and Shift shifts a vector up by one position: $(\mathsf{Shift}\,\boldsymbol{u})_j := u_{j-1}$ for $j \geq 1$, and $(\mathsf{Shift}\,\boldsymbol{u})_0 := 0$. Thus (13) says that carry resolution changes $\boldsymbol{a}$ by a vector of the form $2\boldsymbol{u} - \mathsf{Shift}\,\boldsymbol{u}$, called a *carry certificate*, and (14) says that adding such a vector does not affect carry resolution. The first two rules convert between binary integers and coefficient vectors, and the last two let us defer carry resolution (Section 6.2).

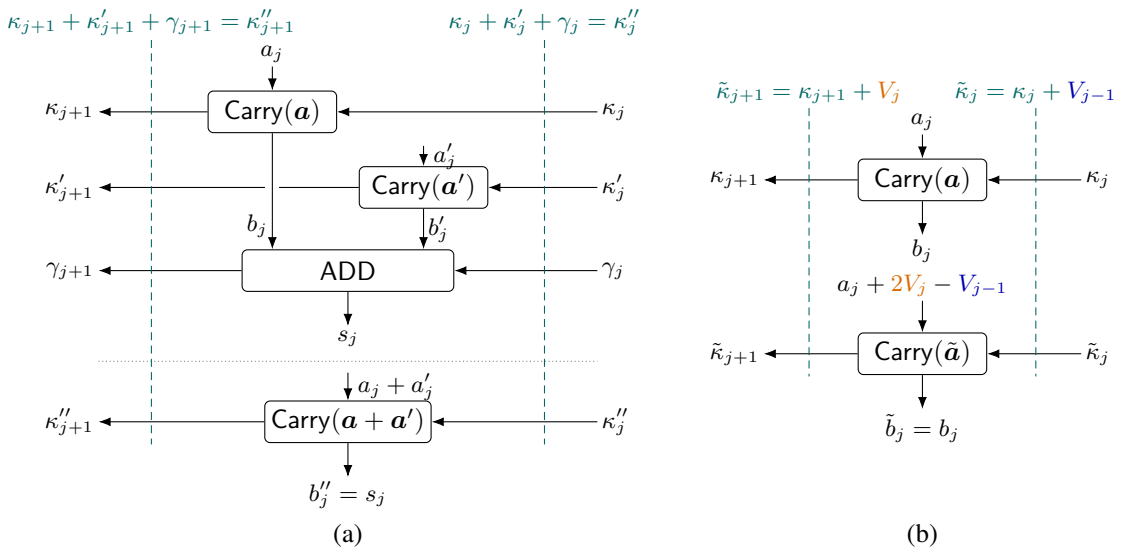


Figure 5: Position j of the carry chains in the derivations of (12) and (14). Each box is position j of the circuit named in it. As in schoolbook arithmetic, positions increase from right to left, so carries enter from position $j-1$ on the right and leave for position $j+1$ on the left. Each dashed cut is labeled with the invariant, which relates the carries crossing it. (a) Above the dotted line, $\mathsf{Carry}(\boldsymbol{a})$ and $\mathsf{Carry}(\boldsymbol{a}')$ output the bits b_j and b'_j, and the adder adds them with its carry γ_j. Below it, $\mathsf{Carry}(\boldsymbol{a}+\boldsymbol{a}')$ resolves the sum $a_j + a'_j$ directly, and the inductive step shows that its output bit b''_j equals the sum bit s_j of the adder. (b) Here $\tilde{\boldsymbol{a}} := \boldsymbol{a} + 2\boldsymbol{V} - \mathsf{Shift}\,\boldsymbol{V}$. The extra carry V_{j-1} entering position j cancels the term $-V_{j-1}$ of $\tilde{a}_j$ (blue), and the term $2V_j$ leaves as the extra carry V_j (orange), so both chains output the same bit b_j.

We now explain how to derive these conclusions related to carry resolution. All conclusions, with the exception of Equation (13), are proven via an induction over the positions. At each position, the inductive step is a chain of equalities, each relating a constant number of w-bit words, so it has size $w^{O(1)}$, and the N positions cost $N\,w^{O(1)}$ in total. Equation (13) needs no induction: each of its coordinates is a rearrangement of the column equation (10), at cost $w^{O(1)}$, so it also costs $N\,w^{O(1)}$.

EF derivation of (12). Four carry chains are involved (Figure 5(a)). Write κ_j, κ'_j, and κ''_j for the carries of $\mathsf{Carry}(\boldsymbol{a})$, $\mathsf{Carry}(\boldsymbol{a}')$, and $\mathsf{Carry}(\boldsymbol{a}+\boldsymbol{a}')$, and b_j, b'_j, and b''_j for their output bits. The adder adds b_j and b'_j with its own carries γ_j and outputs the sum bits s_j, so its column equation (8) reads $b_j + b'_j + \gamma_j = s_j + 2\gamma_{j+1}$. For $j = 0, 1, \ldots, N$, the invariant at position j is the equation

$$\kappa''_j = \kappa_j + \kappa'_j + \gamma_j, \tag{I_j}$$

that is, the carry entering position j of $\mathsf{Carry}(\boldsymbol{a}+\boldsymbol{a}')$ is the total of the carries entering position j of the other three chains. We prove $(\mathrm{I}_0), \ldots, (\mathrm{I}_N)$ by induction on j and, along the way, that $b''_j = s_j$ for every $j < N$. As in the commutativity example, the EF derivation unrolls this induction: it derives (I_0) and then performs the inductive step for $j = 0, 1, \ldots, N-1$ in turn, each time on the wires of position j and citing the invariant (I_j) derived just before.

Base case. At position 0, all carries are zero, so (I_0) holds.

Inductive step. Assume (I_j) for some $j < N$. We derive $b''_j = s_j$ and (I_{j+1}). The column sum of

$\mathsf{Carry}(\boldsymbol{a}+\boldsymbol{a}')$ at position j is

$$\begin{aligned}
(a_j + a'_j) + \kappa''_j &= (a_j + a'_j) + (\kappa_j + \kappa'_j + \gamma_j) && \text{induction hypothesis } (\mathrm{I}_j)\\
&= (a_j + \kappa_j) + (a'_j + \kappa'_j) + \gamma_j && \text{regrouping}\\
&= (b_j + 2\kappa_{j+1}) + (b'_j + 2\kappa'_{j+1}) + \gamma_j && \text{column equations (10) of } \mathsf{Carry}(\boldsymbol{a}) \text{ and } \mathsf{Carry}(\boldsymbol{a}')\\
&= (b_j + b'_j + \gamma_j) + 2(\kappa_{j+1} + \kappa'_{j+1}) && \text{regrouping, distributivity}\\
&= (s_j + 2\gamma_{j+1}) + 2(\kappa_{j+1} + \kappa'_{j+1}) && \text{column equation (8) of the adder}\\
&= s_j + 2(\kappa_{j+1} + \kappa'_{j+1} + \gamma_{j+1}) && \text{regrouping, distributivity.}
\end{aligned}$$

Only the first transition uses the induction hypothesis. The others cite the column equations of the other three chains at position j or rearrange a constant number of words by associativity, commutativity, and distributivity. Since s_j is a bit, $\mathsf{Carry}(\boldsymbol{a}+\boldsymbol{a}')$ outputs $b''_j = s_j$ and passes on the carry $\kappa''_{j+1} = \kappa_{j+1} + \kappa'_{j+1} + \gamma_{j+1}$, which is (I_{j+1}).

Last position. At position N, (I_N) gives $\kappa''_N = \kappa_N + \kappa'_N + \gamma_N$. Since all carries are nonnegative, the premise $\mathcal{R}$, which states $\kappa''_N = 0$, forces $\gamma_N = 0$. Thus the top output bit $s_N = \gamma_N$ of the adder is zero, matching the zero padding of $\mathsf{Carry}(\boldsymbol{a}+\boldsymbol{a}')$. Together with $b''_j = s_j$ for every $j < N$, this shows that the two sides of (12) agree in all $N+1$ bits.

EF derivation of (14). Write $\tilde{\boldsymbol{a}} := \boldsymbol{a} + 2\boldsymbol{V} - \mathsf{Shift}\,\boldsymbol{V}$, whose entries are $\tilde{a}_j = a_j + 2V_j - V_{j-1}$, with $V_{-1} := 0$. As before, κ_j and b_j are the carries and output bits of $\mathsf{Carry}(\boldsymbol{a})$, and we write $\tilde{\kappa}_j$ and $\tilde{b}_j$ for those of $\mathsf{Carry}(\tilde{\boldsymbol{a}})$ (Figure 5(b)). The invariant is

$$\tilde{\kappa}_j = \kappa_j + V_{j-1},$$

that is, the chain of $\tilde{\boldsymbol{a}}$ carries V_{j-1} more units into position j than the chain of $\boldsymbol{a}$. It holds at position 0, where both carries are zero and $V_{-1} = 0$. Given it at position j, the column sum of $\mathsf{Carry}(\tilde{\boldsymbol{a}})$ at position j is

$$\begin{aligned}
\tilde{a}_j + \tilde{\kappa}_j &= (a_j + 2V_j - V_{j-1}) + (\kappa_j + V_{j-1}) && \text{definition of } \tilde{\boldsymbol{a}}\text{, invariant at position } j\\
&= (a_j + \kappa_j) + 2V_j && \text{regrouping, cancellation}\\
&= (b_j + 2\kappa_{j+1}) + 2V_j && \text{column equation (10) of } \mathsf{Carry}(\boldsymbol{a})\\
&= b_j + 2(\kappa_{j+1} + V_j) && \text{regrouping, distributivity.}
\end{aligned}$$

In the second transition, the extra carry V_{j-1} cancels the term $-V_{j-1}$ of $\tilde{a}_j$ (blue in the figure), and in the last, the term $2V_j$ turns into V_j additional units of carry (orange). Since b_j is a bit, $\mathsf{Carry}(\tilde{\boldsymbol{a}})$ outputs the same bit $\tilde{b}_j = b_j$ as $\mathsf{Carry}(\boldsymbol{a})$ and passes on the carry $\tilde{\kappa}_{j+1} = \kappa_{j+1} + V_j$, which is the invariant at position $j+1$. At the last position, the invariant and the premise $V_{N-1} = 0$ give $\tilde{\kappa}_N = \kappa_N$, so the final carries agree as well.

EF derivations of other conclusions. The derivations of (11) and (13) are easier. Rule (11) compares two chains with the same carries, like the commutativity example, so the final carry of the adder equals that of $\mathsf{Carry}(\boldsymbol{y}+\boldsymbol{z})$, which vanishes by the premise. Rule (13) needs no induction, since each of its coordinates is a rearrangement of the column equation (10).

4.4 Other Reusable Inference Rules

The rest of the roadmap also uses a few reusable inference rules, which we collect in this subsection. We state each of them as a lemma, only in the form that the roadmap needs, and we do not prove them here. After each lemma, we point to the result of the formal technical sections that restates it, or a generalization of it,

and proves it. Several of the lemmas generalize rules of Section 3.1, which states them only in the special forms that the proof of index-generalization needs.

The equality check of the commutativity example combines the equivalences of the output bits by a balanced tree of AND gates. In general, a list of assertions and a check of their conjunction are different objects, and the following rules convert between them.

Lemma 4.1 (Folding and unfolding)**.** *For checks $P_0, \ldots, P_{m-1}$, let $\bigwedge_{j<m} P_j$ be the output of a balanced tree of AND gates with these leaves. Then*

$$\Gamma : \ [P_j]_{j<m} \ \rhd_{\mathsf{EF}}^{O(m)} \ \bigwedge_{j<m} P_j \qquad \textit{and} \qquad \Gamma : \ \bigwedge_{j<m} P_j \ \rhd_{\mathsf{EF}}^{O(m)} \ [P_j]_{j<m}.$$

For instance, the equality check of two arrays is the conjunction of the equalities between their entries. Folding these equalities gives the equality of the arrays, and unfolding recovers them. Lemmas B.2 and B.3 restate these rules and prove them, together with their conditional forms and their forms for OR trees. In particular, Lemma B.3 generalizes Lemma 3.1, the form of unfolding that the proof of index-generalization uses.

Many of the rules below are applied inside a derivation that is itself carried out under an assumption, such as a comparison between two indices. The next rule says that this costs nothing beyond a constant factor.

Lemma 4.2 (Conditional forms)**.** *Let H be a named Boolean wire in Γ. For checks $P_0, \ldots, P_{k-1}$ and $Q_0, \ldots, Q_{l-1}$,*

$$\Gamma : \ [P_j]_{j<k} \ \rhd_{\mathsf{EF}}^{S} \ [Q_i]_{i<l} \qquad \textit{implies} \qquad \Gamma : \ [H \to P_j]_{j<k} \ \rhd_{\mathsf{EF}}^{O(S)} \ [H \to Q_i]_{i<l}.$$

The above lemma says that if we have an EF derivation of size S in the unconditional form, then we have an EF derivation in the conditional form of size $O(S)$. We simply state the lemma without proof in the informal roadmap, and defer its proof to Lemma B.1.

Equal inputs give equal outputs. The following lemma generalizes Lemma 3.2, in which the circuit has a single output and the equality of the inputs is given bit by bit under an antecedent; its own conditional form, where one is needed, is supplied by Lemma 4.2.

Lemma 4.3 (Equality propagation and substitution)**.** *Let x and y be strings of w bits, and let C be a circuit with r output bits whose copies $C(x, z)$ and $C(y, z)$ have the same gates and share all other inputs z. Then*

$$\Gamma : \ x = y \ \rhd_{\mathsf{EF}}^{O(w+|C|+r)} \ C(x, z) = C(y, z).$$

In particular, for a check P, that is, a circuit with one output,

$$\Gamma : \ x = y, \ P(x, z) \ \rhd_{\mathsf{EF}}^{O(w+|P|)} \ P(y, z).$$

The sizes $|C|$ and $|P|$ count only the gates through which the equality is propagated, and exclude the circuits that compute x, y, and z. The strings may be arrays of words. If their equality is given entry by entry, then folding the entry equalities (Lemma 4.1) first gives $x = y$. For instance, equalities between the corresponding products of two pairings propagate through their sum trees in this way. Lemma B.4 restates this lemma and proves it, together with a form in which the equality of the inputs is given bit by bit.

A derivation may also distinguish cases according to the value of an address a given by s wires. We write $[a = j]$ for the equality check between a and the hardwired encoding of a constant j.

Lemma 4.4 (Proof by enumeration)**.** *For an s-bit address a, with $m := 2^s$, and a check c,*

$$\Gamma : \ \big[[a = j] \to c\big]_{j<m} \ \rhd_{\mathsf{EF}}^{O(m \log(m+1))} \ c.$$

The bound covers only the combination of the cases, and the derivations of the m cases are separate. Lemma B.5 restates this lemma and proves it.

Recall the array selector of Section 3.1: for an array $\boldsymbol{A}$ of $m = 2^s$ words and an s-bit address i, the selector $\mathsf{Sel}(\boldsymbol{A}, i)$ is a balanced binary tree of word multiplexers that outputs the entry A_i. The next two lemmas relate the selected entries to the entries at constant positions.

Lemma 4.5 (Selection at a fixed address)**.** *For an array $\boldsymbol{A}$ of $m = 2^s$ words of width w, an s-bit address i, and a constant $j < m$,*

$$\Gamma \ \rhd_{\mathsf{EF}}^{O(w \log(m+1))} \ [i = j] \to \mathsf{Sel}(\boldsymbol{A}, i) = A_j.$$

Lemma B.6 restates this lemma and proves it.

Lemma 4.6 (Selection commutes with entrywise circuits)**.** *Let $\boldsymbol{A}^{(1)}, \ldots, \boldsymbol{A}^{(c)}$ be a constant number c of arrays, each of $m = 2^s$ words of width at most w, and let K be an s-bit address. Let F be a circuit whose inputs are one word from each array and, optionally, an s-bit index. For $j < m$, let F_j be the copy of F on the entries at position j, and let F^* be the copy on the entries selected at K:*

$$F_j := F\big(A_j^{(1)}, \ldots, A_j^{(c)}, \, j\big), \qquad F^* := F\big(\mathsf{Sel}(\boldsymbol{A}^{(1)}, K), \ldots, \mathsf{Sel}(\boldsymbol{A}^{(c)}, K), \, K\big).$$

Here the index input of F_j is the hardwired encoding of j, and if F does not read an index, the last argument is omitted. Then

$$\Gamma \ \rhd_{\mathsf{EF}}^{O(m(|F|+w+s))} \ F^* = \mathsf{Sel}\big((F_j)_{j<m}, K\big).$$

For instance, selection commutes with coordinatewise sums, $\mathsf{Sel}(\boldsymbol{g} + \boldsymbol{g}', K) = \mathsf{Sel}(\boldsymbol{g}, K) + \mathsf{Sel}(\boldsymbol{g}', K)$, and with circuits that read the index, such as a circuit that computes the twiddle factor at each position. The lemma generalizes Lemma 3.3, in which F is a check on the entries of a single array. Lemma B.7 restates it, also allowing F to read further wires that are the same in all its copies, and proves it.

Finally, Lemma 3.4 is the basic form of index-generalization, in which the check reads a single entry of one array. Most applications in Section 7 need the following general form, in which the check reads the entries of several arrays selected at the same index, and possibly the index itself.

Lemma 4.7 (Index-generalization, general form)**.** *Let $\boldsymbol{A}^{(1)}, \ldots, \boldsymbol{A}^{(c)}$ be a constant number c of arrays in Γ, each of $m = 2^s$ words of width at most w. Let P be a check whose inputs are one word from each array and, optionally, an s-bit index. For $j < m$, let P_j be the copy of P on the entries at position j, and for an s-bit index i, let φ_i be the copy on the entries selected at i:*

$$P_j := P\big(A_j^{(1)}, \ldots, A_j^{(c)}, \, j\big), \qquad \varphi_i := P\big(\mathsf{Sel}(\boldsymbol{A}^{(1)}, i), \ldots, \mathsf{Sel}(\boldsymbol{A}^{(c)}, i), \, i\big),$$

with the same convention for the last argument as in Lemma 4.6. If

$$\Gamma' \ \rhd_{\mathsf{EF}}^{S} \ \varphi_i$$

for every context $\Gamma' \supseteq \Gamma$ and every ordered tuple of s wires of Γ', read as an s-bit index and denoted i, then

$$\Gamma \ \rhd_{\mathsf{EF}}^{S+O(m(|P|+w+s))} \ [P_j]_{j<m}.$$

In our applications, $m \le N$, and $|P|$ and w are polylogarithmic in N, so the conclusion costs $S + N \log^{O(1)} N$. We establish the premise by a derivation at an arbitrary index i, which assumes nothing about i and applies only rules that hold in every context (Section 3.2). Lemma B.8 restates this lemma, also allowing P to read further wires of Γ that are the same in all its copies. Its proof is that of Lemma 3.4 in Section 3.3, with Lemma 4.6 in place of its special case Lemma 3.3.

5 FFT Warmup

5.1 Simplifying (False) Assumptions

The FFT networks below use powers of roots of unity as twiddle factors. For powers of two m, we write

$$\omega_m := e^{2\pi \mathrm{i}/m} \qquad \text{and} \qquad \omega := \omega_N = e^{2\pi \mathrm{i}/N}. \tag{15}$$

To illustrate the key ideas, this roadmap ignores the management of numerical precision. Specifically, we pretend the following.

1. In the FFT-based multiplier, every real or complex number that arises, including every root of unity ω_m^j used as a twiddle factor, is represented exactly by a *short word* of $\log^{O(1)} N$ bits. Hence the networks FFT and IFFT compute the discrete Fourier transform and its inverse exactly.

2. Addition and multiplication of complex or real numbers (represented as $\log^{O(1)} N$-bit short words) are computed by circuits of size $\log^{O(1)} N$, and the results are exact with no precision loss.

3. The basic laws of arithmetic on $\log^{O(1)} N$-bit short words have EF derivations of size $\log^{O(1)} N$. For short words x, y, and z, these laws include the following.

   - *Ring laws:* commutativity and associativity of addition and multiplication, distributivity, the laws for adding zero and for multiplying by zero and one, and cancellation, $(x + y) - y = x$.
   - *Order laws,* for real short words: transitivity of $\leq$, and monotonicity: if $x \leq y$, then $x+z \leq y+z$ and $\lfloor x/2 \rfloor \leq \lfloor y/2 \rfloor$, and if also $z \geq 0$, then $xz \leq yz$.
   - *Index arithmetic:* the ring laws for L-bit indices with arithmetic modulo N, and $(x-y) \bmod N = x - y$ for indices $y \leq x$.
   - *Identities of roots of unity,* such as $\omega_m^a \omega_m^b = \omega_m^{a+b}$ and $\omega_m^{2a} = \omega_{m/2}^a$, where the exponents a, b are given by input wires.

4. The summation regrouping law (9) shown earlier transfers to real or complex numbers represented as $\log^{O(1)} N$-bit short words.

5. The FFT convolution (described below), consisting of a forward Fourier transform on each of the two operands, a pointwise product, and an inverse transform, outputs a vector of integers, although its intermediate wires carry complex numbers. The multiplier obtains its final output by performing carry resolution directly on this vector.

The formal technical sections below will handle precision management explicitly, allowing us to remove these assumptions.

5.2 The FFT Networks

Our last example concerns the networks that compute Fourier transforms. Let $N = 2^L$ be a power of two. For a power of two m, the radix-two network FFT_m on vectors of length m is defined recursively. For $m = 1$, it returns its input. For $m \geq 2$, it splits its input $\boldsymbol{v}$ into the even part $\boldsymbol{v}^{\mathrm{e}} := (v_0, v_2, \ldots, v_{m-2})$ and the odd part $\boldsymbol{v}^{\mathrm{o}} := (v_1, v_3, \ldots, v_{m-1})$, which is wiring, computes $\boldsymbol{E} := \mathsf{FFT}_{m/2}(\boldsymbol{v}^{\mathrm{e}})$ and $\boldsymbol{O} := \mathsf{FFT}_{m/2}(\boldsymbol{v}^{\mathrm{o}})$ by two recursive calls, and combines them by $m/2$ *butterflies*: for $r < m/2$,

$$\mathsf{FFT}_m(\boldsymbol{v})_r = E_r + \omega_m^r O_r, \qquad \mathsf{FFT}_m(\boldsymbol{v})_{r+m/2} = E_r - \omega_m^r O_r.$$

We write $\mathsf{FFT} := \mathsf{FFT}_N$. This network has $\frac{N}{2} \log N$ butterflies, each of which performs a constant number of operations on short words, and it computes the discrete Fourier transform, $\mathsf{FFT}(\boldsymbol{v})_r = \sum_{j<N} \omega^{rj} v_j$. The network $\overline{\mathsf{FFT}}$ is the same network with conjugated twiddle factors ω_m^{-r}, and it computes $\overline{\mathsf{FFT}}(\boldsymbol{v})_r = \sum_{j<N} \omega^{-rj} v_j$. The inverse network IFFT computes $\mathsf{IFFT}(\boldsymbol{X})_k = \frac{1}{N} \sum_{r<N} \omega^{-rk} X_r$. It runs the recursion of $\overline{\mathsf{FFT}}$ backwards: a call of size $m \geq 2$ first applies $m/2$ butterflies, which map X_r and $X_{r+m/2}$ to $X_r + X_{r+m/2}$ and $\omega_m^{-r}(X_r - X_{r+m/2})$, then makes one recursive call on each of the two resulting vectors of length $m/2$, and interleaves their outputs. The factor $\frac{1}{N}$ is applied once, at the end (Appendix A.5.3). Finally, the pairing of two vectors of the same length m is $\langle \boldsymbol{u}, \boldsymbol{v} \rangle := \sum_{j<m} u_j v_j$, summed along the balanced tree, without complex conjugation, and $\boldsymbol{u} \odot \boldsymbol{v}$ is their coordinatewise product. For $k < N$, we write $\boldsymbol{e}_k$ for the indicator vector of position k, and $\boldsymbol{\chi}_k$ for the vector with entries $(\boldsymbol{\chi}_k)_r = \omega^{-rk}$, that is, the values of a character of the cyclic group $\mathbb{Z}_N$.

5.3 A Selection of FFT Identities

The rest of the roadmap relies on the following identities of the networks. For vectors $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{X}$ of length N and a constant $k < N$,

$$\textit{Parseval identity}: \quad \Gamma \rhd^{N \log^{O(1)} N}_{\mathsf{EF}} \quad \big\langle \mathsf{FFT}(\boldsymbol{u}), \overline{\mathsf{FFT}}(\boldsymbol{v}) \big\rangle = N \langle \boldsymbol{u}, \boldsymbol{v} \rangle, \tag{16}$$

$$\textit{transposition identity}: \quad \Gamma \rhd^{N \log^{O(1)} N}_{\mathsf{EF}} \quad \big\langle \boldsymbol{u}, \mathsf{IFFT}(\boldsymbol{X}) \big\rangle = \tfrac{1}{N} \big\langle \overline{\mathsf{FFT}}(\boldsymbol{u}), \boldsymbol{X} \big\rangle, \tag{17}$$

$$\textit{cancellation identity}: \quad \Gamma \rhd^{N \log^{O(1)} N}_{\mathsf{EF}} \quad \mathsf{FFT}\big(\mathsf{IFFT}(\boldsymbol{X})\big) = \boldsymbol{X}, \tag{18}$$

$$\textit{impulse identity}: \quad \Gamma \rhd^{N \log^{O(1)} N}_{\mathsf{EF}} \quad \overline{\mathsf{FFT}}(\boldsymbol{e}_k) = \boldsymbol{\chi}_k, \tag{19}$$

$$\textit{shift identity}: \quad \Gamma \rhd^{N \log^{O(1)} N}_{\mathsf{EF}} \quad \overline{\mathsf{FFT}}\big((u_{k-j})_{j<N}\big) = \boldsymbol{\chi}_k \odot \mathsf{FFT}(\boldsymbol{u}). \tag{20}$$

In the above, since k is a constant, $\boldsymbol{e}_k$ and $\boldsymbol{\chi}_k$ are constant vectors, and $(u_{k-j})_{j<N}$ is read from $\boldsymbol{u}$ by wiring. In Section 6.2, the convolution theorem at a fixed coordinate is derived mainly from (16), (17), (19), and (20), and in Section 7.4, associativity uses (18).

We now explain how to derive these identities. Their textbook proofs expand the transforms into their defining sums. For example, the textbook proof of (16) uses $\sum_{r<N} \omega^{r(j-l)} = N \cdot [j = l]$. In EF, it would write out a triple sum over r, j, l with N^3 terms, although the circuits involved have size $N \log^{O(1)} N$. Instead, our derivations follow the recursion of the networks, just as the FFT itself avoids evaluating the defining sums. We present the derivation of (16) in detail. The other derivations have the same structure, and we only sketch them.

EF derivation of (16). The networks FFT on $\boldsymbol{u}$ and $\overline{\mathsf{FFT}}$ on $\boldsymbol{v}$ have the same recursion tree. At each of its nodes, of some size m, the first network calls FFT_m on a subvector $\boldsymbol{u}'$ of $\boldsymbol{u}$, and the second calls $\overline{\mathsf{FFT}}_m$ on the subvector $\boldsymbol{v}'$ of $\boldsymbol{v}$ at the same positions. The claim at such a node is the equation

$$\big\langle \mathsf{FFT}_m(\boldsymbol{u}'), \overline{\mathsf{FFT}}_m(\boldsymbol{v}') \big\rangle = m \langle \boldsymbol{u}', \boldsymbol{v}' \rangle. \tag{P_m}$$

At the root, where $m = N$, $\boldsymbol{u}' = \boldsymbol{u}$, and $\boldsymbol{v}' = \boldsymbol{v}$, the claim is (16). We prove the claims at all nodes by induction on m. The EF derivation unrolls this induction over the recursion tree, from the leaves up: it derives the claims at the N leaves, which have size 1, and then, for $m = 2, 4, \ldots, N$ in turn, performs the inductive step at every node of size m, citing the claims $(\mathrm{P}_{m/2})$ already derived at its two children.

Base case. A call of size 1 returns its input, so at a leaf, (P_1) reads $u'_0 v'_0 = 1 \cdot u'_0 v'_0$, an instance of the identity law.

Inductive step. Consider a node of size $m \geq 2$, and assume $(\mathrm{P}_{m/2})$ at its two children. Write $\boldsymbol{F} := \mathsf{FFT}_m(\boldsymbol{u}')$ and $\boldsymbol{F}' := \overline{\mathsf{FFT}}_m(\boldsymbol{v}')$ for the outputs of the two calls at the node, and $\boldsymbol{E} := \mathsf{FFT}_{m/2}(\boldsymbol{u}'^{\mathrm{e}})$, $\boldsymbol{O} := \mathsf{FFT}_{m/2}(\boldsymbol{u}'^{\mathrm{o}})$, $\boldsymbol{E}' := \overline{\mathsf{FFT}}_{m/2}(\boldsymbol{v}'^{\mathrm{e}})$, and $\boldsymbol{O}' := \overline{\mathsf{FFT}}_{m/2}(\boldsymbol{v}'^{\mathrm{o}})$ for the outputs of the calls at its children. Thus the induction hypothesis states

$$\langle \boldsymbol{E}, \boldsymbol{E}' \rangle = \tfrac{m}{2} \langle \boldsymbol{u}'^{\mathrm{e}}, \boldsymbol{v}'^{\mathrm{e}} \rangle, \qquad \langle \boldsymbol{O}, \boldsymbol{O}' \rangle = \tfrac{m}{2} \langle \boldsymbol{u}'^{\mathrm{o}}, \boldsymbol{v}'^{\mathrm{o}} \rangle.$$

For $r < m/2$, the r-th butterflies of the two calls output $F_r, F_{r+m/2}$ and $F'_r, F'_{r+m/2}$, and their products satisfy the *butterfly identity*

$$\begin{aligned} F_r F'_r + F_{r+m/2} F'_{r+m/2} &= (E_r + \omega_m^r O_r)(E'_r + \omega_m^{-r} O'_r) + (E_r - \omega_m^r O_r)(E'_r - \omega_m^{-r} O'_r) \\ &= 2\big(E_r E'_r + O_r O'_r\big). \end{aligned} \tag{B_r}$$

The first equality substitutes the outputs of the butterflies. The second multiplies out, cancels the cross terms, and uses $\omega_m^r \omega_m^{-r} = 1$. Since the butterfly identity relates a constant number of short words, it has an EF derivation of size $\log^{O(1)} N$. The pairing at the node is then

$$\begin{aligned} \langle \boldsymbol{F}, \boldsymbol{F}' \rangle &= \sum_{r<m/2} \big(F_r F'_r + F_{r+m/2} F'_{r+m/2}\big) && \text{regrouping} \\ &= \sum_{r<m/2} 2\big(E_r E'_r + O_r O'_r\big) && \text{butterfly identities } (\mathrm{B}_r) \\ &= 2\langle \boldsymbol{E}, \boldsymbol{E}' \rangle + 2\langle \boldsymbol{O}, \boldsymbol{O}' \rangle && \text{regrouping, distributivity} \\ &= 2 \cdot \tfrac{m}{2} \langle \boldsymbol{u}'^{\mathrm{e}}, \boldsymbol{v}'^{\mathrm{e}} \rangle + 2 \cdot \tfrac{m}{2} \langle \boldsymbol{u}'^{\mathrm{o}}, \boldsymbol{v}'^{\mathrm{o}} \rangle && \text{induction hypothesis } (\mathrm{P}_{m/2}) \text{ at the two children} \\ &= m\big(\langle \boldsymbol{u}'^{\mathrm{e}}, \boldsymbol{v}'^{\mathrm{e}} \rangle + \langle \boldsymbol{u}'^{\mathrm{o}}, \boldsymbol{v}'^{\mathrm{o}} \rangle\big) && 2 \cdot \tfrac{m}{2} = m, \text{ distributivity} \\ &= m \langle \boldsymbol{u}', \boldsymbol{v}' \rangle && \text{regrouping,} \end{aligned}$$

which is (P_m). Only the fourth transition uses the induction hypothesis. The first groups the two products of each butterfly, which lie in different halves of the balanced tree of $\langle \boldsymbol{F}, \boldsymbol{F}' \rangle$, and the last merges the sums over the even and the odd positions of $\boldsymbol{u}'$ and $\boldsymbol{v}'$ into one sum in index order.

Cost. At a node of size m, the inductive step derives the $m/2$ butterfly identities, each of size $\log^{O(1)} N$, and applies them and distributivity inside sums with at most m terms, at cost $\log^{O(1)} N$ per addition. Its three regroupings, of sums with m terms, cost $m \log^{O(1)} N$ by (9). Thus the step costs $m \log^{O(1)} N$, and it cites the claims at the two children. The nodes of each size together contain N entries, so each of the $\log N + 1$ sizes costs $N \log^{O(1)} N$, and the whole derivation has size $N \log^{O(1)} N$.

EF derivations of other identities. The derivations of (17)–(20) follow the same pattern, and we omit them. Each unrolls an induction over the recursion tree. At a node of size m, the inductive step combines the claims at the two children by one local derivation per butterfly, which uses only the arithmetic of short words and the identities of roots of unity, and, for the pairings in (17), by a constant number of regroupings, at cost $m \log^{O(1)} N$. Each identity thus costs $N \log^{O(1)} N$ in total. In (17) and (18), the call of IFFT at each node is matched with the call of $\overline{\mathsf{FFT}}$ or of FFT at the same node. For the transposition identity, the inductive step moves the pairing through the butterflies of the node, and for the cancellation identity, it cancels the butterflies of the two calls against each other. In (19) and (20), the bits of the constant k are hardwired into the derivation: they determine which subvectors of $\boldsymbol{e}_k$ are nonzero, and how the subvectors of $(u_{k-j})_{j<N}$ at the nodes are matched with those of $\boldsymbol{u}$.

Later, in the formal technical sections, Lemma F.3 proves the transposition identity, and Lemma F.4 proves approximate versions of the other four identities, whose two sides differ by small errors.

6 Main Example: Distributivity of Schoolbook Adder and FFT Multiplier

6.1 FFT Multiplier and Ordinary Convolution

The FFT multiplier. We assume without loss of generality that the inputs x, y, z are n-bit integers, padded with zeros in their high-order bits to N-bit words, where $N = 2^L > 16n$ is a power of 2. The factor 16 leaves room for every product and sum below, including the $O(\log N)$ extra bits from carry resolution. Write

$$\boldsymbol{x} = (x_j)_{j<N} = \mathsf{bits}(x), \quad \boldsymbol{y} = (y_j)_{j<N} = \mathsf{bits}(y), \quad \boldsymbol{z} = (z_j)_{j<N} = \mathsf{bits}(z).$$

An FFT implementation of the multiplier computes

$$\mathsf{FFTMul}(x, y) = \mathsf{Carry}\big(\mathsf{FFTConv}(\boldsymbol{x}, \boldsymbol{y})\big), \text{ where } \mathsf{FFTConv}(\boldsymbol{x}, \boldsymbol{y}) = \mathsf{IFFT}\big(\mathsf{FFT}(\boldsymbol{x}) \odot \mathsf{FFT}(\boldsymbol{y})\big),$$

truncated to its declared output length. Here FFT and IFFT are the networks of the FFT warmup (Section 5), which compute the discrete Fourier transform and its inverse, $\odot$ denotes coordinatewise multiplication, and Carry is the carry-resolution circuit of the basic warmup (Section 4). Each network has $O(N \log N)$ butterflies, each of which performs a constant number of short-word operations, so the multiplier has size $N \log^{O(1)} N$.

Ordinary convolution at a constant index. Given a fixed constant $k < N$, we define the convolution outcome at index k as follows:

$$\mathsf{Conv}_k(\boldsymbol{x}, \boldsymbol{y}) := \sum_{j<N} x_j y_{k-j}, \tag{21}$$

where the index $k - j$ is taken modulo N. Because of the leading zeros, the cyclic indices never wrap around, and $\mathsf{Conv}_k(\boldsymbol{x}, \boldsymbol{y})$ is simply the k-th column sum of schoolbook multiplication. In the naïve derivation (Section 6.2), we will show as an important stepping stone that for every fixed $k < N$,

$$\mathsf{FFTConv}(\boldsymbol{x}, \boldsymbol{y})_k = \mathsf{Conv}_k(\boldsymbol{x}, \boldsymbol{y}). \tag{22}$$

In particular, the output of the FFT convolution is indeed a vector of integers, and resolving its carries yields the product of x and y.

The goal: distributivity. As mentioned, we will use distributivity as an example in our informal roadmap. Our running example is the distributive law of Theorem A.2,

$$\mathsf{FFTMul}\big(x, \mathsf{ADD}(y, z)\big) = \mathsf{ADD}\big(\mathsf{FFTMul}(x, y), \mathsf{FFTMul}(x, z)\big).$$

The bits of x, y, z are free variables, so a single derivation must establish this identity for all of their values. We aim for a derivation of size $N \log^{O(1)} N$, comparable to the size of the circuits on the two sides.

6.2 A Naïve Quadratic EF Derivation

If a derivation of size $N^2 \log^{O(1)} N$ is affordable, a simple way to derive distributivity is to compare the FFT convolution with the ordinary convolution, that is, with schoolbook multiplication, one fixed output coordinate at a time. We first derive that the FFT convolution is linear in its second argument, which includes distributivity as an identity between coefficient vectors, and then apply the carry rules.

Linearity of the FFT convolution. For **every fixed** $k < N$, we derive the following two facts.

1. *The FFT convolution agrees with ordinary convolution:* $\mathsf{FFTConv}(\boldsymbol{x}, \boldsymbol{h})_k = \mathsf{Conv}_k(\boldsymbol{x}, \boldsymbol{h})$ for each vector $\boldsymbol{h}$ that occurs below, such as $\boldsymbol{h} \in \{\boldsymbol{y}, \boldsymbol{z}, \boldsymbol{y} + \boldsymbol{z}\}$. For a single fixed k, selecting the k-th coordinate of $\mathsf{FFTConv}(\boldsymbol{x}, \boldsymbol{h})$ is done by wiring; moreover, $\mathsf{Conv}_k(\boldsymbol{x}, \boldsymbol{h})$ can be implemented in $O(N)$ size through appropriate wiring. It is not hard to see that for a single fixed k, there is an EF derivation of this identity with $N \log^{O(1)} N$ proof size: at a fixed coordinate, the convolution theorem follows from a short chain of standard identities of the Fourier networks. Recall the conjugated network $\overline{\mathsf{FFT}}$, the pairing $\langle \boldsymbol{u}, \boldsymbol{v} \rangle$, the indicator vector $\boldsymbol{e}_k$, and the character $\boldsymbol{\chi}_k$ from the FFT warmup. Then

$$\begin{aligned}
\mathsf{FFTConv}(\boldsymbol{x}, \boldsymbol{h})_k &= \big\langle \boldsymbol{e}_k, \mathsf{IFFT}(\mathsf{FFT}(\boldsymbol{x}) \odot \mathsf{FFT}(\boldsymbol{h})) \big\rangle && \text{(i) selection as pairing} \\
&= \tfrac{1}{N} \big\langle \overline{\mathsf{FFT}}(\boldsymbol{e}_k), \mathsf{FFT}(\boldsymbol{x}) \odot \mathsf{FFT}(\boldsymbol{h}) \big\rangle && \text{(ii) transposition (17)} \\
&= \tfrac{1}{N} \big\langle \boldsymbol{\chi}_k, \mathsf{FFT}(\boldsymbol{x}) \odot \mathsf{FFT}(\boldsymbol{h}) \big\rangle && \text{(iii) impulse identity (19)} \\
&= \tfrac{1}{N} \big\langle \mathsf{FFT}(\boldsymbol{x}), \boldsymbol{\chi}_k \odot \mathsf{FFT}(\boldsymbol{h}) \big\rangle && \text{(iv) regrouping} \\
&= \tfrac{1}{N} \big\langle \mathsf{FFT}(\boldsymbol{x}), \overline{\mathsf{FFT}}\big((h_{k-j})_{j<N}\big) \big\rangle && \text{(v) shift identity (20)} \\
&= \big\langle \boldsymbol{x}, (h_{k-j})_{j<N} \big\rangle = \mathsf{Conv}_k(\boldsymbol{x}, \boldsymbol{h}) && \text{(vi) Parseval-type identity (16).}
\end{aligned} \tag{23}$$

   As in the FFT warmup, $\boldsymbol{e}_k$ and $\boldsymbol{\chi}_k$ are constant vectors, and $(h_{k-j})_{j<N}$ is read from $\boldsymbol{h}$ by wiring. Transitions (ii), (iii), (v), and (vi) are identities of the FFT warmup. Transition (i) writes the k-th entry as a pairing with the constant vector $\boldsymbol{e}_k$: by the short-word laws for multiplying by zero and one and for adding zero, the pairing reduces to its k-th product. Transition (iv) regroups each product of the pairing by the short-word laws and propagates these equalities through the sum tree of the pairing (Lemma 4.3). Each transition costs $N \log^{O(1)} N$.

2. *The convolution is linear in its second argument:* for vectors $\boldsymbol{g}$ and $\boldsymbol{g}'$ that occur below, such as $\boldsymbol{g} = \boldsymbol{y}$ and $\boldsymbol{g}' = \boldsymbol{z}$,

$$\mathsf{Conv}_k(\boldsymbol{x}, \boldsymbol{g} \pm \boldsymbol{g}') = \mathsf{Conv}_k(\boldsymbol{x}, \boldsymbol{g}) \pm \mathsf{Conv}_k(\boldsymbol{x}, \boldsymbol{g}'), \qquad \mathsf{Conv}_k(\boldsymbol{x}, 2\boldsymbol{g}) = 2\mathsf{Conv}_k(\boldsymbol{x}, \boldsymbol{g}). \tag{24}$$

   For the sum of $\boldsymbol{y}$ and $\boldsymbol{z}$, we derive it by the chain

$$\begin{aligned}
\mathsf{Conv}_k(\boldsymbol{x}, \boldsymbol{y} + \boldsymbol{z}) &= \sum_{j<N} x_j (y_{k-j} + z_{k-j}) && \text{definition of } \mathsf{Conv}_k \\
&= \sum_{j<N} \big(x_j y_{k-j} + x_j z_{k-j} \big) && \text{short-word distributivity} \\
&= \sum_{j<N} x_j y_{k-j} + \sum_{j<N} x_j z_{k-j} && \text{regrouping the sum} \\
&= \mathsf{Conv}_k(\boldsymbol{x}, \boldsymbol{y}) + \mathsf{Conv}_k(\boldsymbol{x}, \boldsymbol{z}) && \text{definition of } \mathsf{Conv}_k.
\end{aligned} \tag{25}$$

   The second transition applies short-word distributivity to each of the N products and propagates these equalities through the sum tree (Lemma 4.3), and the third is the regrouping (9) from the warmup, applied to the $2N$ products. The same chain applies to any two vectors $\boldsymbol{g}$ and $\boldsymbol{g}'$. Differences follow by applying it to $\boldsymbol{g} - \boldsymbol{g}'$ and $\boldsymbol{g}'$, whose sum is $\boldsymbol{g}$, and doubled vectors by applying it to $\boldsymbol{g}$ and $\boldsymbol{g}$, since $2\boldsymbol{g} = \boldsymbol{g} + \boldsymbol{g}$. Each instance costs $N \log^{O(1)} N$.

Together, the two facts give the same identities for the FFT convolution at every coordinate k; for instance, $\mathsf{FFTConv}(\boldsymbol{x}, \boldsymbol{y} + \boldsymbol{z})_k = \mathsf{FFTConv}(\boldsymbol{x}, \boldsymbol{y})_k + \mathsf{FFTConv}(\boldsymbol{x}, \boldsymbol{z})_k$. Folding the N coordinates (Lemma 4.1) shows that the FFT convolution is linear in its second argument:

$$\begin{gathered}
\mathsf{FFTConv}(\boldsymbol{x}, \boldsymbol{g} \pm \boldsymbol{g}') = \mathsf{FFTConv}(\boldsymbol{x}, \boldsymbol{g}) \pm \mathsf{FFTConv}(\boldsymbol{x}, \boldsymbol{g}'), \\
\mathsf{FFTConv}(\boldsymbol{x}, 2\boldsymbol{g}) = 2\,\mathsf{FFTConv}(\boldsymbol{x}, \boldsymbol{g}).
\end{gathered} \tag{26}$$

For $\boldsymbol{g} = \boldsymbol{y}$ and $\boldsymbol{g}' = \boldsymbol{z}$, this is distributivity of the FFT convolution as an identity between vectors,

$$\mathsf{FFTConv}(\boldsymbol{x}, \boldsymbol{y} + \boldsymbol{z}) = \mathsf{FFTConv}(\boldsymbol{x}, \boldsymbol{y}) + \mathsf{FFTConv}(\boldsymbol{x}, \boldsymbol{z}). \tag{27}$$

Applying the carry rules. It remains to convert between binary strings and coefficient vectors. We do so by the chain

$$\begin{aligned}
&\mathsf{FFTMul}\big(x, \mathsf{ADD}(y, z)\big) \\
&\quad = \mathsf{Carry}\big(\mathsf{FFTConv}(\boldsymbol{x}, \mathsf{Carry}(\boldsymbol{y} + \boldsymbol{z}))\big) && \text{the adder resolves carries (11)} \\
&\quad = \mathsf{Carry}\big(\mathsf{FFTConv}(\boldsymbol{x}, \boldsymbol{y} + \boldsymbol{z})\big) && \text{deferring carry resolution} \\
&\quad = \mathsf{Carry}\big(\mathsf{FFTConv}(\boldsymbol{x}, \boldsymbol{y}) + \mathsf{FFTConv}(\boldsymbol{x}, \boldsymbol{z})\big) && \text{distributivity (27)} \\
&\quad = \mathsf{ADD}\big(\mathsf{FFTMul}(x, y), \mathsf{FFTMul}(x, z)\big) && \text{adding resolved vectors (12).}
\end{aligned}$$

Here $\boldsymbol{y} + \boldsymbol{z}$ is the coordinatewise sum of the bit vectors of y and z, a coefficient vector with entries in $\{0, 1, 2\}$ that represents $y + z$. By the carry rule (11), resolving its carries yields the binary representation of $y + z$, which is what the schoolbook adder computes, and propagating this equality through the multiplier (Lemma 4.3) gives the first equality. The third equality propagates (27) through the carry circuit in the same way, and the last one adds resolved vectors by the carry rule (12). Since the coefficients are short words, of $\log^{O(1)} N$ bits, each carry rule costs $N \log^{O(1)} N$.

Deferring carry resolution. The second equality of the chain needs one more idea. Its left side applies the FFT convolution to the resolved vector $\mathsf{Carry}(\boldsymbol{h})$, where $\boldsymbol{h} := \boldsymbol{y} + \boldsymbol{z}$, and its right side applies it to $\boldsymbol{h}$ itself. By the carry certificate (13), the two vectors are related by

$$\boldsymbol{h} = \mathsf{Carry}(\boldsymbol{h}) + 2\boldsymbol{u} - \mathsf{Shift}\,\boldsymbol{u},$$

where $\boldsymbol{u}$ collects the carries of $\mathsf{Carry}(\boldsymbol{h})$. The idea is that the FFT convolution turns the difference $2\boldsymbol{u} - \mathsf{Shift}\,\boldsymbol{u}$ into a difference of the same form, $2\boldsymbol{V} - \mathsf{Shift}\,\boldsymbol{V}$ for $\boldsymbol{V} := \mathsf{FFTConv}(\boldsymbol{x}, \boldsymbol{u})$, and that adding a difference of this form does not change the output of carry resolution.

Besides linearity (26), this needs that the FFT convolution commutes with shifts. Since the convolution is cyclic, the natural shift is the cyclic shift CShift_1 by one position, the fixed rearrangement of wires given by $(\mathsf{CShift}_1 \boldsymbol{v})_j := v_{j-1}$, with indices modulo N. Like linearity, the identity

$$\mathsf{FFTConv}(\boldsymbol{x}, \mathsf{CShift}_1 \boldsymbol{u}) = \mathsf{CShift}_1 \boldsymbol{V} \tag{28}$$

is derived one coordinate at a time. At a fixed coordinate k, the column sum $\mathsf{Conv}_k(\boldsymbol{x}, \mathsf{CShift}_1 \boldsymbol{u})$ reads the entries $(\mathsf{CShift}_1 \boldsymbol{u})_{k-j} = u_{k-1-j}$ by wiring, so it is a copy of $\mathsf{Conv}_{k-1}(\boldsymbol{x}, \boldsymbol{u})$ on the same wires. Item 1 at k and at $k - 1$ therefore gives $\mathsf{FFTConv}(\boldsymbol{x}, \mathsf{CShift}_1 \boldsymbol{u})_k = V_{k-1}$, which is the k-th entry of $\mathsf{CShift}_1 \boldsymbol{V}$.

The carry rules, however, use the shift Shift, which drops the last entry instead of moving it to position 0. The two shifts agree on every vector whose last entry is zero. This holds for $\boldsymbol{u}$, whose last entry u_{N-1} is the final carry of $\mathsf{Carry}(\boldsymbol{y} + \boldsymbol{z})$, and for $\boldsymbol{V}$; we show both under Premises below. Then

$$\begin{aligned}
&\mathsf{FFTConv}(\boldsymbol{x}, \boldsymbol{h}) \\
&\quad = \mathsf{FFTConv}\big(\boldsymbol{x}, \mathsf{Carry}(\boldsymbol{h}) + 2\boldsymbol{u} - \mathsf{CShift}_1 \boldsymbol{u}\big) && \text{carry certificate, } \mathsf{Shift}\,\boldsymbol{u} = \mathsf{CShift}_1 \boldsymbol{u} \\
&\quad = \mathsf{FFTConv}\big(\boldsymbol{x}, \mathsf{Carry}(\boldsymbol{h})\big) + 2\boldsymbol{V} - \mathsf{FFTConv}(\boldsymbol{x}, \mathsf{CShift}_1 \boldsymbol{u}) && \text{linearity (26)} \\
&\quad = \mathsf{FFTConv}\big(\boldsymbol{x}, \mathsf{Carry}(\boldsymbol{h})\big) + 2\boldsymbol{V} - \mathsf{CShift}_1 \boldsymbol{V} && \text{shift compatibility (28)} \\
&\quad = \mathsf{FFTConv}\big(\boldsymbol{x}, \mathsf{Carry}(\boldsymbol{h})\big) + 2\boldsymbol{V} - \mathsf{Shift}\,\boldsymbol{V} && \mathsf{Shift}\,\boldsymbol{V} = \mathsf{CShift}_1 \boldsymbol{V}.
\end{aligned}$$

The first transition propagates the carry certificate through the FFT convolution (Lemma 4.3). Thus the two convolutions differ by $2\boldsymbol{V} - \mathsf{Shift}\,\boldsymbol{V}$. By the carry rule (14), whose premise is $V_{N-1} = 0$, adding this difference merely increases the carry out of each position j by V_j and leaves every output bit unchanged. Propagating the chain through the carry circuit and applying this rule gives

$$\begin{aligned}\mathsf{Carry}\big(\mathsf{FFTConv}(\boldsymbol{x},\boldsymbol{h})\big) &= \mathsf{Carry}\big(\mathsf{FFTConv}(\boldsymbol{x},\mathsf{Carry}(\boldsymbol{h})) + 2\boldsymbol{V} - \mathsf{Shift}\,\boldsymbol{V}\big)\\ &= \mathsf{Carry}\big(\mathsf{FFTConv}(\boldsymbol{x},\mathsf{Carry}(\boldsymbol{h}))\big),\end{aligned}$$

which is the second equality. Later, in the formal technical sections, Lemma I.2 proves this deferral for the rounded FFT convolution of the actual multiplier.

Premises. Three of the carry rules above have premises: the final carry of $\mathsf{Carry}(\boldsymbol{y}+\boldsymbol{z})$ vanishes for (11), the coefficient V_{N-1} vanishes for (14), and the final carry of $\mathsf{Carry}\big(\mathsf{FFTConv}(\boldsymbol{x},\boldsymbol{y}) + \mathsf{FFTConv}(\boldsymbol{x},\boldsymbol{z})\big)$ vanishes for (12). All three follow from the leading zeros of the inputs: the bit vectors $\boldsymbol{x}$, $\boldsymbol{y}$, and $\boldsymbol{z}$ vanish from position n onward, with $2n + \lceil\log(2N+1)\rceil \le N$. We derive them from bounds on carries and coefficients, each proved by unrolling an induction over the positions of a carry chain or over the nodes of a sum tree, using the order laws for short words (assumption 3). Recall that the carry chain on a vector $\boldsymbol{h}$ starts with the carry $\kappa_0 = 0$ and passes on $\kappa_{j+1} = \lfloor (h_j + \kappa_j)/2 \rfloor$.

- *The carries of* $\mathsf{Carry}(\boldsymbol{y}+\boldsymbol{z})$. The entries of $\boldsymbol{y}+\boldsymbol{z}$ are at most 2, so every carry is at most 1, since $\lfloor (2+1)/2 \rfloor = 1$. From position n on, the entries vanish, so the carries vanish from position $n+1$ on, since $\lfloor (0+1)/2 \rfloor = 0$. In particular, the final carry vanishes, and so does every entry $u_j = \kappa_{j+1}$ of the carry vector with $j \ge n$.
- *The coefficient* V_{N-1}. By item 1 at the coordinate $N-1$, $V_{N-1} = \mathsf{Conv}_{N-1}(\boldsymbol{x},\boldsymbol{u}) = \sum_{j<N} x_j u_{N-1-j}$. Every product in this sum has a zero factor: $x_j = 0$ if $j \ge n$, and otherwise $N-1-j \ge N-n \ge n$, so $u_{N-1-j} = 0$. Hence $V_{N-1} = 0$.
- *The final carry of* $\mathsf{Carry}\big(\mathsf{FFTConv}(\boldsymbol{x},\boldsymbol{y}) + \mathsf{FFTConv}(\boldsymbol{x},\boldsymbol{z})\big)$. By item 1, the k-th coefficient of $\mathsf{FFTConv}(\boldsymbol{x},\boldsymbol{y})$ is the column sum $\mathsf{Conv}_k(\boldsymbol{x},\boldsymbol{y}) = \sum_{j<N} x_j y_{k-j}$ of N products of bits, so it lies between 0 and N. It vanishes for $k \ge r := 2n-1$, since every product then has a zero factor: $x_j = 0$ if $j \ge n$, and otherwise $k - j \ge n$, so $y_{k-j} = 0$. The same holds for $\mathsf{FFTConv}(\boldsymbol{x},\boldsymbol{z})$, so the entries of the sum are at most $2N$ and vanish from position r on. Hence every carry is at most $2N$, since $\lfloor (2N+2N)/2 \rfloor = 2N$, and from position r on, each carry is at most half of the previous one. The carries, including the final one, therefore vanish from position $r + \lceil\log(2N+1)\rceil \le N$ on.

The same bounds show that the declared output lengths of the multiplier suffice. Like linearity and shift compatibility, the bounds on the coefficients are derived one coordinate at a time.

Cost. Throughout, the coordinate k is a constant, hardwired into the derivation. In particular, the circuit for $\mathsf{Conv}_k(\boldsymbol{x},\boldsymbol{y})$ reads each entry y_{k-j} directly by wiring and needs no routing circuitry. Each coordinatewise derivation has size $N \log^{O(1)} N$, and summed over the N coordinates, they cost $N^2 \log^{O(1)} N$, which dominates the total.

7 Compressing the Derivation to Quasilinear Size

The naïve derivation is quadratic because it handles the N output coordinates separately. Each coordinate depends on all N inputs, so its derivation has size about $N \log^{O(1)} N$, and there are N of them. Indeed, the N column sums $\mathsf{Conv}_k(\boldsymbol{x},\boldsymbol{y})$ alone form a circuit of size $N^2 \log^{O(1)} N$. Yet the N derivations are essentially

identical: they differ only in the hardwired constant k. The key question is therefore how to avoid paying for each coordinate separately. *Can we write the coordinatewise derivation once, for a variable coordinate, and use it for all N coordinates without making N copies of it*? Our answer has two parts: derivations at a variable index, and index-generalization.

Section 7.1 introduces the circuits at a variable index that we use, explains derivations at a variable index, and collects the lemmas about these circuits that we need. Section 7.2 applies index-generalization to derive, at quasi-linear cost, the facts that the naïve derivation obtained one coordinate at a time, and Section 7.3 combines them into a derivation of distributivity. Section 7.4 discusses associativity, and Section 7.5 sketches how to remove the simplifying assumptions.

7.1 Additional Circuits and Inference Rules for a Variable Index

This subsection introduces the circuits and inference rules at a variable index used in Section 7.2. We work in the simplified setting of the FFT warmup. We prove Corollary 7.4 and lemma 7.5 here and refer to the formal technical sections for the other proofs and their finite-precision versions.

Circuits computing constants at a variable index. To make the coordinate a variable, we replace the constant coordinate by index wires and reason about entries whose position is given by these wires rather than by a constant. Let i, k, and r denote L-bit indices given by wires, with index arithmetic modulo N. We use the following circuits. The last two, the routing circuits, rearrange the entries of an array $\boldsymbol{v}$ of length N.

- *The indicator vector $\boldsymbol{e}_k$.* It is computed by a decoder, which sets $(\boldsymbol{e}_k)_j := [k = j]$ for every $j < N$. Each test $[k = j]$ is a conjunction of L literals, one for each bit of k, so the decoder has $O(N \log N)$ gates.

- *The twiddle computation circuit $\mathfrak{q}_m$.* For each $m = 2^s \le N$, this circuit takes an s-bit address a and computes the coefficient ω_m^a. Writing $a = \sum_{\ell<s} 2^\ell a_\ell$, define

$$\mathfrak{q}_m(a) := \prod_{\ell<s}^{\text{left}} \begin{cases} 1, & a_\ell = 0, \\ \omega_m^{2^\ell}, & a_\ell = 1. \end{cases}$$

  Each factor is selected by a word multiplexer between two hardwired constants; multiply the selected factors from left to right. For $m = 1$, the output is the hardwired value 1. There are $O(\log m)$ selections and multiplications, so the circuit has $\log^{O(1)} N$ gates under the simplifying assumptions. An exponent is reduced modulo m by retaining its low s bits. Below, ω_m^a with a variable exponent abbreviates $\mathfrak{q}_m(a \bmod m)$; the coefficients in the FFT network itself remain hardwired. These twiddle computation circuits are auxiliary to the proof.

- *The character $\boldsymbol{\chi}_k$.* At each position $r < N$, compute the address $(-rk) \bmod N$ and set $(\boldsymbol{\chi}_k)_r := \mathfrak{q}_N((-rk) \bmod N)$. Thus each entry represents ω^{-rk} and has $\log^{O(1)} N$ gates; the whole vector has $N \log^{O(1)} N$ gates.

Lemma 7.1 (Twiddle computation rules)**.** *Fix $m = 2^s \le N$ with $s \ge 1$, let a, b be s-bit addresses, and fix*

$\rho < m$. *In the simplified setting, the twiddle circuits satisfy*

$$\Gamma \rhd_{\mathsf{EF}}^{\log^{O(1)} N} [a = \rho] \implies \mathfrak{q}_m(a) = \omega_m^\rho, \tag{29a}$$

$$\Gamma \rhd_{\mathsf{EF}}^{\log^{O(1)} N} \mathfrak{q}_m(a)\mathfrak{q}_m(b) = \mathfrak{q}_m(a+b), \tag{29b}$$

$$\Gamma \rhd_{\mathsf{EF}}^{\log^{O(1)} N} \overline{\mathfrak{q}_m(a)} = \mathfrak{q}_m(-a), \tag{29c}$$

$$\Gamma \rhd_{\mathsf{EF}}^{\log^{O(1)} N} [a < m/2] \implies \mathfrak{q}_m(a + m/2) = -\mathfrak{q}_m(a), \tag{29d}$$

$$\Gamma \rhd_{\mathsf{EF}}^{\log^{O(1)} N} \mathfrak{q}_m(2a) = \mathfrak{q}_{m/2}(a \bmod (m/2)). \tag{29e}$$

All addresses are reduced modulo the subscript of $\mathfrak{q}$. *In* (29a), ω_m^ρ *is a hardwired coefficient; this rule is for one fixed address* ρ.

We defer the proofs to Appendix E, where Appendix E.1 defines the finite-precision circuit $\mathfrak{q}_m$. Lemma E.1 proves fixed-address agreement with the hardwired coefficients, with total cost $m \log^{O(1)} N$ for all m addresses. Lemma E.3 proves the remaining rules: the product and conjugation identities become approximate, with explicitly bounded errors due to finite precision, while the half-period and doubling identities remain exact. Here we use their exact forms under the FFT warmup's simplifying assumptions.

Routing circuits. We introduce the reflection circuit which reverses an array through fixed wiring, and a cyclic shift circuit that shifts an array by a variable index.

- *The reflection* $\mathsf{Reflect}$. It is the fixed rearrangement of wires given by $(\mathsf{Reflect}\,\boldsymbol{v})_j := v_{-j}$, so it has no gates.
- *The cyclic shift* CShift_k. It is given by $(\mathsf{CShift}_k \boldsymbol{v})_j := v_{j-k}$ and is computed by a barrel shifter [PSW02] that moves words instead of bits. Writing $k = \sum_{t<L} 2^t k_t$, it has L stages. Stage t consists of N word multiplexers, all controlled by the bit k_t, which shift the array cyclically by 2^t positions if $k_t = 1$ and pass it through unchanged otherwise. The shifts selected by the L stages add up to k. In total, the barrel shifter has $NL = O(N \log N)$ word multiplexers, and hence $N \log^{O(1)} N$ gates.

Since $(\mathsf{CShift}_k \mathsf{Reflect}\,\boldsymbol{h})_j = h_{k-j}$, the routed array $\mathsf{CShift}_k \mathsf{Reflect}\,\boldsymbol{h}$ computes the vector $(h_{k-j})_{j<N}$ of (20). Earlier, in Sections 5.3 and 6.2, the index k was a constant rather than a variable, so we simply obtained this vector $(h_{k-j})_{j<N}$ from $\boldsymbol{h}$ by wiring. Likewise, at a constant coordinate k, the column sum $\mathsf{Conv}_k(\boldsymbol{x}, \boldsymbol{y}) = \sum_{j<N} x_j y_{k-j}$ reads each entry y_{k-j} by wiring. At a variable coordinate k, we read these entries from the routed array $\mathsf{CShift}_k \mathsf{Reflect}\,\boldsymbol{y}$ instead, and we define the column sum at a variable coordinate by

$$\mathsf{Conv}_k(\boldsymbol{x}, \boldsymbol{y}) := \langle \boldsymbol{x}, \mathsf{CShift}_k \mathsf{Reflect}\,\boldsymbol{y} \rangle.$$

Since the barrel shifter has $O(N \log N)$ word multiplexers, this column sum has size $N \log^{O(1)} N$, just like a column sum at a fixed coordinate. As far as circuit size is concerned, a derivation at a variable index can thus be as small as one at a fixed coordinate, up to polylogarithmic factors.

Routing-related rules at a variable index. We now introduce big-step rules that handle routing at a variable index and indexing into an array at a variable index. In these rules and in Section 7.2, the indicator vector $\boldsymbol{e}_k$ and the character $\boldsymbol{\chi}_k$ are no longer constants: they are computed by the circuits above from the index wires k.

Proving the rules at a variable index is more delicate than proving their counterparts for a hardwired constant k. When k is a constant, the selected entry, the indicator vector, and the shift are given by wires and constants, so that the rules hold by inspection of the wiring. At a variable index, however, the derivation must

instead be valid for all N values of k at once. For the rules that involve CShift_k, for instance, the derivation follows the structure of the barrel shifter stage by stage. We defer the proofs of Lemmas 7.2 and 7.3 to the formal technical sections, chiefly to Lemmas D.2 and D.4, and we point to the exact counterpart of each identity after its statement.

Lemma 7.2 (Decoders and routing)**.** *For arrays $\boldsymbol{u}$ and $\boldsymbol{v}$ of length N and L-bit indices i and k,*

$$\Gamma \ \rhd_{\mathsf{EF}}^{N \log^{O(1)} N} \ \langle \boldsymbol{e}_k, \boldsymbol{v} \rangle = \mathsf{Sel}(\boldsymbol{v}, k), \tag{30}$$

$$\Gamma \ \rhd_{\mathsf{EF}}^{N \log^{O(1)} N} \ \mathsf{Sel}(\mathsf{CShift}_k \, \boldsymbol{v}, i) = \mathsf{Sel}(\boldsymbol{v}, i-k), \qquad \mathsf{Sel}(\mathsf{Reflect} \, \boldsymbol{v}, i) = \mathsf{Sel}(\boldsymbol{v}, -i), \tag{31}$$

$$\Gamma \ \rhd_{\mathsf{EF}}^{N \log^{O(1)} N} \ \mathsf{Sel}(\mathsf{CShift}_k \mathsf{Reflect} \, \boldsymbol{v}, i) = \mathsf{Sel}(\boldsymbol{v}, k-i), \tag{32}$$

$$\Gamma \ \rhd_{\mathsf{EF}}^{N \log^{O(1)} N} \ \langle \mathsf{CShift}_k \mathsf{Reflect} \, \boldsymbol{u}, \mathsf{CShift}_k \mathsf{Reflect} \, \boldsymbol{v} \rangle = \langle \boldsymbol{u}, \boldsymbol{v} \rangle, \tag{33}$$

where the addresses $i-k$, $-i$, and $k-i$ are computed by short circuits for arithmetic modulo N.

Later, in the formal technical sections, we prove generalizations of these identities: (30) in Lemma D.11, (31), and hence (32), in Lemma D.2, which builds on Lemma D.1, and (33) in Lemma D.10.

Lemma 7.3 (Routing preserves entrywise checks)**.** *Let $\boldsymbol{v}$ be an array of length N, let k be an L-bit index, and let P be a check of size $\log^{O(1)} N$ whose inputs are one entry and an L-bit position. Then*

$$\Gamma : \ [P(v_j, j)]_{j<N} \ \rhd_{\mathsf{EF}}^{N \log^{O(1)} N} \ \big[P\big((\mathsf{CShift}_k \mathsf{Reflect} \, \boldsymbol{v})_j, \, k-j\big)\big]_{j<N}, \tag{34}$$

where the position j in the premises is hardwired, and the position $k-j$ in the conclusions is computed by a short circuit for arithmetic modulo N.

Since $(\mathsf{CShift}_k \mathsf{Reflect} \, \boldsymbol{v})_j = v_{k-j}$, the entry at position j of the routed array is the entry of $\boldsymbol{v}$ at position $k-j$, and the lemma says that it inherits the check from there. The derivation follows the barrel shifter stage by stage, as described above, carrying the check through each word multiplexer by a case distinction on its control bit. Later, in the formal technical sections, Lemma D.4 restates this lemma for arrays of arbitrary words and proves it.

We use the lemma through the following two instances.

Corollary 7.4 (Routing preserves bounds and zeros)**.** *For an array $\boldsymbol{v}$ of length N, an L-bit index k, a constant short word B, and a constant position $n < N$,*

$$\Gamma : \ [0 \le v_j \le B]_{j<N} \ \rhd_{\mathsf{EF}}^{N \log^{O(1)} N} \ \big[0 \le (\mathsf{CShift}_k \mathsf{Reflect} \, \boldsymbol{v})_j \le B\big]_{j<N}, \tag{35}$$

$$\Gamma : \ [v_j = 0]_{n \le j < N} \ \rhd_{\mathsf{EF}}^{N \log^{O(1)} N} \ \big[(k-j \bmod N) \ge n \ \rightarrow \ (\mathsf{CShift}_k \mathsf{Reflect} \, \boldsymbol{v})_j = 0\big]_{j<N}. \tag{36}$$

Proof. Both rules are instances of Lemma 7.3.

- *Bounds.* Take for $P(\xi, p)$ the check $0 \le \xi \le B$, which does not read the position p. Then the premises and conclusions of (34) are those of (35).
- *Zeros.* Take for $P(\xi, p)$ the check $[p \ge n] \rightarrow \xi = 0$. Then the conclusions of (34) are those of (36), and its premises $P(v_j, j)$ follow from the given ones. For $n \le j < N$, the premise $v_j = 0$ gives the consequent. For $j < n$, the antecedent $[j \ge n]$ is a closed check that evaluates to false.

Each of the N premise derivations for the zeros has size $\log^{O(1)} N$, so both rules cost $N \log^{O(1)} N$. □

Later, in the formal technical sections, Lemma D.4 also states the analogue of (35) for norm bounds, and the proof of Lemma G.2 applies Lemma D.4 with the two checks above to the routed operand of a column sum.

Selecting an output at a variable index. The derivations in Section 7.2 compare Fourier-network outputs at an arbitrary index r, then generalize over r. Selecting output coordinate r corresponds to applying row r of the Fourier matrix, whose entries are $(\omega^{rj})_{j<N}$, to the input vector. The following lemma expresses this selected output as a weighted sum of the input entries. This lets us apply the array rules above without unfolding the network separately for each output coordinate.

Lemma 7.5 (Selected rows of the Fourier networks)**.** *For a vector $\boldsymbol{v}$ of length N and an L-bit index r,*

$$\textit{selected-row identity}: \quad \Gamma \ \rhd_{\mathsf{EF}}^{N \log^{O(1)} N} \ \mathsf{Sel}\big(\mathsf{FFT}(\boldsymbol{v}), r\big) = \sum_{j<N} \omega^{rj} v_j. \tag{37}$$

The right-hand side is the direct Fourier sum*: compute each ω^{rj} by $\mathfrak{q}_N(rj \bmod N)$, multiply it by v_j, and add the N products along a balanced tree. The same identity holds for $\overline{\mathsf{FFT}}$, with ω^{-rj} in place of ω^{rj}.*

Proof. We follow the recursion of the network, as in the derivation of (16). At a node of size m of the recursion tree, the network calls FFT_m on a subvector $\boldsymbol{v}'$ of $\boldsymbol{v}$. The row $r' := r \bmod m$ at this node consists of the $\log m$ low-order bits of r, so it is given by wiring. Let

$$D_m(\boldsymbol{v}') := \sum_{j<m} \omega_m^{r'j} v'_j$$

be the direct Fourier sum at the node, using $\mathfrak{q}_m(r'j \bmod m)$ for each coefficient. At the root this circuit is supplied by the conclusion; at every other node we construct it once and reuse its outputs. The claim at the node is the equation

$$\mathsf{Sel}\big(\mathsf{FFT}_m(\boldsymbol{v}'), r'\big) = D_m(\boldsymbol{v}'). \tag{R_m}$$

At the root, where $m = N$, $\boldsymbol{v}' = \boldsymbol{v}$, and $r' = r$, the claim is (37). The EF derivation proves the claims at all nodes from the leaves up, citing at each node the claims already derived at its two children.

Base case. At a leaf, where $m = 1$, the selector outputs v'_0, and $D_1(\boldsymbol{v}') = 1 \cdot v'_0 = v'_0$ by the identity law.

Inductive step. Consider a node of size $m \geq 2$. Let $r'' := r \bmod (m/2)$ be the row at its two children, and let $\boldsymbol{E} := \mathsf{FFT}_{m/2}(\boldsymbol{v}'^{\mathrm{e}})$ and $\boldsymbol{O} := \mathsf{FFT}_{m/2}(\boldsymbol{v}'^{\mathrm{o}})$ be the outputs of their calls. First, the butterflies at the node give

$$\mathsf{Sel}\big(\mathsf{FFT}_m(\boldsymbol{v}'), r'\big) = \mathsf{Sel}(\boldsymbol{E}, r'') + \omega_m^{r'} \mathsf{Sel}(\boldsymbol{O}, r'').$$

Indeed, for each fixed row $\rho < m$, if $r' = \rho$, then selection at a fixed address (Lemma 4.5) identifies the three selected entries with $\mathsf{FFT}_m(\boldsymbol{v}')_\rho$, $E_{\rho \bmod (m/2)}$, and $O_{\rho \bmod (m/2)}$. The fixed-address rule (29a) of Lemma 7.1 identifies the computed coefficient $\mathfrak{q}_m(r')$ with the hardwired value ω_m^ρ. For $\rho < m/2$, the displayed equality is the defining butterfly formula. For $\rho \geq m/2$, the half-period rule (29d), with fixed-address agreement, gives $\omega_m^\rho = -\omega_m^{\rho - m/2}$, matching the subtraction branch. These are m cases, each of size $\log^{O(1)} N$, and proof by enumeration (Lemma 4.4) combines them. Second, regrouping (9) splits the direct sum into its even and odd terms, and thus

$$D_m(\boldsymbol{v}') = \sum_{j<m/2} \omega_m^{2r'j} v'_{2j} + \sum_{j<m/2} \omega_m^{r'(2j+1)} v'_{2j+1} = D_{m/2}(\boldsymbol{v}'^{\mathrm{e}}) + \omega_m^{r'} D_{m/2}(\boldsymbol{v}'^{\mathrm{o}}).$$

For the second equality, the doubling rule (29e) of Lemma 7.1 gives $\omega_m^{2r'j} = \omega_{m/2}^{r''j}$, after index arithmetic identifies the two child addresses modulo $m/2$. The product rule (29b) gives $\omega_m^{r'(2j+1)} = \omega_m^{r'} \omega_m^{2r'j}$. Propagate these equalities through the products and sum trees using Lemma 4.3, and factor out $\omega_m^{r'}$ by distributivity (assumption 3). By the claims $(\mathrm{R}_{m/2})$ at the two children, the right-hand sides of the last two displays agree, which proves (R_m).

Cost. At a node of size m, the missing selectors, address circuits, twiddle computations, products, and sum trees have $m \log^{O(1)} N$ gates. Each is constructed once and reused in the comparisons above. The $O(m)$ applications of the rules in Lemma 7.1 cost $m \log^{O(1)} N$ in total; enumeration, regrouping, and equality propagation also fit this bound. The nodes of each size together contain N entries, so each of the $\log N + 1$ sizes costs $N \log^{O(1)} N$, and the whole derivation has size $N \log^{O(1)} N$. The derivation for $\overline{\mathsf{FFT}}$ uses the conjugation rule (29c) to identify each conjugated computed coefficient with $\mathfrak{q}_m(-a)$, and then the same argument applies. □

Later, in the formal technical sections, Lemma F.1 proves this identity without the simplifying assumptions, by the same induction over the recursion tree. There, the twiddle factors are fixed-point approximations of the roots of unity, so the identities of roots of unity hold only up to a tiny error (Lemma E.3). In particular, factoring the twiddle factor out of the odd part of the direct sum introduces an error at each node. The induction tracks these errors, and at the root the two sides of (37) differ by at most the product of a bound on the entries of $\boldsymbol{v}$, a polynomial factor in N, and the precision of the stored roots. Lemma F.1 states that bound, with the error notation introduced there.

7.2 Applying the Index-Generalization Paradigm

Index-generalization turns a derivation at a variable index into assertions for all N indices, without paying for N copies of the derivation (Lemmas 3.4 and 4.7). We next apply this paradigm repeatedly to derive distributivity of the FFT multiplier over the schoolbook adder in quasi-linear size. Some of these applications are nested; Remark 7.12 at the end of this section explains why this is allowed and why the size stays quasi-linear.

Routing identities. At a constant shift, the following identities are immediate, since the routing circuits are then rearrangements of wires. At a variable shift k, we verify them entry by entry.

Lemma 7.6 (Routing identities)**.** *For vectors $\boldsymbol{g}$, $\boldsymbol{g}'$, and $\boldsymbol{u}$ of length N and L-bit indices k and r,*

$$\Gamma \ \rhd^{N \log^{O(1)} N}_{\mathsf{EF}} \ \mathsf{CShift}_k\mathsf{Reflect}(\boldsymbol{g} + \boldsymbol{g}') = \mathsf{CShift}_k\mathsf{Reflect}\,\boldsymbol{g} + \mathsf{CShift}_k\mathsf{Reflect}\,\boldsymbol{g}', \tag{38}$$

$$\Gamma \ \rhd^{N \log^{O(1)} N}_{\mathsf{EF}} \ \mathsf{CShift}_k\mathsf{Reflect}\,\mathsf{CShift}_1\boldsymbol{u} = \mathsf{CShift}_{k-1}\mathsf{Reflect}\,\boldsymbol{u}, \tag{39}$$

$$\Gamma \ \rhd^{N \log^{O(1)} N}_{\mathsf{EF}} \ \mathsf{CShift}_k\mathsf{Reflect}\,\left(\omega^{-r(k-j)}\right)_{j<N} = \left(\omega^{-rj}\right)_{j<N}. \tag{40}$$

Here, the shift $k-1$ is computed from k by a short circuit for subtraction modulo N. In (40), the coefficients $\omega^{-r(k-j)}$ and ω^{-rj} are outputs of the circuit $\mathfrak{q}_N$ defined above, at addresses $-r(k-j) \bmod N$ and $-rj \bmod N$, respectively.

Proof. Each identity is an equality $\boldsymbol{A} = \boldsymbol{B}$ between two arrays of length N, which we verify entry by entry. The base context contains the vectors and indices of the statement and the arrays on both sides. At an arbitrary L-bit index i, we derive that the two selected entries agree, $\mathsf{Sel}(\boldsymbol{A}, i) = \mathsf{Sel}(\boldsymbol{B}, i)$, and then we generalize over i.

For (38), we use the selector routing (32) and that selection commutes with coordinatewise sums (Lemma 4.6):

$$\begin{aligned}
\mathsf{Sel}\big(\mathsf{CShift}_k\mathsf{Reflect}(\boldsymbol{g}+\boldsymbol{g}'), i\big) &= \mathsf{Sel}(\boldsymbol{g}+\boldsymbol{g}', k-i) && (32)\\
&= \mathsf{Sel}(\boldsymbol{g}, k-i) + \mathsf{Sel}(\boldsymbol{g}', k-i) && \text{Lemma 4.6}\\
&= \mathsf{Sel}(\mathsf{CShift}_k\mathsf{Reflect}\,\boldsymbol{g}, i) + \mathsf{Sel}(\mathsf{CShift}_k\mathsf{Reflect}\,\boldsymbol{g}', i) && (32) \text{ for } \boldsymbol{g} \text{ and } \boldsymbol{g}'\\
&= \mathsf{Sel}\big(\mathsf{CShift}_k\mathsf{Reflect}\,\boldsymbol{g} + \mathsf{CShift}_k\mathsf{Reflect}\,\boldsymbol{g}', i\big) && \text{Lemma 4.6.}
\end{aligned}$$

For (39), we use the selector routing (32) for the shifts k and $k-1$, and (31) for the constant shift by one:

$$\begin{aligned}
\mathsf{Sel}(\mathsf{CShift}_k\mathsf{Reflect}\,\mathsf{CShift}_1\boldsymbol{u}, i) &= \mathsf{Sel}(\mathsf{CShift}_1\boldsymbol{u}, k-i) && (32)\\
&= \mathsf{Sel}(\boldsymbol{u}, k-i-1) && (31)\\
&= \mathsf{Sel}\big(\boldsymbol{u}, (k-1)-i\big) && \text{index arithmetic}\\
&= \mathsf{Sel}(\mathsf{CShift}_{k-1}\mathsf{Reflect}\,\boldsymbol{u}, i) && (32) \text{ for the shift } k-1.
\end{aligned}$$

For (40), let $\boldsymbol{w}' := (\omega^{-r(k-j)})_{j<N}$ and $\boldsymbol{w} := (\omega^{-rj})_{j<N}$. Their entry circuits compute the addresses $-r(k-j) \bmod N$ and $-rj \bmod N$, respectively, and apply q_N. Selection commutes with these circuits by Lemma 4.6. Thus

$$\begin{aligned}
\mathsf{Sel}(\mathsf{CShift}_k\mathsf{Reflect}\,\boldsymbol{w}', i) &= \mathsf{Sel}(\boldsymbol{w}', k-i) && (32)\\
&= \omega^{-r(k-(k-i))} && \text{Lemma 4.6}\\
&= \omega^{-ri} && \text{index arithmetic}\\
&= \mathsf{Sel}(\boldsymbol{w}, i) && \text{Lemma 4.6.}
\end{aligned}$$

In the transitions marked "index arithmetic," the arithmetic of short words identifies two addresses modulo N, and equality propagation carries this through the selector or the twiddle computation. Each chain has size $N \log^{O(1)} N$.

Each chain derives a check of size $\log^{O(1)} N$ that compares the entries of two arrays selected at i, whereas Lemma 3.4 concerns a check on the entries of a single array. So here we need the general form of index-generalization, and we apply Lemma 4.7 over i. It gives $A_j = B_j$ for every position $j < N$, and folding these equalities (Lemma 4.1) gives $\boldsymbol{A} = \boldsymbol{B}$, at total cost $N \log^{O(1)} N$. □

Impulse and shift identities at a variable index. The impulse and shift identities for FFT for a constant index k were described earlier in (19) and (20). To compress the distributivity proof to quasi-linear size, we need to introduce their counterparts where k is now a variable index:

Lemma 7.7 (Impulse and shift identities at a variable index)**.** *For a vector $\boldsymbol{h}$ of length N and an L-bit index k,*

$$\textit{impulse identity}: \quad \Gamma \ \rhd_{\mathsf{EF}}^{N\log^{O(1)}N} \ \overline{\mathsf{FFT}}(\boldsymbol{e}_k) = \boldsymbol{\chi}_k, \tag{41}$$

$$\textit{shift identity}: \quad \Gamma \ \rhd_{\mathsf{EF}}^{N\log^{O(1)}N} \ \overline{\mathsf{FFT}}(\mathsf{CShift}_k\mathsf{Reflect}\,\boldsymbol{h}) = \boldsymbol{\chi}_k \odot \mathsf{FFT}(\boldsymbol{h}). \tag{42}$$

Proof. Each identity asserts the equality of two vectors that depend on k. Every entry of the left side depends on all N entries of the input of the network, as in $\overline{\mathsf{FFT}}(\boldsymbol{e}_k)_r = \sum_{j<N} \omega^{-rj}(\boldsymbol{e}_k)_j = \omega^{-rk}$, so deriving the identity one row at a time would cost $N \log^{O(1)} N$ per row, and $N^2 \log^{O(1)} N$ in total. Instead, we derive it at an arbitrary L-bit row index r and generalize over the row. The base context contains $\boldsymbol{h}$, k, and the arrays on both sides. Let $\boldsymbol{w} := (\omega^{-rj})_{j<N}$ be the vector of twiddle factors at row r. By the selected-row identity (37) for $\overline{\mathsf{FFT}}$, $\mathsf{Sel}(\overline{\mathsf{FFT}}(\boldsymbol{v}), r) = \langle \boldsymbol{w}, \boldsymbol{v} \rangle$ for every vector $\boldsymbol{v}$.

For (41), we derive

$$\begin{aligned}
\mathsf{Sel}\big(\overline{\mathsf{FFT}}(\boldsymbol{e}_k), r\big) &= \langle \boldsymbol{e}_k, \boldsymbol{w} \rangle && \text{selected-row identity (37)}\\
&= \mathsf{Sel}(\boldsymbol{w}, k) && \text{decoder pairing (30)}\\
&= \omega^{-rk} = \mathsf{Sel}(\boldsymbol{\chi}_k, r) && \text{selection through the twiddle computation, Lemma 4.6.}
\end{aligned}$$

The first transition also commutes the two factors of each product in the pairing (assumption 3 and Lemma 4.3). The last one uses that selection commutes with the twiddle computation, which reads the index (Lemma 4.6): the entry of $\boldsymbol{w}$ at k and the entry of $\boldsymbol{\chi}_k$ at r are both outputs of q_N at the address $(-rk) \bmod N$.

For (42), let $\boldsymbol{w}' := (\omega^{-r(k-j)})_{j<N}$, so that $\boldsymbol{w} = \mathsf{CShift}_k\mathsf{Reflect}\,\boldsymbol{w}'$ by (40). We derive

$$
\begin{aligned}
&\mathsf{Sel}\big(\overline{\mathsf{FFT}}(\mathsf{CShift}_k\mathsf{Reflect}\,\boldsymbol{h}), r\big) \\
&\quad = \langle \boldsymbol{w}, \mathsf{CShift}_k\mathsf{Reflect}\,\boldsymbol{h}\rangle && \text{selected-row identity (37)} \\
&\quad = \langle \mathsf{CShift}_k\mathsf{Reflect}\,\boldsymbol{w}', \mathsf{CShift}_k\mathsf{Reflect}\,\boldsymbol{h}\rangle && \text{(40) and Lemma 4.3} \\
&\quad = \langle \boldsymbol{w}', \boldsymbol{h}\rangle = \sum_{j<N} \omega^{-r(k-j)} h_j && \text{pairing routing (33)} \\
&\quad = \omega^{-rk} \sum_{j<N} \omega^{rj} h_j && \text{(29b), distributivity} \\
&\quad = \omega^{-rk}\,\mathsf{Sel}\big(\mathsf{FFT}(\boldsymbol{h}), r\big) && \text{selected-row identity (37)} \\
&\quad = \mathsf{Sel}\big(\boldsymbol{\chi}_k \odot \mathsf{FFT}(\boldsymbol{h}), r\big) && \text{selection through the product.}
\end{aligned}
$$

The fourth transition applies (29b) of Lemma 7.1 at addresses $-rk$ and rj. Index arithmetic identifies their sum with $-r(k-j)$ modulo N, giving $\omega^{-r(k-j)} = \omega^{-rk}\omega^{rj}$. Propagate this equality through each of the N products and factor out ω^{-rk} at every node of the sum tree by distributivity. The twiddle-rule applications cost $N \log^{O(1)} N$ altogether. The last transition uses $\mathsf{Sel}(\boldsymbol{\chi}_k, r) = \omega^{-rk}$, as above, and that selection commutes with the coordinatewise product (Lemma 4.6).

Each of the two derivations has size $N \log^{O(1)} N$, including the application of Lemma 7.6 at the row r, which itself generalizes over a coordinate index. Both conclusions compare the entries of two arrays selected at r, so here, too, we need the general form of index-generalization. We apply Lemma 4.7 over r and fold the resulting row equalities (Lemma 4.1). This gives both identities, at total cost $N \log^{O(1)} N$. □

Later, in the formal technical sections, Lemma F.4 proves approximate versions of these identities. It states them only at a variable index, which also covers the constant indices of Section 5.3.

The convolution theorem at a variable coordinate. Recall that at a variable coordinate k, the column sum is $\mathsf{Conv}_k(\boldsymbol{x}, \boldsymbol{h}) = \langle \boldsymbol{x}, \mathsf{CShift}_k\mathsf{Reflect}\,\boldsymbol{h}\rangle$.

Lemma 7.8 (Convolution theorem at a variable coordinate)**.** *For vectors $\boldsymbol{x}$ and $\boldsymbol{h}$ of length N and an L-bit index k,*

$$\Gamma \;\rhd_{\mathsf{EF}}^{N \log^{O(1)} N}\; \mathsf{Sel}\big(\mathsf{FFTConv}(\boldsymbol{x}, \boldsymbol{h}), k\big) = \mathsf{Conv}_k(\boldsymbol{x}, \boldsymbol{h}). \tag{43}$$

Proof. We follow the chain (23) of the naïve derivation, which now starts from the selected entry rather than from the k-th output wire:

$$
\begin{aligned}
\mathsf{Sel}\big(\mathsf{FFTConv}(\boldsymbol{x}, \boldsymbol{h}), k\big) &= \big\langle \boldsymbol{e}_k, \mathsf{IFFT}(\mathsf{FFT}(\boldsymbol{x}) \odot \mathsf{FFT}(\boldsymbol{h}))\big\rangle && \text{(i) decoder pairing (30)} \\
&= \tfrac{1}{N}\big\langle \overline{\mathsf{FFT}}(\boldsymbol{e}_k), \mathsf{FFT}(\boldsymbol{x}) \odot \mathsf{FFT}(\boldsymbol{h})\big\rangle && \text{(ii) transposition (17)} \\
&= \tfrac{1}{N}\big\langle \boldsymbol{\chi}_k, \mathsf{FFT}(\boldsymbol{x}) \odot \mathsf{FFT}(\boldsymbol{h})\big\rangle && \text{(iii) impulse identity (41)} \\
&= \tfrac{1}{N}\big\langle \mathsf{FFT}(\boldsymbol{x}), \boldsymbol{\chi}_k \odot \mathsf{FFT}(\boldsymbol{h})\big\rangle && \text{(iv) regrouping} \\
&= \tfrac{1}{N}\big\langle \mathsf{FFT}(\boldsymbol{x}), \overline{\mathsf{FFT}}(\mathsf{CShift}_k\mathsf{Reflect}\,\boldsymbol{h})\big\rangle && \text{(v) shift identity (42)} \\
&= \big\langle \boldsymbol{x}, \mathsf{CShift}_k\mathsf{Reflect}\,\boldsymbol{h}\big\rangle = \mathsf{Conv}_k(\boldsymbol{x}, \boldsymbol{h}) && \text{(vi) Parseval-type identity (16).}
\end{aligned}
$$

Four of the six transitions carry over directly. Transitions (ii) and (vi) are identities of the FFT warmup that hold for all inputs of the networks, so the index plays no role in them. Transition (i) is the decoder pairing,

which holds at a variable index, and transition (iv) regroups each product by the short-word laws, as in the naïve derivation. Transitions (iii) and (v) are where the variable index matters. They use the impulse and shift identities at a variable index (Lemma 7.7), in which $\boldsymbol{e}_k$ and $\boldsymbol{\chi}_k$ are computed from the index wires and the vector $(h_{k-j})_{j<N}$ of (23) is the routed array $\mathsf{CShift}_k\mathsf{Reflect}\,\boldsymbol{h}$. Equality propagation (Lemma 4.3) carries these identities into the pairings. Each transition costs $N\log^{O(1)} N$. □

Later, in the formal technical sections, Lemma G.1 proves this lemma for the actual multiplier. There, the selected output of the FFT convolution is only close to the integer $\mathsf{Conv}_k(\boldsymbol{x},\boldsymbol{h})$, and rounding it gives exactly $\mathsf{Conv}_k(\boldsymbol{x},\boldsymbol{h})$ (Section 7.5).

The conclusion (43) concerns the single coordinate selected by k. Each of the next three lemmas generalizes over k a derivation that uses it.

Linearity, shift compatibility, and coefficient bounds. The next three lemmas derive, at quasi-linear cost, the remaining facts that the naïve derivation obtained one coordinate at a time: linearity of the FFT convolution (26), shift compatibility (28), and the bounds on the coefficients that supply the premises of the carry rules.

Lemma 7.9 (Linearity of the FFT convolution)**.** *For vectors $\boldsymbol{x}$, $\boldsymbol{g}$, and $\boldsymbol{g}'$ of length N,*

$$\Gamma \;\rhd_{\mathsf{EF}}^{N\log^{O(1)} N}\; \mathsf{FFTConv}(\boldsymbol{x},\boldsymbol{g}\pm\boldsymbol{g}') = \mathsf{FFTConv}(\boldsymbol{x},\boldsymbol{g}) \pm \mathsf{FFTConv}(\boldsymbol{x},\boldsymbol{g}'),$$
$$\Gamma \;\rhd_{\mathsf{EF}}^{N\log^{O(1)} N}\; \mathsf{FFTConv}(\boldsymbol{x},2\boldsymbol{g}) = 2\,\mathsf{FFTConv}(\boldsymbol{x},\boldsymbol{g}).$$

Proof. We derive the identity for sums. As in Section 6.2, the other two follow from it, applied to $\boldsymbol{g}-\boldsymbol{g}'$ and $\boldsymbol{g}'$, whose sum is $\boldsymbol{g}$, and to $\boldsymbol{g}$ and $\boldsymbol{g}$, since $2\boldsymbol{g}=\boldsymbol{g}+\boldsymbol{g}$; equality propagation (Lemma 4.3) carries these coordinatewise identities through the FFT convolution. The base context contains $\boldsymbol{x}$, $\boldsymbol{g}$, $\boldsymbol{g}'$, and the three FFT convolutions $\mathsf{FFTConv}(\boldsymbol{x},\boldsymbol{g}+\boldsymbol{g}')$, $\mathsf{FFTConv}(\boldsymbol{x},\boldsymbol{g})$, and $\mathsf{FFTConv}(\boldsymbol{x},\boldsymbol{g}')$. At an arbitrary L-bit coordinate k, we derive

$$\begin{aligned}
&\mathsf{Sel}\big(\mathsf{FFTConv}(\boldsymbol{x},\boldsymbol{g}+\boldsymbol{g}'),k\big)\\
&\quad= \big\langle \boldsymbol{x}, \mathsf{CShift}_k\mathsf{Reflect}(\boldsymbol{g}+\boldsymbol{g}')\big\rangle && \text{Lemma 7.8}\\
&\quad= \big\langle \boldsymbol{x}, \mathsf{CShift}_k\mathsf{Reflect}\,\boldsymbol{g} + \mathsf{CShift}_k\mathsf{Reflect}\,\boldsymbol{g}'\big\rangle && \text{routing split (38)}\\
&\quad= \big\langle \boldsymbol{x}, \mathsf{CShift}_k\mathsf{Reflect}\,\boldsymbol{g}\big\rangle + \big\langle \boldsymbol{x}, \mathsf{CShift}_k\mathsf{Reflect}\,\boldsymbol{g}'\big\rangle && \text{distributivity, regrouping}\\
&\quad= \mathsf{Sel}\big(\mathsf{FFTConv}(\boldsymbol{x},\boldsymbol{g}),k\big) + \mathsf{Sel}\big(\mathsf{FFTConv}(\boldsymbol{x},\boldsymbol{g}'),k\big) && \text{Lemma 7.8.}
\end{aligned}$$

The second transition propagates the routing split into the pairing (Lemma 4.3). The third applies short-word distributivity to each of the N products and regroups the sum by (9), as in the chain (25). The derivation has size $N\log^{O(1)} N$, including the applications of Lemmas 7.6 and 7.8 with their own applications of index-generalization. Its conclusion is a check of size $\log^{O(1)} N$ on the entries of three arrays selected at the same index k. So we need the general form of index-generalization, and we apply Lemma 4.7 over k. It gives the N coordinate equalities, and folding them (Lemma 4.1) gives the identity for sums, at total cost $N\log^{O(1)} N$. □

Here the paradigm pays off most visibly: the final derivation contains a single column sum $\mathsf{Conv}_k(\boldsymbol{x},\boldsymbol{g})$, at the variable coordinate k, in place of the N column sums of the naïve derivation, and similarly for the other two convolutions.

Lemma 7.10 (Shift compatibility)**.** *For vectors $\boldsymbol{x}$ and $\boldsymbol{u}$ of length N,*

$$\Gamma \;\rhd_{\mathsf{EF}}^{N\log^{O(1)} N}\; \mathsf{FFTConv}(\boldsymbol{x},\mathsf{CShift}_1\boldsymbol{u}) = \mathsf{CShift}_1\,\mathsf{FFTConv}(\boldsymbol{x},\boldsymbol{u}).$$

Proof. Write $\boldsymbol{V} := \mathsf{FFTConv}(\boldsymbol{x}, \boldsymbol{u})$. The base context contains $\boldsymbol{x}$, $\boldsymbol{u}$, and the arrays $\mathsf{FFTConv}(\boldsymbol{x}, \mathsf{CShift}_1\boldsymbol{u})$ and $\mathsf{CShift}_1\boldsymbol{V}$. For a fixed coordinate k, the column sums $\mathsf{Conv}_k(\boldsymbol{x}, \mathsf{CShift}_1\boldsymbol{u})$ and $\mathsf{Conv}_{k-1}(\boldsymbol{x}, \boldsymbol{u})$ are the same circuit. At an arbitrary L-bit coordinate k, however, they read $\boldsymbol{u}$ through two different routings, which the routing identity (39) identifies:

$$\begin{aligned}
\mathsf{Sel}\big(\mathsf{FFTConv}(\boldsymbol{x}, \mathsf{CShift}_1\boldsymbol{u}), k\big) &= \langle \boldsymbol{x}, \mathsf{CShift}_k\mathsf{Reflect}\,\mathsf{CShift}_1\boldsymbol{u}\rangle && \text{Lemma 7.8}\\
&= \langle \boldsymbol{x}, \mathsf{CShift}_{k-1}\mathsf{Reflect}\,\boldsymbol{u}\rangle && \text{routing identity (39)}\\
&= \mathsf{Sel}(\boldsymbol{V}, k-1) && \text{Lemma 7.8 at the index } k-1\\
&= \mathsf{Sel}(\mathsf{CShift}_1\boldsymbol{V}, k) && \text{selector routing (31).}
\end{aligned}$$

The second transition propagates the routing identity into the pairing (Lemma 4.3). The derivation has size $N \log^{O(1)} N$. Its conclusion compares the entries of two arrays selected at k, so we need the general form of index-generalization: we apply Lemma 4.7 over k and fold the coordinate equalities (Lemma 4.1). This gives the identity, at total cost $N \log^{O(1)} N$. □

Later, in the formal technical sections, Lemma H.1 proves the analogous identity for the rounded FFT convolution of the actual multiplier.

Lemma 7.11 (Coefficient bounds)**.** *Let $\boldsymbol{x}$ be a bit vector of length N, let $\boldsymbol{g}$ be a vector of length N, and let n be a position with $2n \le N$. Put $r := 2n - 1$. Then*

$$\Gamma :\ [0 \le g_j \le 1]_{j<N},\ [x_j = 0,\ g_j = 0]_{n\le j<N} \ \rhd_{\mathsf{EF}}^{N \log^{O(1)} N}$$
$$\big[0 \le \mathsf{FFTConv}(\boldsymbol{x}, \boldsymbol{g})_j \le N\big]_{j<N}, \qquad \big[\mathsf{FFTConv}(\boldsymbol{x}, \boldsymbol{g})_j = 0\big]_{r\le j<N}.$$

Proof. The base context contains $\boldsymbol{x}$, $\boldsymbol{g}$, $\mathsf{FFTConv}(\boldsymbol{x}, \boldsymbol{g})$, and the premises. At an arbitrary L-bit coordinate k, by Lemma 7.8, the selected coefficient $\xi := \mathsf{Sel}(\mathsf{FFTConv}(\boldsymbol{x}, \boldsymbol{g}), k)$ equals the column sum

$$\mathsf{Conv}_k(\boldsymbol{x}, \boldsymbol{g}) = \sum_{j<N} x_j (\mathsf{CShift}_k\mathsf{Reflect}\,\boldsymbol{g})_j,$$

and equality propagation (Lemma 4.3) transfers checks on the column sum to ξ. We derive two such checks.

- *Magnitude.* By (35) for $B = 1$, the routed entries $(\mathsf{CShift}_k\mathsf{Reflect}\,\boldsymbol{g})_j$ lie between 0 and 1. Since the entries of $\boldsymbol{x}$ are bits, each product lies between 0 and 1, and an induction over the sum tree, unrolled from the leaves, shows that the column sum lies between 0 and N (assumption 3). This derives the check $0 \le \xi \le N$. The check reads only a single entry of the single array $\mathsf{FFTConv}(\boldsymbol{x}, \boldsymbol{g})$, so the basic form of index-generalization suffices: Lemma 3.4, applied over k, gives $0 \le \mathsf{FFTConv}(\boldsymbol{x}, \boldsymbol{g})_j \le N$ for every position $j < N$.

- *Support.* If $k \ge r$, every product vanishes. Indeed, $x_j = 0$ if $j \ge n$. If $j < n$, then $k \ge j$, so the index $k - j$ does not wrap around, and $k - j \ge r - (n-1) = n$, so the routed entry vanishes by (36). These steps are short-word derivations under the antecedent $k \ge r$, and the laws for adding zeros, applied along the sum tree, show that the column sum vanishes under it (assumption 3 and Lemma 4.2). Transferring this check from the column sum to ξ by Lemma 4.3, again in conditional form (Lemma 4.2), derives $[k \ge r] \to \xi = 0$. The check also reads the index k, so we need the general form of index-generalization, which allows checks that read the index. We apply Lemma 4.7 over k. It gives the check $[j \ge r] \to \mathsf{FFTConv}(\boldsymbol{x}, \boldsymbol{g})_j = 0$ for every position $j < N$, with j hardwired. For $r \le j < N$, the antecedent is a closed check that evaluates to true, which gives $\mathsf{FFTConv}(\boldsymbol{x}, \boldsymbol{g})_j = 0$.

Each of the two derivations at k has size $N \log^{O(1)} N$, so the total cost is $N \log^{O(1)} N$. □

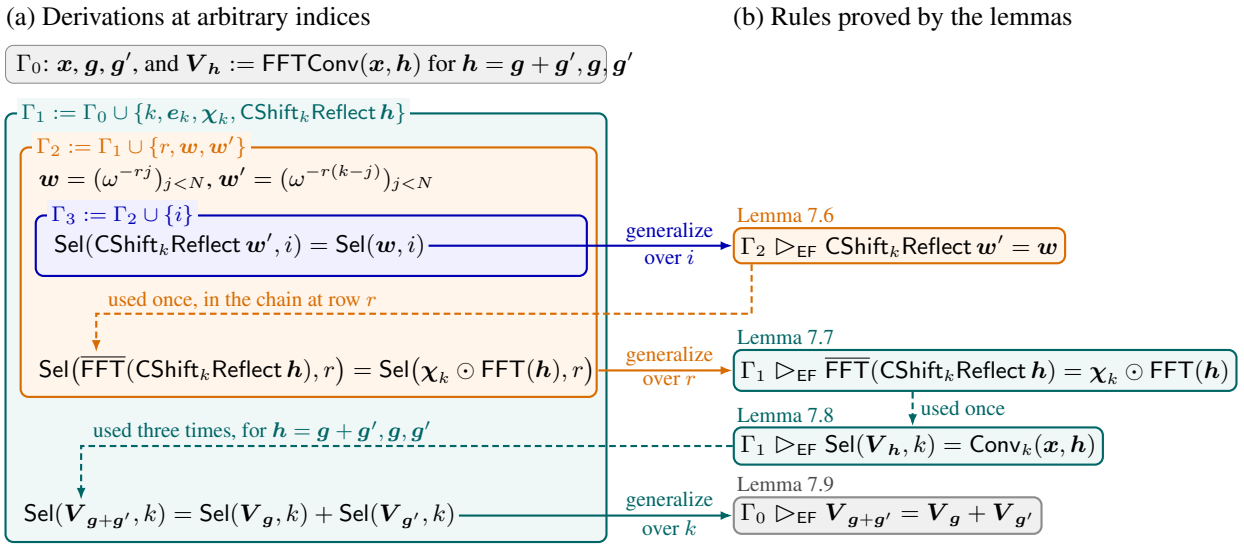


Figure 6: Nested index-generalization in the proof of Lemma 7.9, along one chain of lemmas. (a) Each box is a derivation in the context defined in its tag. This context adds an index and circuits computed from it to the previous context, and the derivation assumes nothing about that index. (b) Each solid arrow generalizes the last line of a box over its index (Lemma 4.7), and the resulting rule holds in the previous context. Each dashed arrow points to where a rule is used. Lemma 7.8 has no solid arrow, since its proof applies index-generalization only through Lemma 7.7.

Later, in the formal technical sections, Lemma G.2 proves these bounds for the rounded FFT convolution, given bounds on the input entries and indices from which each input array is zero.

Remark 7.12 (Nested applications)**.** Several applications of index-generalization in this section are nested: the derivation that one application generalizes contains another application. For instance, Lemma 7.9 derives linearity at a variable coordinate k and then generalizes over k. Its derivation at k uses the convolution theorem at k (Lemma 7.8), which in turn uses the impulse and shift identities at k (Lemma 7.7). These identities compare two vectors that depend on k, and Lemma 7.7 derives them at a variable row r and then generalizes over r. Thus a generalization over r takes place inside the derivation that is generalized over k (Figure 6). Two facts make this work.

- *The inner application is allowed.* Index-generalization only requires its premise to hold at an arbitrary index (Section 3.2). The derivation at r meets this requirement: it assumes nothing about r, even though its context also contains k and the vectors computed from it, such as $\boldsymbol{e}_k$ and $\boldsymbol{\chi}_k$. Generalizing over r removes only r from the context, so the resulting rule may still mention k, and the derivation at k uses it through Lemma 7.8 (Figure 6).

- *The cost stays quasi-linear.* Each solid arrow in Figure 6 adds only $N \log^{O(1)} N$ to the size of the derivation that it generalizes (Lemma 4.7). Each use of a rule, along a dashed arrow, adds the full cost of the rule to the derivation that uses it. The derivation at r uses the rule obtained by generalizing over i once. The derivation at k applies Lemma 7.8 three times, to $\boldsymbol{g} + \boldsymbol{g}'$, $\boldsymbol{g}$, and $\boldsymbol{g}'$, and each application uses the rule obtained over r once. Hence each level of nesting multiplies the size of the level below it by a constant, the number of such uses, and adds $N \log^{O(1)} N$. Our derivations have at most three levels, over a coordinate k, a row r, and a further index i, since Lemma 7.7 itself uses Lemma 7.6, which generalizes over i.

Lemmas 7.10 and 7.11 have the same structure as Lemma 7.9, so the same two facts apply to them. Each generalizes over k a derivation that applies Lemma 7.8: twice in Lemma 7.10, at k and at the computed index $k-1$, and once in Lemma 7.11. Lemma 7.10 also uses the routing identity (39), which generalizes over i, and Lemma 7.11 generalizes over k twice, once for each of its two checks. So all derivations in this section have size $N \log^{O(1)} N$.

7.3 Putting Everything Together

We now derive distributivity at quasi-linear cost. As in Section 6.2, we use the leading zeros of the inputs: the bit vectors $\boldsymbol{x}$, $\boldsymbol{y}$, and $\boldsymbol{z}$ vanish from position n onward, with $2n + \lceil \log(2N+1) \rceil \le N$.

Proposition 7.13 (Distributivity in quasi-linear size)**.** *For such a position n,*

$$\Gamma : \ [x_j = 0,\ y_j = 0,\ z_j = 0]_{n \le j < N} \ \rhd_{\mathsf{EF}}^{N \log^{O(1)} N}$$
$$\mathsf{FFTMul}\big(x, \mathsf{ADD}(y,z)\big) = \mathsf{ADD}\big(\mathsf{FFTMul}(x,y), \mathsf{FFTMul}(x,z)\big).$$

Proof. We derive the chain of the naïve derivation,

$$\begin{aligned}
&\mathsf{FFTMul}\big(x, \mathsf{ADD}(y,z)\big) \\
&\quad = \mathsf{Carry}\big(\mathsf{FFTConv}(\boldsymbol{x}, \mathsf{Carry}(\boldsymbol{y}+\boldsymbol{z}))\big) && \text{carry rule (11)} \\
&\quad = \mathsf{Carry}\big(\mathsf{FFTConv}(\boldsymbol{x}, \boldsymbol{y}+\boldsymbol{z})\big) && \text{deferring carry resolution} \\
&\quad = \mathsf{Carry}\big(\mathsf{FFTConv}(\boldsymbol{x}, \boldsymbol{y}) + \mathsf{FFTConv}(\boldsymbol{x}, \boldsymbol{z})\big) && \text{Lemma 7.9} \\
&\quad = \mathsf{ADD}\big(\mathsf{FFTMul}(x,y), \mathsf{FFTMul}(x,z)\big) && \text{carry rule (12),}
\end{aligned}$$

now with the lemmas above in place of its coordinatewise derivations. As in Section 6.2, the first transition propagates the carry rule (11) through the multiplier (Lemma 4.3), and the last one is the carry rule (12). The third transition is Lemma 7.9 for $\boldsymbol{g} = \boldsymbol{y}$ and $\boldsymbol{g}' = \boldsymbol{z}$, propagated through the carry circuit.

The second transition is the deferral of Section 6.2. Let $\boldsymbol{h} := \boldsymbol{y} + \boldsymbol{z}$, let $\boldsymbol{u}$ be the carry vector of $\mathsf{Carry}(\boldsymbol{h})$, and put $\boldsymbol{V} := \mathsf{FFTConv}(\boldsymbol{x}, \boldsymbol{u})$. Starting from the carry certificate (13), the deferral derives

$$\mathsf{FFTConv}(\boldsymbol{x}, \boldsymbol{h}) = \mathsf{FFTConv}\big(\boldsymbol{x}, \mathsf{Carry}(\boldsymbol{h})\big) + 2\boldsymbol{V} - \mathsf{Shift}\,\boldsymbol{V}.$$

It uses linearity for a sum, a difference, and a doubled vector, now supplied by Lemma 7.9, and shift compatibility, now supplied by Lemma 7.10. It then applies the carry rule (14) to the vector $\mathsf{FFTConv}(\boldsymbol{x}, \mathsf{Carry}(\boldsymbol{h}))$, which removes the difference $2\boldsymbol{V} - \mathsf{Shift}\,\boldsymbol{V}$ after carry resolution.

It remains to derive the premises, from the same bounds as in Section 6.2.

- *The carries of* $\mathsf{Carry}(\boldsymbol{y}+\boldsymbol{z})$. The entries of $\boldsymbol{y}+\boldsymbol{z}$ are at most 2 and vanish from position n on, so every carry is at most 1, and the carries vanish from position $n+1$ on. This is an induction along the carry chain, unrolled over its N positions, at cost $N \log^{O(1)} N$. Hence the final carry vanishes, as the carry rule (11) requires. Moreover, the entries of the carry vector $\boldsymbol{u}$ lie between 0 and 1 and vanish from position n on. In particular, $u_{N-1} = 0$, so $\mathsf{Shift}\,\boldsymbol{u} = \mathsf{CShift}_1 \boldsymbol{u}$, as the deferral requires.

- *The coefficient* V_{N-1}. By the previous item, $\boldsymbol{u}$ satisfies the premises of Lemma 7.11, which gives $V_j = 0$ for $r \le j < N$, where $r = 2n-1 \le N-1$. In particular, $V_{N-1} = 0$, which is the premise of the carry rule (14) and gives $\mathsf{Shift}\,\boldsymbol{V} = \mathsf{CShift}_1 \boldsymbol{V}$.

- *The final carry of* $\mathsf{Carry}\big(\mathsf{FFTConv}(\boldsymbol{x}, \boldsymbol{y}) + \mathsf{FFTConv}(\boldsymbol{x}, \boldsymbol{z})\big)$. By Lemma 7.11 for $\boldsymbol{g} = \boldsymbol{y}$ and for $\boldsymbol{g} = \boldsymbol{z}$, the entries of $\mathsf{FFTConv}(\boldsymbol{x}, \boldsymbol{y}) + \mathsf{FFTConv}(\boldsymbol{x}, \boldsymbol{z})$ are at most $2N$ and vanish from position r on. As in Section 6.2, an induction along the carry chain shows that every carry is at most $2N$ and that the carries vanish from position $r + \lceil \log(2N+1) \rceil \le N$ on. In particular, the final carry vanishes, as the carry rule (12) requires.

The same bounds show that the declared output lengths of the multiplier suffice. In total, the derivation applies Lemmas 7.9 to 7.11 a constant number of times. Otherwise, it consists of the carry rules, of inductions along carry chains, and of equality propagations through circuits of size $N \log^{O(1)} N$. Its size is therefore $N \log^{O(1)} N$. □

7.4 Associativity and Other Applications

Associativity of multiplication uses the same tools. The new difficulty is that the outer multiplication receives the output of the inner one, so comparing both sides with schoolbook multiplication, as we did for distributivity, would expand a double convolution sum with quadratically many terms per coordinate. Instead, we compare the transform of the inner output with the pointwise product that produced it. The inner output is $\mathsf{IFFT}\big(\mathsf{FFT}(\boldsymbol{x}) \odot \mathsf{FFT}(\boldsymbol{y})\big)$, and the cancellation identity (18) returns that pointwise product when FFT is applied to it again. By equality propagation (Lemma 4.3), each side of associativity therefore equals IFFT of the triple pointwise product $\mathsf{FFT}(\boldsymbol{x}) \odot \mathsf{FFT}(\boldsymbol{y}) \odot \mathsf{FFT}(\boldsymbol{z})$, bracketed one way or the other, and the two agree by coordinatewise associativity of multiplication. Deferring carry resolution, as in Section 6.2, completes the proof; the paradigm enters through that deferral.

Associativity of addition, by contrast, is verified column by column in linear size. Beyond the uses described above, the paradigm appears wherever an identity between length-N arrays is naturally verified entry by entry, for example, in the identities for the routing circuits CShift_k and $\mathsf{Reflect}$ (Lemma D.3).

7.5 Removing the Simplifying Assumptions

In the actual implementation, the twiddle factors are hardwired approximations of the roots of unity with $\log^{O(1)} N$ fractional bits, computed once by the circuit generator. All other arithmetic is exact: every operation retains its full result, so word widths grow to $\log^{O(1)} N$ bits, and approximation enters only through these constants. Linearity, the transposition identity, and the routing identities therefore remain exact. The identities that depend on the accuracy of the roots—the product law for roots of unity, the selected-row identity, and cancellation—hold only up to a small error. Each of them is again a comparison circuit on finite words, derived in EF with its error tracked explicitly, and the hardwired constants themselves are certified by evaluating closed check circuits.

Since the convolution output is then only close to the integer vector of column sums, the multiplier rounds each output entry to the nearest integer before resolving carries. The accumulated error stays below $1/4$, so rounding returns exactly that column sum, and an integer gap—two integer vectors at distance less than one are equal—carries the identities above from the unrounded outputs to the rounded ones. The structure of the argument is unchanged. In particular, index-generalization applies as before, since the check P may be any small circuit, including an approximate comparison. The formal sections carry out this accounting, with the error notation introduced there (Appendices E and G).

8 Additional Related Work

Frege and EF. As mentioned earlier, EF extends the Frege system. In a Frege proof, every line is a propositional formula, obtained from a fixed finite set of axiom schemes by modus ponens, and the proof size counts the symbols of all its lines. In comparison with EF, Frege lacks the *extension* rule, which lets a proof introduce a fresh variable e together with the line $e \leftrightarrow \phi$ for an arbitrary formula ϕ, and then use e wherever ϕ is meant. The extension rule was introduced to shorten proofs. Every Frege proof is already an EF proof, so EF polynomially simulates Frege; whether Frege polynomially simulates EF is a long-standing open question.

Since the work of Cook and Reckhow [CR79], proving superpolynomial lower bounds for increasingly strong proof systems has been a central program of proof complexity [Kra95, Kra19]. For Frege and EF, no superpolynomial lower bounds are known, and the best known lower bounds are only $\Omega(n^2)$ symbols and $\Omega(n)$ lines for tautologies of size n [Bus95, Bus21]. Feasible interpolation, a technique that gives lower bounds for some weaker proof systems, fails for EF and Frege under cryptographic assumptions [KP98, BPR00].

There are a couple of proof systems that are known to be equivalent to EF up to a polynomial overhead. For instance, a system called Circuit Frege (CF) and EF simulate each other with quadratic overhead when the conclusions are formulas [Jeř07]. Moreover, if we allow the conclusion to be written by a check circuit in the context, the simulation of CF by EF incurs only a linear overhead (see [Jeř07, Proof of Lemma 2.5]). Also, Substitution Frege (SF) and EF are equivalent up to polynomial overhead [Bus21]. We focus on Extended Frege due to its fine-grained connection to cryptographic applications (see [MDS25] for a detailed exposition).

Bounded arithmetic and feasible mathematics. Besides Cook's propositional translation, PV proves the soundness of EF, so EF polynomially simulates every proof system whose soundness PV proves [Coo75]. These correspondences extend to Buss's theories of bounded arithmetic [Bus86, KP90]. A large body of work asks which mathematical and complexity-theoretic statements have feasible proofs, from linear algebra [SC04] and approximate counting [Jeř07] to the PCP theorem [Pic15] and circuit lower bounds [Raz95a, MP20]. These results have applications to the unprovability of complexity bounds [Raz95b, KO17, PS21, LO23, CLO24].

Deferred Formal Technical Details

Our main technical ideas have been described in the informal roadmap. For completeness, we now provide the full circuit implementations and EF derivations. The main added complication is the precision management and error tracking, since we are using finite precision to represent complex numbers in the FFT multiplier.

A The Implementation Circuits

Fix an input length $n \geq 1$. All logarithms are to base two. Write $[m] = \{0, \ldots, m-1\}$.

A *word* of width w consists of w Boolean wires, indexed from its least significant bit. An array of length m consists of m words. Unsigned words use the usual binary encoding; signed words use two's complement.

A.1 Integer Circuit Primitives

This subsection defines circuits on integer bit strings. The unsigned adder ADD is used both on short words and on full input integers of $O(n)$ bits. The other primitives are applied (in our FFT-multiplier implementation) only to short words, whose widths are polylogarithmic in n.

Full-result arithmetic. The addition, subtraction, multiplication, negation, and absolute-value circuits below retain enough bits in their intermediate and output words to represent their exact results without overflow. Each operation specifies its output width, fixed when the circuit is constructed.

Unsigned integer arithmetic. An unsigned integer of width w is represented as a binary string of w bits. We define the following operations.

- *Addition (ripple-carry adder).* For unsigned strings of lengths r, s, ADD is the ripple-carry adder with output length $\max(r, s) + 1$. Missing input bits are zero. If a_j, b_j are its bits and γ_j its carry, its gates are

  $$\gamma_0 := 0, \qquad d_j := a_j \oplus b_j \oplus \gamma_j, \qquad \gamma_{j+1} := (a_j b_j) \vee (a_j \gamma_j) \vee (b_j \gamma_j).$$

  Products of bits in this display denote AND gates. The last output bit is the last carry. This is the addition implementation in the final identities. The circuit uses $O(r + s)$ Boolean gates.

- *Zero-extension.* To zero-extend an unsigned w-bit word to width $w' \geq w$, add $w' - w$ zeros in the new high positions. This preserves the represented integer. Output bits $0, \ldots, w-1$ are wired to the corresponding input bits, and the remaining output bits are hardwired to zero. The operation uses only wiring, with no new Boolean gates and $O(w')$ output bits.

Signed integer arithmetic. Signed words use *two's-complement encoding*: a w-bit word $b_{w-1} \cdots b_0$ represents the integer

$$-b_{w-1} 2^{w-1} + \sum_{j=0}^{w-2} b_j 2^j.$$

The highest bit b_{w-1} is the sign bit: zero for a nonnegative value and one for a negative value. The representable range is -2^{w-1} to $2^{w-1} - 1$.

The following circuits retain full integer results at the widths stated below.

- *Sign extension.* To *sign-extend* a word, copy its sign bit into each added high position; this increases its width without changing its integer value. For example, extending from four to five bits sends 0011 to 00011 (both represent 3) and 1101 to 11101 (both represent -3). Sign extension, like zero extension, uses only wiring, with no new Boolean gates and $O(w')$ output bits when extending to width $w' \geq w$.

- *Addition* (AddInt). For signed inputs of widths r, s, AddInt sign-extends both inputs to width $\max(r, s) + 1$ and uses the ripple-carry gates defined above. Its signed output retains $\max(r, s) + 1$ bits.

- *Negation* (Neg). To negate a signed r-bit integer, first extend it by one sign bit, complement the bits, and add one with the adder above. The signed output retains $r + 1$ bits, accommodating the negation of the most negative input.

- *Subtraction* (SubInt). For signed inputs of widths r, s, SubInt negates the second input as above, sign-extends the first input and the negated second input to the common width $\max(r, s) + 1$, and adds them with ripple-carry gates. Its signed output retains $\max(r, s) + 1$ bits.

- *Absolute value* (Abs). For a signed r-bit input, Abs uses the original sign bit to select the negated input or the original input sign-extended to $r + 1$ bits. The result is nonnegative and retains $r + 1$ signed bits, accommodating the absolute value of the most negative input. The selection uses Mux and requires no order comparator.

- *Multiplication* (MulInt). For signed inputs of widths r, s, MulInt forms the unsigned partial products of their absolute values, sums the shifted rows with a balanced tree of ripple adders, and restores the sign by conditional negation. Its signed output retains $r + s$ bits.

These circuits are applied to the short numerators of scaled values in Appendices A.2 and A.3. For input and working widths at most w, addition, subtraction, negation, and absolute value use $O(w)$ Boolean gates; schoolbook multiplication uses $O(w^2)$.

Multiplication by powers of two. For a signed or unsigned w-bit integer x and fixed $k \geq 0$, implement both $2^k x$ and $x2^k$ by a left shift with $w + k$ output bits. Output bit j is hardwired to zero for $j < k$ and wired to x_{j-k} for $k \leq j < w + k$. Thus the circuit appends k low zero bits and retains the full integer result, with the same signed or unsigned interpretation. It uses only wiring, with no new Boolean gates and $O(w + k)$ output bits.

Integer quotient and remainder by powers of two. For a signed w-bit integer x and fixed $0 \leq k \leq w$, the expressions $\lfloor x/2^k \rfloor$ and $x \bmod 2^k$ denote the following circuits. The quotient is implemented by a right shift with sign extension, retaining w signed bits: output bit j is wired to x_{j+k} when $j + k < w$ and to the sign bit x_{w-1} otherwise. The remainder is implemented by the low-bit wiring $(x_0, \ldots, x_{k-1})$, interpreted as unsigned; for $k = 0$, it is a single hardwired zero bit. The quotient rounds toward negative infinity, and the remainder is in $\{0, \ldots, 2^k - 1\}$, including when x is negative. Both operations use only wiring, with $O(w)$ output bits. For unsigned inputs, use zero extension in the quotient instead of sign extension. If $k > w$, first extend the input to width k.

Binary carry resolution. This circuit converts a vector of integer coefficients into a binary string. It is the carry resolution of Section 4.3, generalized from nonnegative to signed coefficients. For a length-m vector $\boldsymbol{a}$ of signed integer coefficients, compute the following chain:

$$\kappa_0 := 0, \qquad h_j := a_j + \kappa_j, \qquad b_j := h_j \bmod 2, \qquad \kappa_{j+1} := \lfloor h_j/2 \rfloor \quad (j < m).$$

Define the binary output by

$$\boldsymbol{b} := \mathsf{Carry}(\boldsymbol{a}) = (b_0, \ldots, b_{m-1}).$$

Thus b_j is the bit of weight 2^j. The sums h_j and carries κ_j are intermediate words; the terminal carry κ_m is retained as an auxiliary output but is not an entry of $\boldsymbol{b}$.

The chain is schoolbook carrying with a word in place of a carry bit: a coefficient may exceed one, so what passes to the next column is the entire part of h_j above its low bit. Every column therefore satisfies

$$a_j + \kappa_j = h_j = b_j + 2\kappa_{j+1}, \qquad b_j \in \{0, 1\},$$

an identity about integers that assumes nothing about the sign of h_j, because the quotient of the preceding paragraph rounds toward negative infinity and its remainder is a genuine bit; for instance $-3 = 2 \cdot (-2) + 1$. Telescoping these columns gives

$$\sum_{j<m} a_j 2^j = \sum_{j<m} b_j 2^j + 2^m \kappa_m.$$

A negative coefficient is thus absorbed by passing a negative carry, that is, a borrow, to the next column, and the terminal carry is nonzero when the represented integer does not fit in m bits, and negative when that integer is negative. For the multiplier's coefficient arrays, Appendix I derives the carry bounds and $\kappa_m = 0$, so that the returned bits encode the full integer $\sum_{j<m} a_j 2^j$.

Use a common signed width w for the coefficients a_j and carries κ_j, and width $w + 1$ for $h_j := \mathsf{AddInt}(a_j, \kappa_j)$. The quotient and remainder use the circuits defined above, and κ_{j+1} retains w bits by omitting the quotient's repeated high sign bit: halving a $(w + 1)$-bit signed value lands in the w-bit signed range, so the top two bits of $\lfloor h_j/2 \rfloor$ agree and dropping one preserves its value. The chain therefore runs at a fixed width, and its carries do not grow with m. The circuit contains m signed adders, each using $O(w)$ Boolean gates; the quotient, remainder, and output selection use only wiring. Its total size is therefore $O(mw)$, with m binary output bits and the auxiliary w-bit terminal carry.

A.2 Scaled Real Arithmetic Circuits

Scale and width. A scaled real number is a fixed-point representation of reals: it has a nonnegative integer scale f and a signed numerator of width $w \geq 1$. Write

$$\mathcal{D}_{f,w} = \{(A; f) : A \text{ is a signed } w\text{-bit numerator}\}, \qquad (A; f) \text{ represents } A2^{-f}.$$

Write $\mathrm{sc}(x) = f$ and $\mathrm{wd}(x) = w$. Neither parameter is an input wire. The abbreviation $x \in \mathcal{D}_f$ specifies only the scale; the width is supplied by its input declaration or by the circuit producing x. An array may be sign-extended to a common component width before routing or selection. These extensions preserve values and do not discard bits. Complex values use the separate type $\mathcal{C}_{f,w}$ defined in Appendix A.3.

Throughout the paper, a scale is written f or g, decorated as needed: primes f', g' name further scales of the same kind, subscripts name a distinguished or indexed one, as in the input scale f_0, a per-level family f_s, and the raw-output scale f_* of Appendix A.5. Widths are written w with the same freedom, and Greek letters are reserved for error bounds. Scales named inside a proof are local to it.

Every arithmetic operator below retains its full result. Its output scale and width are determined by its operand types, without a magnitude hypothesis. Equality of scaled values means: promote to a common scale, sign-extend to a common width, and apply the bitwise equality check. Thus equal values may have different original encodings. The gate proofs of arithmetic identities include these padding equivalences.

Promotion between scales. For $g \geq f$, promotion is the exact wiring

$$\mathsf{Prmt}_{f\to g} : \mathcal{D}_{f,w} \longrightarrow \mathcal{D}_{g,w+g-f}, \qquad (A; f) \longmapsto (2^{g-f}A; g).$$

Recall from Appendix A.1 that multiplication by a power of two is implemented by a left shift. Sign extension appends high sign bits while leaving the scale unchanged. Promotion and sign extension preserve the represented value. Write $\mathsf{Prmt}_g(x)$ when the input scale is already known. Promotion acts entrywise on arrays.

The operators below act on scaled real values by applying the integer primitives of Appendix A.1 to their numerators. Their cost is polynomial in their input and output widths. In this paper these widths are polylogarithmic in n; Appendix A.5.4 establishes this for the multiplier, and the proofs check the corresponding bounds when introducing auxiliary circuits. Carry resolution operates along an array of short words.

Addition, subtraction, and multiplication. For input widths w_1, w_2, addition and subtraction have signatures

$$\mathsf{Add}_f, \mathsf{Sub}_f : \mathcal{D}_{f,w_1} \times \mathcal{D}_{f,w_2} \longrightarrow \mathcal{D}_{f,\max(w_1,w_2)+1}.$$

Their numerator circuits are AddInt and SubInt. Both operands must have scale f; promotion is explicit. Multiplication has signature

$$\mathsf{Mul}_{f,g} : \mathcal{D}_{f,w_1} \times \mathcal{D}_{g,w_2} \longrightarrow \mathcal{D}_{f+g,w_1+w_2},$$

and numerator circuit MulInt. These are full-result operations on every input bit string. We suppress width indices on the operator names because the input wires and these rules determine them.

In longer formulas we use the compact operator notation

$$x +_f y := \mathsf{Add}_f(x, y), \qquad x -_f y := \mathsf{Sub}_f(x, y), \qquad x \cdot_{f,g} y := \mathsf{Mul}_{f,g}(x, y).$$

Juxtaposition xy abbreviates $\mathsf{Mul}_{\mathrm{sc}(x),\mathrm{sc}(y)}(x, y)$, with output scale $\mathrm{sc}(x) + \mathrm{sc}(y)$, except for the fixed power-of-two scalar factors specified below.

Multiplication and exact division by powers of 2. For $x = (A; f) \in \mathcal{D}_{f,w}$ and fixed $k \geq 0$, define

$$2^k x = x2^k := (2^k A; f) \in \mathcal{D}_{f,w+k},$$
$$x/2^k = 2^{-k}x = x2^{-k} := (A; f + k) \in \mathcal{D}_{f+k,w}.$$

Multiplication uses the integer left shift of Appendix A.1 on the numerator, retaining the scale. Exact division retains the numerator bits and moves the binary point k places to the left: only the scale increases. Both operations use only wiring and discard no bits, with $w + k$ output bits for multiplication and w for division. For an integer input, use scale zero; first zero-extend an unsigned input by one bit to give it a signed numerator. For integer x, the quotient $\lfloor x/2^k \rfloor$ of Appendix A.1 instead rounds down and returns an integer.

This convention includes powers of two that are hardwired named parameters of the construction rather than literals. An explicitly written MulInt or $\mathsf{Mul}_{f,g}$ always denotes the full multiplier, allowing its output to be compared with the specialized wiring in Lemma C.1.

Rule for an omitted operator index. Compile arithmetic expressions from their named inputs upwards, retaining their indicated parentheses. For $x \pm y$, put $k = \max(\mathrm{sc}(x), \mathrm{sc}(y))$ and use $\mathsf{Prmt}_k(x) \pm_k \mathsf{Prmt}_k(y)$. If their input widths are w_x, w_y and scales f, g, the promoted widths are $w_x + k - f, w_y + k - g$, so the sum or difference has width $\max(w_x + k - f, w_y + k - g) + 1$. General products add scales and numerator widths. No operation in this convention discards high numerator bits.

For example, for $x \in \mathcal{D}_f$, $y \in \mathcal{D}_g$, and $g \le f$,

$$x +_f \mathsf{Prmt}_f(y)$$

names Add_f after promoting y. The circuit $\mathsf{Prmt}_g(x +_f \mathsf{Prmt}_f(y))$ first adds in $\mathcal{D}_f$ and then promotes to $\mathcal{D}_g$, for $g \ge f$. It differs from $\mathsf{Prmt}_g(x) +_g \mathsf{Prmt}_g(y)$; their equality is proved by the inference rule in Appendix C.

Unary negation and absolute value preserve the scale and increase the signed numerator width by one, accommodating the most negative input. A selector first promotes its alternatives to the same scale and sign-extends them to the same width, then routes their bits. Integer literals have scale zero and the least sufficient signed width, with at least two bits; a bit coefficient uses two signed bits. The constant 2^{-r} has numerator 1 and scale r. Apart from the power-of-two scalar operations above, scalar products and powers use the full product operators. These rules apply to numerical values; modular address arithmetic and the long binary integer interfaces have their separately specified encodings.

Rounding to an integer. Define $\mathsf{Round}(x) := \lfloor x + 1/2 \rfloor$. For $x = (A; f) \in \mathcal{D}_{f,w}$ with $f \ge 1$, compute $T := \mathsf{AddInt}(A, 2^{f-1})$ in $b = \max(w, f+1) + 1$ signed bits, and compute $\lfloor T/2^f \rfloor$ using the quotient circuit of Appendix A.1. Use its low $b - f$ output bits as Q; these are precisely $T_f, \ldots, T_{b-1}$. The remaining f quotient-output bits are only repeated copies of the sign bit T_{b-1}. Thus Q is the exact integer quotient and the output is $(Q; 0) \in \mathcal{D}_{0,b-f}$. Only the low bits $T_0, \ldots, T_{f-1}$ of the original sum are discarded. At $f = 0$, rounding is the identity. For a real vector, apply Round coordinatewise. Rounding has no fixed destination width and discards no high bits.

Sizes of the scaled operators. Real operations use a constant number of the integer primitives. Promotion and sign extension are wiring; rounding uses an adder and a fixed quotient shift. Their circuit sizes are polynomial in the largest working width. The arithmetic operations are exact on every input; the rounding inference rule in Appendix C proves its stated relation to a nearby integer.

Widths for carry resolution. Sign-extend the rounded coefficients to a common signed width w. Each carry has width w, each full sum has width $w + 1$, and taking its upper w bits computes the integer quotient by two. Thus no high bits are discarded inside the carry chain. Its terminal carry and the omitted binary output bits are handled in Appendix I.

A.3 Scaled Complex Arithmetic Circuits

Complex words. Put $\mathcal{C}_{f,w} = \mathcal{D}_{f,w} \times \mathcal{D}_{f,w}$. Its two signed numerators encode $(A + \mathrm{i}B)2^{-f}$. We sometimes write $\mathcal{C}_f$ for short, omitting the width parameter, without risk of ambiguity. A real value x is embedded as $x + \mathrm{i}0$, with both components at the same scale and width.

Addition, subtraction, and multiplication. Complex addition and subtraction apply the real operators componentwise, with output width $\max(w_1, w_2) + 1$. Complex multiplication has signature

$$\mathsf{Mul}_{f,g} : \mathcal{C}_{f,w_1} \times \mathcal{C}_{g,w_2} \longrightarrow \mathcal{C}_{f+g,w_1+w_2+1}.$$

Its full numerators are

$$\mathsf{SubInt}(\mathsf{MulInt}(A, B), \mathsf{MulInt}(A', B')), \qquad \mathsf{AddInt}(\mathsf{MulInt}(A, B'), \mathsf{MulInt}(A', B)).$$

Each component product has $w_1 + w_2$ bits and each component sum has one more bit. All these bits are retained.

Multiplication and exact division by powers of 2. For $x \in \mathcal{C}_{f,w}$ and fixed $k \geq 0$, apply the real operations of Appendix A.2 to both components:

$$2^k x,\ x2^k \in \mathcal{C}_{f,w+k}, \qquad x/2^k = 2^{-k}x = x2^{-k} \in \mathcal{C}_{f+k,w}.$$

Multiplication shifts both numerators left by k while keeping the scale. Exact division keeps both numerators and their widths unchanged and increases the scale by k. These operations use only wiring, with $2(w + k)$ output bits for multiplication and $2w$ for division; no complex multiplier is constructed.

Promotion, conjugation, and notation. Promotion increases scale and width by the same amount. Negation increases both component widths by one; conjugation negates the imaginary numerator and sign-extends the real numerator to the same width. Selection preserves the common padded type. The notation $+_f$, $-_f$, and juxtaposition applies to complex operations too. Other mixed operations first embed the real operand. All signatures are exact on every input.

Vector addition and $\odot$ act coordinatewise. Rounding and carry resolution act on real entries. A real bit coefficient is embedded with two signed bits and a zero imaginary component. For an $m = 2^s$ FFT, the output of forward level i has scale $f_0 + it$; its width is obtained independently from the full-result operator rules. A balanced addition tree keeps its scale while adding one numerator bit per level. The width recurrence for the butterflies is given in Appendix A.5.

Norm notation. For a complex value $z = x + \mathrm{i}y$, write $|z|_1 = |x| + |y|$. For an array $\boldsymbol{v}$, the assertion $\|\boldsymbol{v}\|_\infty \leq e$ means that $|v_j|_1 \leq e$ for every coordinate j. These are numerical bounds used to state correctness; their Boolean check circuits are defined with the EF inference rules in Appendix C.1. They add no gates to the multiplier.

A.4 FFT Transform Parameters and Complex Constants

FFT parameters. Everything from here on is governed by four parameters. Put

$$N = 2^{\lceil \log(16(n+1)) \rceil}, \qquad L = \log N, \qquad p = 1024(L+1)^2, \qquad t = Lp.$$

The parameters mean the following:

- N is the *transform length*: the number of coefficients in the arrays the multiplier transforms, taken as a power of two so that the FFT recursion halves exactly, and large enough that a nested product of three n-bit inputs still occupies fewer than N coefficients (Appendix A.5.4 checks $3n + 2L + 4 < N$). Thus $N = O(n)$, and the constant 16 leaves room for every carry and zero-padding margin used below, so that no step needs a tight estimate.
- L is the *number of butterfly levels*: both transforms are L-level networks, and every hardwired table is indexed by L-bit addresses.
- p is the *precision of the hardwired roots of unity*: each stored root is given with p fractional bits, and it is the only source of approximation in the construction, since every other operation retains its exact result.

- Finally $t = Lp$ is the *twiddle scale*: a twiddle factor is a product of at most L stored roots, each with p fractional bits, so Lp fractional bits represent it exactly. It is the common scale of the twiddle table constructed below and the scale increment of one butterfly (Appendix A.5).

We now specify the hardwired constants in the FFT/IFFT multiplier and a finite algorithm for precomputing their bits.

Hardwired roots of unity (in $\mathcal{C}_{p,p+2}$). Set $h = 2^{-p} \in \mathcal{D}_p$. We construct scaled complex constants $z_1, \ldots, z_L \in \mathcal{C}_p$. Each component of z_s, real or imaginary, has a $(p+2)$-bit numerator, consisting of $p+1$ numeric bits and one sign bit, and represents a multiple of h in $[-1, 1]$. The constant z_s approximates the primitive 2^s-th root of unity

$$\omega_{2^s} = e^{2\pi \mathrm{i}/2^s} = \cos(2\pi/2^s) + \mathrm{i}\sin(2\pi/2^s)$$

of (15): thus z_1 approximates -1, z_2 approximates i, z_3 approximates $(1+\mathrm{i})/\sqrt{2}$, and each further constant halves the angle. The first two are exact: take $z_1 := \mathsf{Prmt}_p(-1)$ and $z_2 := \mathsf{Prmt}_p(\mathrm{i})$, where the two integer complex constants initially have scale zero. The remaining constants $z_3, \ldots, z_L$ are precomputed by the circuit generator as follows. Write $z_s = a + \mathrm{i}b$ for $s \geq 2$, and let $z_{s+1} := u + \mathrm{i}v$, where $u, v \in \mathcal{D}_p$ are the largest multiples of h with $u^2 \leq (1+a)/2$ and $2uv \leq b$; written in the scales of the precomputed values, these conditions read

$$u^2 \leq \mathsf{Prmt}_{2p}\big((\mathsf{Prmt}_p(1) +_p a)/2\big) < (u +_p h)^2, \tag{44}$$

$$2uv \leq \mathsf{Prmt}_{2p}(b) < 2u(v +_p h). \tag{45}$$

The circuit generator finds the numerator of each by binary search over $[0, 2^p]$, comparing integers only. This precomputation runs once, when the circuit family is generated, and the resulting constants are then hardwired in the FFT multiplier.

Hardwired twiddle factors (in $\mathcal{C}_{t,w_q}$, where $w_q = t + 3L + 2$). For fixed $m = 2^s$, $1 \leq s \leq L$, and fixed address $a = \sum_{r<s} 2^r a_r$, precompute the constant

$$q_m(a) := \mathsf{Prmt}_t\left(\prod_{r<s}^{\text{left}} z_{s-r}^{a_r}\right). \tag{46}$$

Set $q_1(0) := \mathsf{Prmt}_t(1)$ for the size-one base case. The constant $q_m(a)$ approximates the complex coefficient

$$\omega_m^a = \prod_{r<s} \omega_m^{2^r a_r} = e^{2\pi \mathrm{i} a/m}.$$

Here the address bit $a_r = 1$ selects the factor z_{s-r}, and $a_r = 0$ selects $z_{s-r}^0 := \mathsf{Prmt}_p(1)$, the representation of 1 at scale p rather than the scale-zero integer 1, so that each of the s factors carries scale p. Every resulting value belongs to $\mathcal{C}_t$ at the common component width w_q, using only sign extension after the full product. Indeed, each factor—a root, or the stand-in $\mathsf{Prmt}_p(1)$—has component width $p+2$. Complex multiplication adds the widths of its two inputs and one further bit (Appendix A.3), so the s factors and the $s-1$ multiplications between them give a full product of scale $sp \leq t$ and width $s(p+2) + (s-1) = sp + 3s - 1$, whatever the bracketing. The final promotion shifts its numerator left by $t - sp$ bits, leaving $t + 3s - 1 \leq t + 3L - 1 < w_q$ bits. No fractional or high bits are discarded. Here the address bits are hardwired parameters of the table entry. Interpret addresses modulo m.

The twiddle factors are the coefficients used in combining the even and odd child outputs in the FFT's defining recursion (Appendix A.5.1). The table of these coefficients is called the *twiddle table*.

Remark A.1 (Hardwired twiddles versus computed twiddles)**.** Every twiddle factor in the multiplier is a hardwired constant: the circuit generator evaluates the recipe above once for each size m and each output address a, and wires the resulting bits in. The later EF proofs, however, cannot work with the table alone: to apply the index-generalization paradigm of Section 3, we will introduce an auxiliary circuit in Appendix E.1 that computes these twiddle factors by the defining equation (46), and prove that the constants it generates are equal to the hardwired ones (Lemma E.1). We stress that this auxiliary circuit appears only in the EF derivations and is not part of the FFT multiplier, so its gates and its certificates are charged to the EF proofs.

The structure of the twiddle computation circuit lets us argue at a *variable* address: it builds a twiddle factor from the bits of its address, so the identities these factors obey can be derived once, at a variable address (Lemma E.3), instead of one table entry at a time. The superscript left in (46) fixes the order of multiplication for this purpose: the factors are taken in order of increasing r, left-associated into a running product. That order matters only in this auxiliary circuit and the EF derivations. The hardwired constants in the FFT multiplier are exact products, whose values do not depend on that order.

A.5 The Binary Integer Multiplier

Fix positive input lengths r, s and put $\ell = r + s + L + 2$, requiring $\ell < N$. The circuit

$$\mathsf{FFTMul}_{r,s} : \{0,1\}^r \times \{0,1\}^s \longrightarrow \{0,1\}^\ell$$

takes unsigned binary inputs $x = (x_0, \ldots, x_{r-1})$ and $y = (y_0, \ldots, y_{s-1})$, with bits indexed from the least significant position. Its internal gates and output wires are defined by the following composition. The parameters N, L, p, t, the stored constants, and the word widths are those already specified; they are hardwired.

Input embedding. Form length-N complex arrays $\boldsymbol{a}, \boldsymbol{b}$ at fractional scale zero. Set $a_j := x_j + \mathrm{i}0$ for $j < r$ and $a_j := 0 + \mathrm{i}0$ for $r \le j < N$; set $b_j := y_j + \mathrm{i}0$ for $j < s$ and $b_j := 0 + \mathrm{i}0$ for $s \le j < N$. Each bit is zero-extended to the allocated real numerator word, and every imaginary numerator is hardwired to zero. Thus each input bit is one base-two coefficient.

Multiplier overview. Using the component circuits defined in the following subsections, define

$$\mathsf{FFTConv}^{\mathrm{raw}}(\boldsymbol{a}, \boldsymbol{b}) := \mathsf{IFFT}\big(\mathsf{FFT}(\boldsymbol{a}) \odot \mathsf{FFT}(\boldsymbol{b})\big),$$

$$\mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{a}, \boldsymbol{b})_j := \mathsf{Round}\big(\mathrm{Re}\,\mathsf{FFTConv}^{\mathrm{raw}}(\boldsymbol{a}, \boldsymbol{b})_j\big).$$

The raw convolution applies a forward FFT to each input array, multiplies the transformed arrays pointwise, and applies the inverse FFT. The rounded convolution rounds each real output component to an integer and discards its imaginary component. Finally, the integer multiplier applies Carry to the rounded coefficients and returns the first ℓ bits:

$$\mathsf{FFTMul}_{r,s}(x, y) := \big(\mathsf{Carry}(\mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{a}, \boldsymbol{b}))_j\big)_{0 \le j < \ell}.$$

Here $\boldsymbol{a}, \boldsymbol{b}$ are the scale-zero embeddings of x, y defined above, and $\ell = r + s + L + 2$ is the declared output length. The rounding circuit is defined in Appendix A.2; the carry circuit is defined in Appendix A.1 and applied here at length $m = N$, with the common rounded-coefficient width as specified in Appendix A.2. The terminal carry and the carry-output bits at indices $j \ge \ell$ remain available in the joint context but are not returned. The proof of Lemma I.4 shows that these bits and the terminal carry are zero. This slightly oversized output retains leading zeros and avoids a separate tight range proof. Every occurrence of FFTMul denotes this concrete composition, with its declared input lengths, even when subscripts are suppressed. Corresponding output strings are zero-padded to a common length before comparison.

Signatures, scales, and widths. Write $\mathcal{C}_{f,w}$ for scale f and component numerator width w. These are independent parameters. Recall the twiddle width $w_q = t + 3L + 2$, and put $\Delta w = w_q + 3$. For inputs of common component width w, each butterfly raises the scale by t and sign-extends its full outputs to width $w + \Delta w$; the circuit-size accounting below derives this bound. On length-N complex arrays, the component circuits have signatures

$$\mathsf{FFT} : \mathcal{C}^N_{f,w} \longrightarrow \mathcal{C}^N_{f+Lt,\, w+L\Delta w},$$
$$\mathsf{IFFT} : \mathcal{C}^N_{f,w} \longrightarrow \mathcal{C}^N_{f+Lt+L,\, w+L\Delta w},$$
$$\odot : \mathcal{C}^N_{f,w_1} \times \mathcal{C}^N_{g,w_2} \longrightarrow \mathcal{C}^N_{f+g,\, w_1+w_2+1}.$$

Consequently the raw composition has signature

$$\mathsf{FFTConv}^{\mathrm{raw}} : \mathcal{C}^N_{f,w_1} \times \mathcal{C}^N_{g,w_2} \longrightarrow \mathcal{C}^N_{f+g+3Lt+L,\, w_1+w_2+3L\Delta w+1}.$$

For the embedded scale-zero inputs, each coefficient has two signed bits per component. The scale and width schedule is

$$\boldsymbol{a}, \boldsymbol{b} \in \mathcal{C}^N_{0,2}, \qquad \mathsf{FFT}(\boldsymbol{a}), \mathsf{FFT}(\boldsymbol{b}) \in \mathcal{C}^N_{Lt,\, 2+L\Delta w},$$
$$\mathsf{FFT}(\boldsymbol{a}) \odot \mathsf{FFT}(\boldsymbol{b}) \in \mathcal{C}^N_{2Lt,\, 5+2L\Delta w},$$
$$\mathsf{FFTConv}^{\mathrm{raw}}(\boldsymbol{a}, \boldsymbol{b}) \in \mathcal{C}^N_{f_*, w_*}, \qquad f_* = 3Lt + L, \quad w_* = 5 + 3L\Delta w,$$
$$\mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{a}, \boldsymbol{b}) \in \mathcal{D}^N_{0, w_{\mathrm{round}}}, \qquad w_{\mathrm{round}} = w_* + 1 - f_*.$$

Since $w_* > f_*$, the rounding rule of Appendix A.2 gives the displayed width w_{round}. The carry chain then uses w_{round}-bit carries and $(w_{\mathrm{round}} + 1)$-bit sums. In these two forward transforms, butterfly level i outputs entries in $\mathcal{C}_{it,\, 2+i\Delta w}$ for $1 \le i \le L$. In the inverse transform on the pointwise product, reversed level j outputs entries in $\mathcal{C}_{(2L+j)t,\, 5+(2L+j)\Delta w}$ for $1 \le j \le L$, before the final division by N increases the scale by L and leaves the width unchanged. Widths are determined by the input widths and the operator rules, independently of the scales; shorter entries are sign-extended when a common array width is required. The butterfly-level index and the representation scale are therefore distinct; routing and selecting entries preserve their scale. The recursive definitions below specify the scaled operators at each level. Every internal operation retains its exact result on all inputs. Approximation enters through the hardwired constants and the final rounding; width growth is accounted for below.

A.5.1 The Forward FFT

For an input array $\boldsymbol{v} \in \mathcal{C}^m_{f_0}$, write FFT_m for the forward network on its m complex entries. Set $\mathsf{FFT}_1(\boldsymbol{v}) := \boldsymbol{v}$. For $m \ge 2$, let $v^{\mathrm{e}}_j := v_{2j}$ and $v^{\mathrm{o}}_j := v_{2j+1}$ for $j < m/2$, and first compute

$$\boldsymbol{E} := \mathsf{FFT}_{m/2}(\boldsymbol{v}^{\mathrm{e}}), \qquad \boldsymbol{O} := \mathsf{FFT}_{m/2}(\boldsymbol{v}^{\mathrm{o}}).$$

These two recursive size-$m/2$ FFT calls are the children of the size-m call; $\boldsymbol{E}$ and $\boldsymbol{O}$ are their output arrays. Splitting $\boldsymbol{v}$ into its even- and odd-indexed subarrays is wiring, so both recursive calls receive arrays in $\mathcal{C}^{m/2}_{f_0}$. For $m = 2^s$, each recursive call has $s - 1$ butterfly levels, and its output entries therefore belong to $\mathcal{C}_f$, where $f = f_0 + (s-1)t$. Put $k = f + t$. The hardwired twiddle belongs to $\mathcal{C}_t$. For each fixed output position $a < m$, put $b := a \bmod (m/2)$ and define

$$(\mathsf{FFT}_m(\boldsymbol{v}))_a := \mathsf{Add}_k\big(\mathsf{Prmt}_k(E_b), \mathsf{Mul}_{t,f}(q_m(a), O_b)\big).$$

All operators are the complex versions from Appendix A.3. The product has scale $f + t = k$, so both inputs to the adder have scale k. The index b specifies fixed wiring, and $q_m(a)$ is the hardwired table entry for this

output; no address arithmetic or selector is added to the FFT circuit. The outputs at b and $b + m/2$ form one butterfly pair. They share E_b, O_b and the promotion of E_b, but use separate products with $q_m(b)$ and $q_m(b+m/2)$. Set $\mathsf{FFT} := \mathsf{FFT}_N$. The operators are the signed integer circuits and promotion wiring already defined in Appendices A.2 to A.3.

Each forward butterfly uses the full-result complex operators of Appendix A.3. Both hardwired twiddles have scale t; both outputs have scale $f + t = f_0 + st$. Only sign extension is used to give the two outputs their common width, specified in the circuit-size accounting below. No numerator bits are removed.

A.5.2 Pointwise Product

For input arrays $\boldsymbol{u} \in \mathcal{C}^N_{f_0}$ and $\boldsymbol{v} \in \mathcal{C}^N_{g_0}$, the forward outputs satisfy $\mathsf{FFT}(\boldsymbol{u}) \in \mathcal{C}^N_{f_0+Lt}$ and $\mathsf{FFT}(\boldsymbol{v}) \in \mathcal{C}^N_{g_0+Lt}$. Write $\odot$ for pointwise multiplication of complex arrays: $(\boldsymbol{a} \odot \boldsymbol{b})_j := a_j b_j$. Each pointwise product $(\mathsf{FFT}(\boldsymbol{u}))_j (\mathsf{FFT}(\boldsymbol{v}))_j$ uses the complex $\mathsf{Mul}_{f_0+Lt, g_0+Lt}$ of Appendix A.3 and belongs to $\mathcal{C}_{f_0+g_0+2Lt}$; the resulting array belongs to $\mathcal{C}^N_{f_0+g_0+2Lt}$.

A.5.3 The Inverse FFT

For an input array $\boldsymbol{v} \in \mathcal{C}^m_f$, define the unnormalized inverse network $\mathsf{IFFT}^{\mathrm{raw}}_m$ recursively. The superscript records that division by the transform length is postponed until after all recursive calls. Set $\mathsf{IFFT}^{\mathrm{raw}}_1(\boldsymbol{v}) := \boldsymbol{v}$. For $m \geq 2$, first form two length-$m/2$ arrays $\boldsymbol{s}, \boldsymbol{r}$ by applying a reversed butterfly to each pair $v_a, v_{a+m/2}$, for $a < m/2$:

$$s_a := \mathsf{Prmt}_{f+t}\big(\mathsf{Add}_f(v_a, v_{a+m/2})\big),$$

$$r_a := \mathsf{Mul}_{t,f}\big(\overline{q_m(a)}, \mathsf{Sub}_f(v_a, v_{a+m/2})\big).$$

Here $\overline{q_m(a)}$ is the hardwired complex conjugate of the lower-half forward coefficient, also at scale t. Both arrays belong to $\mathcal{C}^{m/2}_{f+t}$. Recursively compute

$$\boldsymbol{E} := \mathsf{IFFT}^{\mathrm{raw}}_{m/2}(\boldsymbol{s}), \qquad \boldsymbol{O} := \mathsf{IFFT}^{\mathrm{raw}}_{m/2}(\boldsymbol{r}).$$

The output is obtained by interleaving these two recursive outputs:

$$\mathsf{IFFT}^{\mathrm{raw}}_m(\boldsymbol{v}) := (E_0, O_0, E_1, O_1, \ldots, E_{m/2-1}, O_{m/2-1}).$$

This interleaving is implemented by wiring. For $m = 2^\ell$, the unnormalized output belongs to $\mathcal{C}^m_{f+\ell t}$. In contrast to the forward recursion, the reversed butterflies precede the two recursive calls. Define the implemented length-N inverse transform by

$$\mathsf{IFFT}(\boldsymbol{v}) := N^{-1} \mathsf{IFFT}^{\mathrm{raw}}_N(\boldsymbol{v}).$$

The final division is the exact scale change described below; the recursive calls perform no normalization. The reversed butterfly likewise uses the full-result complex operators. Both s_a, r_a have scale $f + t$, and their component widths are sign-extended to the common butterfly width specified below. For an inverse input at scale f, reversed level j has scale $f + jt$. The final division by $N = 2^L$ leaves the numerator and its width unchanged and increases the scale to $f + Lt + L$. This exact scale change uses no integer quotient or rounding gate.

A.5.4 Circuit Size Accounting

Recall the twiddle width $w_q = t + 3L + 2$ and the butterfly width increment $\Delta w = w_q + 3$ from the overview. Conjugating a twiddle uses at most $w_q + 1$ bits. If the two butterfly inputs have common component width w, a forward butterfly produces widths at most $w + w_q + 2$: its product has width $w + w_q + 1$, its promoted other input has width $w + t$, and the final sum adds one bit. Each forward pair uses two complex multipliers and two adders, sharing the promotion of its even-child input. This is still a constant number of short-word operations per pair. A reversed butterfly produces widths at most $w + w_q + 3$. Sign-extend both outputs to the common width $w + \Delta w$. This padding is the only adjustment at a butterfly boundary.

A size-2^s call therefore has the exact scale and width signatures

$$\mathsf{FFT}_{2^s} : \mathcal{C}^{2^s}_{f,w} \longrightarrow \mathcal{C}^{2^s}_{f+st,w+s\Delta w}, \qquad \mathsf{IFFT}^{\mathrm{raw}}_{2^s} : \mathcal{C}^{2^s}_{f,w} \longrightarrow \mathcal{C}^{2^s}_{f+st,w+s\Delta w}.$$

Normalization changes the inverse scale to $f + st + s$ without increasing the width. A pointwise product of component widths w_1, w_2 has width $w_1 + w_2 + 1$. Thus the raw convolution has output scale $f + g + 3Lt + L$ and component width

$$w_1 + w_2 + 3L\Delta w + 1.$$

For the two-bit coefficient embeddings the final component width is $w_* = 5 + 3L\Delta w = O(Lt) = O(L^4)$, as in the overview. Rounding at scale $f_* = 3Lt + L$ uses a $(w_* + 1)$-bit sum and returns $w_{\mathrm{round}} = w_* + 1 - f_*$ bits. Since $\Delta w = t + 3L + 5$, this rounded width is $O(L^2)$. The carry chain uses w_{round}-bit carries and $(w_{\mathrm{round}} + 1)$-bit sums, so it adds no width growth with the number of coefficients. A fixed number of network compositions also has $O(L^4)$ widths: each transform adds $L\Delta w$ and each pointwise product adds its two input widths plus one.

Let $W = O(L^4)$ be a common upper bound on the implementation's numerator widths and scales. The proofs below also bound the widths and scales of their auxiliary arithmetic circuits by $O(W)$ when those circuits are introduced. These bounds follow from circuit structure, independently of numerical range assertions. The long input and output integers remain arrays of $O(n)$ bits.

The parameter N is fixed by the original n for the entire joint computation, including nested multiplications. For $n \geq 1$, $L = 4 + \lceil \log(n + 1) \rceil \leq n + 4$, so

$$3n + 2L + 7 \leq 5n + 15 < 16(n + 1) \leq N.$$

This also supplies the carry and zero-padding margins used below. With three n-bit inputs, a nested product has declared length $3n + 2L + 4 < N$, and a sum of two products has length $2n + L + 3 < N$. Hence the circuits compute ordinary integer expressions without modular wraparound. Both networks have $O(NL)$ butterflies, every short arithmetic circuit has polynomial size in W, and carry resolution costs $O(NW)$. Each butterfly has a constant number of signed integer additions and multiplications on $O(W)$ bits; the scale-alignment shifts are wiring. The circuit sizes are $N \log^{O(1)} N$.

A.6 EF Statements of Commutativity, Associativity, and Distributivity for Integer Arithmetic

We use the Boolean circuits, constant-size EF rules, expanded contexts, and designated assertions of Section 2. A proof of equivalence supplies the joint gate definitions of the two circuits and derives the equality checks on their corresponding output wires. This is the circuit-equivalence convention of Ma, Dai, and Shi [MDS25, Section 2.2].

Theorem A.2 (Main theorem). *Fix $n \geq 1$ and let x, y, z denote unsigned n-bit words available in Γ. The three identities involving the multiplier are derived in quasi-linear size,*

$$\Gamma \ \rhd_{\mathsf{EF}}^{n \log^{O(1)}(n+2)}$$
$$\mathsf{FFTMul}(x, \mathsf{ADD}(y, z)) = \mathsf{ADD}(\mathsf{FFTMul}(x, y), \mathsf{FFTMul}(x, z)),$$
$$\mathsf{FFTMul}(\mathsf{FFTMul}(x, y), z) = \mathsf{FFTMul}(x, \mathsf{FFTMul}(y, z)),$$
$$\mathsf{FFTMul}(x, y) = \mathsf{FFTMul}(y, x),$$

and the two identities involving only the adder are derived in linear size,

$$\Gamma \ \rhd_{\mathsf{EF}}^{O(n)}$$
$$\mathsf{ADD}(\mathsf{ADD}(x, y), z) = \mathsf{ADD}(x, \mathsf{ADD}(y, z)),$$
$$\mathsf{ADD}(x, y) = \mathsf{ADD}(y, x).$$

In both parts of Theorem A.2, the required implementation and equality-check definitions have size of the same order as the stated bound, so the additional and complete proof bounds agree.

The rest of the paper proves Theorem A.2 by showing, for each fixed input length n, how to construct an EF derivation of the stated identities. Starting with x, y, z as free input words, the circuits encoding the three multiplier assertions, including the arithmetic circuits and their output-equality checks, have size $n \log^{O(1)}(n+2)$, and Theorem A.2 bounds the additional EF derivation establishing them by $n \log^{O(1)}(n+2)$ as well, including any auxiliary circuit definitions introduced in the proof. Thus the total size of these assertion encodings and their EF derivation is $n \log^{O(1)}(n+2)$. The two adder assertions are proved column by column, introducing no circuit beyond the two adders and the equality check of the conclusion itself, so for them both bounds are $O(n)$: the derivation matches the size of the schoolbook adder up to a constant factor.

B Reusable Big-Step Inference Rules

This section proves the basic inference rules stated in Section 3.1, in the general forms that the rest of the paper uses. It also proves two further rules used throughout the paper, folding and proof by enumeration, and adds inference rules for conditional forms and for selection at a fixed address. Throughout, we use the conventions of Section 3.1 for balanced AND and OR trees and the array selector Sel, and we define the equality check EQ below. At the end of the section, we state index-generalization in the general form that our applications use; its proof is the one given in Section 3.3.

Conditional forms. Rules are often applied inside a derivation that is itself carried out under a hypothesis. The following rule supplies the conditional form of every other rule in the paper, at the same cost up to a constant factor.

Lemma B.1 (Conditional forms). *Let H be a named Boolean wire in Γ. For checks $P_0, \ldots, P_{k-1}$ and $Q_0, \ldots, Q_{l-1}$,*

$$\Gamma : [P_j]_{j<k} \ \rhd_{\mathsf{EF}}^{S} \ [Q_i]_{i<l} \qquad \textit{implies} \qquad \Gamma : [H \to P_j]_{j<k} \ \rhd_{\mathsf{EF}}^{O(S)} \ [H \to Q_i]_{i<l}.$$

Proof. Place $H \to$ in front of every assertion of the given derivation. Each small-step inference has a fixed conditional counterpart, derivable from it in a constant number of steps, so each original step becomes a constant number of steps. Each new assertion is one implication gate over H and the check of the original assertion, and each missing implication check has constant size over the named wires. The gate definitions in Γ, including every EF extension definition introduced by the derivation, are not relativized and remain unconditional. The derivation therefore grows by a constant factor, so its cost is $O(S)$. □

A rule that is available as an implication rather than as a derivation needs no relativization: if $\Gamma \rhd_{\mathsf{EF}}^{S} P \to Q$, then one instance of $(P \to Q) \to \big((H \to P) \to (H \to Q)\big)$ and two applications of modus ponens give $\Gamma : H \to P \rhd_{\mathsf{EF}}^{S+O(|P|+|Q|)} H \to Q$. In the rules below we display the conditional form whenever later sections cite it by number; each such form is an instance of Lemma B.1.

Folding and unfolding. The following two lemmas give folding and unfolding in general form. Lemma 3.1 is the OR-tree form of Lemma B.3 that returns all leaves.

Lemma B.2 (Folding Boolean assertions)**.** *Let H be any named Boolean wire in Γ for the conditional statements below.*

AND tree. *The unconditional and conditional forms are*

$$\Gamma : [P_j]_{j<m} \;\rhd_{\mathsf{EF}}^{O(m)}\; \bigwedge_{j<m} P_j, \tag{47}$$

$$\Gamma : [H \to P_j]_{j<m} \;\rhd_{\mathsf{EF}}^{O(m)}\; H \to \bigwedge_{j<m} P_j.$$

OR tree. *The corresponding forms are*

$$\Gamma : [\neg P_j]_{j<m} \;\rhd_{\mathsf{EF}}^{O(m)}\; \neg\left(\bigvee_{j<m} P_j\right),$$

$$\Gamma : [H \to \neg P_j]_{j<m} \;\rhd_{\mathsf{EF}}^{O(m)}\; H \to \neg\left(\bigvee_{j<m} P_j\right).$$

In each big-step inference rule, the displayed premises and their check circuits are supplied in Γ. In the conditional forms, H need not be asserted true. All other intermediate checks are charged. If the tree must be constructed within an enclosing proof, its $m-1$ gates and any missing negation checks add $O(m)$ to that proof's cost.

Proof. Process internal nodes from the leaves to the root. For an AND gate, the two child assertions $c_\ell = 1, c_r = 1$ give $c_v = 1$ by a fixed propositional derivation from its gate definition. For an OR gate, $c_\ell = 0, c_r = 0$ give $c_v = 0$ by the corresponding fixed derivation. Construct any missing negation checks and retain each established parent assertion. There are $m-1$ nodes, each costing a constant number of gates and small-step inferences. The root assertion is the desired conclusion. The conditional versions are instances of Lemma B.1. □

Lemma B.3 (Unfolding a Boolean tree)**.** *Let Γ contain the specified tree's gate definitions. In each pair of big-step inference rules below, the first returns all leaf assertions and the second returns the assertion at one fixed leaf address $j < m$. Let H be any named Boolean wire in Γ for the conditional statements.*

AND tree. *From the assertion that the AND-tree output is true,*

$$\Gamma : \bigwedge_{j<m} P_j \;\rhd_{\mathsf{EF}}^{O(m)}\; [P_j]_{j<m}, \tag{48}$$

$$\Gamma : \bigwedge_{j<m} P_j \;\rhd_{\mathsf{EF}}^{O(\log m)}\; P_j. \tag{49}$$

The conditional forms are

$$\Gamma : H \to \bigwedge_{j<m} P_j \;\rhd_{\mathsf{EF}}^{O(m)}\; [H \to P_j]_{j<m},$$

$$\Gamma : H \to \bigwedge_{j<m} P_j \;\rhd_{\mathsf{EF}}^{O(\log m)}\; H \to P_j.$$

OR tree. *From the assertion that the OR-tree output is false,*

$$\Gamma : \neg\left(\bigvee_{j<m} P_j\right) \;\rhd_{\mathsf{EF}}^{O(m)}\; [\neg P_j]_{j<m},$$

$$\Gamma : \neg\left(\bigvee_{j<m} P_j\right) \;\rhd_{\mathsf{EF}}^{O(\log m)}\; \neg P_j.$$

The conditional forms are

$$\Gamma : H \to \neg\left(\bigvee_{j<m} P_j\right) \;\rhd_{\mathsf{EF}}^{O(m)}\; [H \to \neg P_j]_{j<m},$$

$$\Gamma : H \to \neg\left(\bigvee_{j<m} P_j\right) \;\rhd_{\mathsf{EF}}^{O(\log m)}\; H \to \neg P_j.$$

Proof. At an AND gate, $c_v = 1$ and its defining equivalence give both $c_\ell = 1$ and $c_r = 1$ by constant-size inference rules. At an OR gate, $c_v = 0$ gives both $c_\ell = 0$ and $c_r = 0$. Starting from the root, visit every internal node once and retain both child assertions; this obtains the whole list in $O(m)$ additional records, including any missing negation gates. For one fixed leaf, visit only its root-to-leaf path and derive the required child at each split, at cost $O(\log m)$ because the tree has height $O(\log m)$. The conditional versions are instances of Lemma B.1, with the same traversal costs. Retaining all the unfolded assertions permits subsequent references to them without repeating the traversal. □

Equality checks. For two w-bit strings x, y, with $w \geq 1$, define

$$\mathsf{EQ}(x, y) := \bigwedge_{j<w} (x_j \leftrightarrow y_j).$$

Each bitwise equivalence is a constant-size Boolean circuit, and the conjunction is a balanced binary AND tree. The check has $O(w)$ gates and compares the strings themselves, including when they encode addresses.

Equality propagation and substitution. The following lemma gives equality propagation in general form. Lemma 3.2 is the single-output case of its bitwise conditional form (54).

Lemma B.4 (Equality propagation and substitution)**.** *Let x, y be w-bit strings and let C be a Boolean circuit with $r \geq 1$ output bits and $|C|$ gates. The copies $C(x, z)$ and $C(y, z)$ have the same gate structure and share all other input wires z. Then*

$$\Gamma : \mathsf{EQ}(x, y) \;\rhd_{\mathsf{EF}}^{O(w+|C|+r)}\; \mathsf{EQ}(C(x, z), C(y, z)). \tag{50}$$

For a Boolean wire H, the conditional version is

$$\Gamma : H \to \mathsf{EQ}(x, y) \;\rhd_{\mathsf{EF}}^{O(w+|C|+r)}\; H \to \mathsf{EQ}(C(x, z), C(y, z)). \tag{51}$$

In particular, for a single-output Boolean circuit P,

$$\Gamma : \mathsf{EQ}(x, y),\ P(x, z) \;\rhd_{\mathsf{EF}}^{O(w+|P|)}\; P(y, z), \tag{52}$$

$$\Gamma : H \to \mathsf{EQ}(x, y),\ H \to P(x, z) \;\rhd_{\mathsf{EF}}^{O(w+|P|)}\; H \to P(y, z). \tag{53}$$

The input equality may also be given bit by bit:

$$\Gamma : \big[H \to (x_t \leftrightarrow y_t)\big]_{t<w} \;\rhd_{\mathsf{EF}}^{O(|C|+r)}\; H \to \mathsf{EQ}(C(x, z), C(y, z)), \tag{54}$$

and likewise without the antecedent H. The sizes $|C|, |P|$ count only the gates through which equality is propagated, and exclude the circuits producing the input wires x, y, z. Finally, the check EQ is symmetric and transitive: for w-bit strings x, y, u,

$$\Gamma : \mathsf{EQ}(x, y) \;\rhd_{\mathsf{EF}}^{O(w)}\; \mathsf{EQ}(y, x), \qquad \Gamma : \mathsf{EQ}(x, y),\ \mathsf{EQ}(y, u) \;\rhd_{\mathsf{EF}}^{O(w)}\; \mathsf{EQ}(x, u), \tag{55}$$

and likewise in the conditional form, with a common antecedent H on every premise and on the conclusion. Chaining n word equalities applies the second rule $n-1$ times, at cost $O(nw)$, and reuses the given equalities without copying their derivations.

Proof. Unfold the input equality by Lemma B.3, at cost $O(w)$. In the bitwise form (54), the bit equalities are given, and this step is skipped. Traverse the corresponding gates of the two copies in topological order. At each gate, a fixed propositional inference rule derives equality of the two output bits from equality of the corresponding input bits and the two gate definitions. Shared input wires and constants use reflexive bit equalities. Each gate has bounded fan-in, so its local derivation and any missing intermediate equality check cost $O(1)$. Retain each derived equality once and reuse it at every outgoing wire. All gates therefore cost $O(|C|)$, even with unbounded fanout. Apply Lemma B.2 to the r output-bit equalities, at cost $O(r)$.

For $C = P$, the final output equality and $P(x, z)$ give $P(y, z)$ by a constant-size propositional inference rule. The conditional forms (51) and (53) are instances of Lemma B.1, as is (54) relative to its unconditional variant. For (55), unfold the premises by Lemma B.3, apply at each of the w bit positions the constant-size propositional derivation of symmetry, respectively transitivity, of an equivalence, and fold the resulting bit equalities by Lemma B.2; the conditional forms are again instances of Lemma B.1. □

Proof by enumeration. Proof by enumeration works by enumerating all possible values of some word a (henceforth called address), and then combining all the assertions into one at the end, as stated in the following lemma.

Lemma B.5 (Proof by enumeration)**.** *Let $m = 2^s$, with $s \geq 1$, let a be an s-bit address, and let c be a Boolean output. Write $[a = j]$ for the equality check between a and the hardwired s-bit encoding of j. Then*

$$\Gamma \ \rhd_{\mathsf{EF}}^{O(m \log(m+1))} \ \bigvee_{j<m} [a = j], \tag{56}$$

and

$$\Gamma : \big([a = j] \to c\big)_{j<m} \ \rhd_{\mathsf{EF}}^{O(m \log(m+1))} \ c. \tag{57}$$

Proof. Construct a binary tree of address-prefix checks, starting from the constant one at the root. At a node with prefix-check output p and next address bit b, introduce the two child outputs

$$p_0 := p \wedge \neg b, \qquad p_1 := p \wedge b.$$

A constant-size propositional derivation from their gate definitions proves

$$p_0 \vee p_1 \leftrightarrow p.$$

Each leaf checks one complete address. For each fixed $j < m$, the conditional unfolding and folding statements of Lemmas B.2 and B.3, together with the constant-bit equivalences, identify its leaf output with $[a = j]$ in $O(s)$ additional gates and small-step inferences. Use an OR tree with the same shape and binary address order. Proceeding from the leaves to the root, the displayed identity identifies each OR-subtree output with its corresponding prefix-check output. The root prefix check is one, so this proves $\bigvee_{j<m}[a = j]$. The prefix circuit has $O(m)$ gates. Constructing any missing address checks, OR-tree gates, and equivalence checks, and deriving the leaf and subtree equivalences, costs $O(ms + m)$ in total.

For the second statement, use the same OR tree. At an internal node with child outputs u, v, a constant-size propositional inference rule derives

$$(u \vee v) \to c \quad \text{from} \quad u \to c, \ v \to c.$$

Starting from the supplied address cases and proceeding to the root gives $(\bigvee_{j<m}[a = j]) \to c$. The first statement and modus ponens give c. Constructing the OR tree if needed, introducing the internal implication checks, and combining the cases costs $O(m)$. Every implication refers to the same output wire c, whose circuit is shared. Together with the first derivation, the total cost is $O(m \log(m + 1))$. □

Array selectors. Recall the selector $\mathsf{Sel}_{m,w}(A, i)$ of Section 3.1, a balanced binary tree of word multiplexers with s levels and $O(mw)$ gates, which selects the entry A_i of an array of $m = 2^s$ words of width w. Selector-output equalities use the binary-string check EQ.

Lemma B.6 (Selector at a fixed address)**.** *For an array $\boldsymbol{A}$ of $m = 2^s$ words of width $w \geq 1$, an s-bit address i, and every fixed $j < m$,*

$$\Gamma \ \rhd_{\mathsf{EF}}^{O(w \log(m+1))} \ [i = j] \Longrightarrow \mathsf{Sel}_{m,w}(A, i) = A_j.$$

Here $[i = j]$ is the equality check against the hardwired address j; for $m = 1$, it is the constant one. For $m = 2$ the selector is a single word multiplexer, $\mathsf{Sel}_{2,w}\big((y, x), c\big) = \mathsf{Mux}(c, x, y)$, and the two cases read

$$\Gamma \ \rhd_{\mathsf{EF}}^{O(w)} \ c \Longrightarrow \mathsf{Mux}(c, x, y) = x, \qquad \neg c \Longrightarrow \mathsf{Mux}(c, x, y) = y, \tag{58}$$

with the control wire itself as the antecedent.

Proof. For $s \geq 1$, apply conditional unfolding from Lemma B.3 to $[i = j]$ to obtain all s address-bit values under this antecedent, at cost $O(s)$. At each node on the path to leaf j, its control-bit value and the multiplexer definitions imply equality between each output bit and the corresponding bit of the selected child. These are constant-size propositional derivations, retained under the antecedent $[i = j]$. Compose the bit equalities along the s levels, then use conditional folding from Lemma B.2 to obtain the word-equality check at the root. The local derivations and their missing equality and implication gates cost $O(w(s + 1))$ in total. For $m = 1$, fold the reflexive bit equalities of A_0 at cost $O(w)$. The array entries and the existing selector gates are shared throughout. For $m = 2$ there is one level and one address bit, and the one-bit checks $[c = 1]$ and $[c = 0]$ are equivalent to the wires c and $\neg c$ by constant-size derivations; replacing the antecedents accordingly gives (58) at cost $O(w)$. Conversely, these equivalences put an assertion made under c or $\neg c$ back into the form $[c = j] \to \cdot$ that Lemma B.5 consumes. □

Selection commutes with entrywise circuits. The following lemma states in general form that selecting an entry and then applying a circuit gives the same outcome as applying the circuit to every entry and then selecting. Lemma 3.3 is its special case of a single-output check on one array.

Lemma B.7 (Selection commutes with entrywise circuits)**.** *Let $\boldsymbol{A}^{(1)}, \ldots, \boldsymbol{A}^{(c)}$ be a fixed number c of arrays, each consisting of $m = 2^s$ words of width at most w, and let K be an s-bit address. Let F be a Boolean circuit with $r \geq 1$ output bits and $|F|$ gates whose inputs are one word from each array and, optionally, an s-bit index. Besides these inputs, F may read a list z of $|z|$ further wires of Γ, and all copies of F below read the same wires z. For each $j < m$, let F_j be the copy of F applied to the entries $A_j^{(1)}, \ldots, A_j^{(c)}$ and, if F reads an index, to the hardwired s-bit encoding of j. Let F^* be the copy of F applied to the selected words $\mathsf{Sel}(\boldsymbol{A}^{(1)}, K), \ldots, \mathsf{Sel}(\boldsymbol{A}^{(c)}, K)$ and, if F reads an index, to K. Then*

$$\Gamma \;\rhd_{\mathsf{EF}}^{O(m(|F|+w+s+|z|))}\; F^* = \mathsf{Sel}\big((F_j)_{j<m}, K\big),$$

where the right-hand side selects among the r-bit outputs of the copies F_j. The address K may be any wires of Γ. The conclusion circuit consists of the selectors $\mathsf{Sel}(\boldsymbol{A}^{(t)}, K)$, the circuit F^*, the copies F_j, the selector of their outputs, and the equality check. The selectors for the other wires of F are charged.

Proof. Every wire g of F has m copies $g_0, \ldots, g_{m-1}$, one in each copy F_j, and one more copy g^* in F^*. Write $\hat{g} := \mathsf{Sel}((g_j)_{j<m}, K)$ for its selected copy. We derive $g^* \leftrightarrow \hat{g}$ for every wire of F, in topological order. At the output bits, these are the bitwise equivalences between the two sides of the conclusion, and folding (Lemma B.2, (47)) combines them into the stated equality check at cost $O(r)$.

Input bits from an array. For a bit of the word read from $\boldsymbol{A}^{(t)}$, the selected copy $\hat{g}$ is the corresponding bit of $\mathsf{Sel}(\boldsymbol{A}^{(t)}, K)$, since a word selector consists of one-bit selectors with shared address bits. This is the wire g^* itself, so nothing needs to be derived.

Index bits. For bit t of the index, the copies are the hardwired bits $(j_t)_{j<m}$, where j_t denotes bit t of j, and $g^* = K_t$. Build the selector $\hat{g}$ of these constants, with $O(m)$ gates, and derive $\hat{g} \leftrightarrow K_t$ along its tree. At the multiplexers controlled by K_t, the two subtrees carry the constants 0 and 1, so their outputs equal the control bit K_t. Below these multiplexers, both subtrees carry the same constant, and above them, both subtrees carry the value K_t. This costs $O(m)$ for each of the s index bits.

Further input wires. For a wire g among z, all copies are the wire g itself. Build the selector $\hat{g}$ of m copies of g, with $O(m)$ gates, and derive $\hat{g} \leftrightarrow g$ along its tree: each of its multiplexers has the wire g on both data inputs, so a constant-size derivation identifies its output with g. This costs $O(m)$ for each wire of z.

Gates. Let g be a gate that computes a Boolean operation of bounded arity a from the wires $x^1, \ldots, x^a$. Build the one-bit selector $\hat{g}$, with $O(m)$ gates, unless it belongs to the conclusion. For a common control bit c, the identity

$$g\big(\mathsf{Mux}(c, x_1, y_1), \ldots, \mathsf{Mux}(c, x_a, y_a)\big) \leftrightarrow \mathsf{Mux}\big(c, g(x_1, \ldots, x_a), g(y_1, \ldots, y_a)\big)$$

has a constant-size Frege derivation. All selectors at K have the same tree shape and the same control bit at each level. Apply the identity at every node of the selector trees of $g, x^1, \ldots, x^a$, from the leaves to the root; at a leaf j, the required equivalence is the definition of the gate g_j. At the root, this gives $\hat{g} \leftrightarrow g(\hat{x}^1, \ldots, \hat{x}^a)$, at cost $O(m)$. The equivalences $x^{t*} \leftrightarrow \hat{x}^t$ for the inputs of g, which precede g in topological order, and the definition of g^* then give $g^* \leftrightarrow \hat{g}$ by a constant-size derivation.

In total, the new selectors and derivations cost $O(m(|F| + s + |z|))$, and the final folding costs $O(r)$. Every output bit is a gate or an input bit of F, so $r \le |F| + cw + s + |z|$, and the total cost is $O(m(|F| + w + s + |z|))$. □

Index-generalization. Finally, we state index-generalization in the general form that our applications use, in which the check may read several arrays selected at the same index, as well as the index itself and further wires of the base context. For a check on one array that reads nothing else, it is Lemma 3.4, whose proof was given in Section 3.3; the same proof applies to the general form.

Lemma B.8 (Index-generalization, general form)**.** *Let $\boldsymbol{A}^{(1)}, \ldots, \boldsymbol{A}^{(c)}$ be a fixed number c of arrays available in Γ, each consisting of $m = 2^s$ words of width at most w. Let P be a single-output Boolean check of circuit size $|P|$ whose inputs are one word from each array and, optionally, an s-bit index. Besides these inputs, P may read a list z of $|z|$ further wires of Γ, and all copies of P below read the same wires z. For each $j < m$, let P_j be the copy of P applied to the entries $A_j^{(1)}, \ldots, A_j^{(c)}$ and, if P reads an index, to the hardwired s-bit encoding of j. For an s-bit index i, let φ_i be the copy of P applied to the selected words $\mathsf{Sel}(\boldsymbol{A}^{(1)}, i), \ldots, \mathsf{Sel}(\boldsymbol{A}^{(c)}, i)$ and, if P reads an index, to i. If*

$$\Gamma' \ \rhd_{\mathsf{EF}}^{S} \ \varphi_i = 1$$

for every context $\Gamma' \supseteq \Gamma$ and every ordered tuple of s wires of Γ', read as an s-bit index and denoted i, then

$$\Gamma \ \rhd_{\mathsf{EF}}^{S+O(m(|P|+w+s+|z|))} \ [P_j = 1]_{j<m}.$$

Proof. The proof of Lemma 3.4 in Section 3.3 applies with two changes. First, the conclusion circuit $p = \varphi_K$ at the witness now contains a selector at K for each of the c arrays, which cost $O(mw)$ in total. Second, in Step 3, the failure flags $\beta_j = \neg P_j$ are the copies of the check $\neg P$ at the entries at position j, at the hardwired encoding of j, and at the further wires z, so Lemma B.7 gives $\neg p = \mathsf{Sel}((\beta_j)_{j<m}, K) = T$ at cost $O(m(|P|+w+s+|z|))$. The other steps are unchanged, and the total cost is $S+O(m(|P|+w+s+|z|))$. □

C Short-Word Library

C.1 Auxiliary Circuitry

The proofs reuse the arithmetic circuits already defined in Appendices A.1 to A.2. We now specify Boolean checks, their use on scaled values, and the big-step inference rules for the resulting short-word library. For input and working widths $O(W)$, each primitive short-word operation has $W^{O(1)}$ Boolean gates and is generated in time polynomial in W. Circuits on arrays are charged separately for their number of words. The reusable properties asserted below have EF derivations generated in polynomial time and of additional size $W^{O(1)}$. Constructing their conclusion circuits from the supplied operand wires also costs $W^{O(1)}$; the circuits producing those operands are shared.

Equality checks. Extend the binary-string check EQ of Appendix B to numerical inputs according to their declared types.

1. *Integer words.* Zero-extend unsigned words or sign-extend signed words to a common width, then apply the binary-string check. If signed and unsigned words are compared, first represent both at a common signed width with an extra sign bit for the unsigned operand.

2. *Scaled real numbers.* Promote both inputs exactly to the maximum of their scales, then compare their integer numerators by the integer-word check. Widths are aligned by extension only.

3. *Scaled complex numbers.* Apply the scaled-real check separately to the real and imaginary components, and take the AND of the two outputs.

For arrays of any of these types, equality is the AND of the coordinatewise equality checks. Word and vector equalities below abbreviate these circuits. Each scalar check has $O(W)$ gates for the working widths used here; array checks are charged for their number of entries. These checks belong to the EF proofs and add no gates to the FFT/IFFT multiplier.

Order checks. The checks $x < y$ and $x \le y$ use the following cases. The checks $x > y$ and $x \ge y$ reverse the inputs. A binary string must have a specified integer interpretation to be ordered; no order on complex numbers is used.

1. *Unsigned integer words.* For unsigned words $X = (x_j)_{j<w}$ and $Y = (y_j)_{j<w}$, zero-extend to a common width w and subtract X from Y. The borrow bits and difference bits are

$$b_0 := 0, \qquad d_j := y_j \oplus x_j \oplus b_j,$$

$$b_{j+1} := (\neg y_j \wedge (x_j \vee b_j)) \vee (x_j \wedge b_j) \qquad (0 \le j < w).$$

   Retain the final borrow even though only a Boolean comparison is requested. Define

$$\mathsf{LE}_{\mathrm{u}}(X, Y) := \neg b_w, \qquad \mathsf{LT}_{\mathrm{u}}(X, Y) := \neg b_w \wedge \neg \mathsf{EQ}(X, Y).$$

   Thus LE_{u} checks $X \le Y$ and LT_{u} checks $X < Y$. The nonnegative comparator LE used in Appendix C.1 is LE_{u}.

2. *Signed integer words.* For signed words, first sign-extend to a common width, and write their sign bits as s_X, s_Y. Define

$$\mathsf{LT}_{\mathrm{s}}(X, Y) := (s_X \wedge \neg s_Y) \vee \big((s_X \leftrightarrow s_Y) \wedge \mathsf{LT}_{\mathrm{u}}(X, Y)\big),$$

$$\mathsf{LE}_{\mathrm{s}}(X, Y) := \mathsf{LT}_{\mathrm{s}}(X, Y) \vee \mathsf{EQ}(X, Y).$$

   For equal signs, unsigned order on the two's-complement bit patterns agrees with signed order. Nonnegative absolute-value outputs use the unsigned circuits. The finite arithmetic lemma below supplies EF proofs about these specified gates.

3. *Scaled real numbers.* Promote both operands to the maximum of their scales, sign-extend the integer numerators to a common width, and use the signed comparator. Nonnegative numerator comparisons use the unsigned comparator. Mixed signed and unsigned integer comparisons first use a common signed representation as in the equality check. Each scalar order check has $O(W)$ gates. In the signed-comparator formula, LT_{u} compares the underlying bit patterns as unsigned words.

Absolute values and norms. These operations use the full-result arithmetic circuits of Appendices A.1 to A.3, according to the input type.

1. *Integer words.* $\mathsf{Abs}(X)$ is the full-result integer absolute-value circuit; for an unsigned input, its value is already nonnegative.

2. *Scaled real numbers.* For $x = X2^{-f}$, the absolute value has numerator $\mathsf{Abs}(X)$ and scale f. For real inputs, $|x|_1$ means this ordinary absolute value.

3. *Scaled complex numbers.* For $z = (X+\mathrm{i}Y)2^{-f}$, the 1-norm $|z|_1$ has numerator $\mathsf{AddInt}(\mathsf{Abs}(X), \mathsf{Abs}(Y))$ and scale f, where AddInt retains the full integer sum. We also define the squared 2-norm by

$$|z|_2^2 := (\operatorname{Re} z)^2 + (\operatorname{Im} z)^2 = (X^2 + Y^2)2^{-2f}.$$

   This notation directly names the circuit with two full real multipliers and an adder, at output scale $2f$.

Binary strings without a numerical interpretation have no absolute value or norm here.

Norm and approximation checks. For a length-N vector $\boldsymbol{v}$ and a nonnegative scaled real bound e, write

$$\|\boldsymbol{v}\|_\infty \le e \quad \text{for} \quad |v_j|_1 \le e \quad \text{for every } j < N.$$

Here $|v_j|_1$ is ordinary absolute value for real or integer entries and $|\operatorname{Re} v_j| + |\operatorname{Im} v_j|$ for complex entries. For scalar real or complex values x, y, write

$$x \overset{e}{\approx} y \quad \text{for} \quad |x - y|_1 \le e.$$

For vectors $\boldsymbol{u}, \boldsymbol{v}$ of the same length, $\boldsymbol{u} \overset{e}{\approx} \boldsymbol{v}$ denotes $\|\boldsymbol{u} - \boldsymbol{v}\|_\infty \le e$. These notations name Boolean checks, implemented as follows. To implement $\|\boldsymbol{v}\|_\infty \le e$, promote the entries and bound exactly to a common comparison scale f, and write $e = E2^{-f}$. Use the unsigned comparator $\mathsf{LE} = \mathsf{LE}_\mathrm{u}$ on nonnegative magnitudes. The output depends on the entry type:

1. *Scaled real and integer vectors.* Write $v_j = X_j 2^{-f}$. The output is

$$\bigwedge_{j<N} \mathsf{LE}\big(\mathsf{Abs}(X_j), E\big).$$

   Integer entries are scaled real numbers at scale zero, promoted to f if needed for comparison with the bound. This check uses no imaginary component or addition.

2. *Scaled complex vectors.* Write $v_j = (X_j + \mathrm{i}Y_j)2^{-f}$. The output is

$$\bigwedge_{j<N} \mathsf{LE}\big(\mathsf{AddInt}(\mathsf{Abs}(X_j), \mathsf{Abs}(Y_j)), E\big).$$

   The sum of the component absolute values represents the complex 1-norm.

In both cases the conjunction is a binary AND tree and all arithmetic retains its full result. The declared entry type determines which check is used.

To implement $x \overset{e}{\approx} y$ for scalars, first embed any mixed real and complex operands into the complex representation; integer scalars are real values at scale zero. Let $f_0 = \max\{\mathrm{sc}(x), \mathrm{sc}(y), \mathrm{sc}(e)\}$. To specify a fixed comparison scale $f \ge f_0$, write

$$x \overset{e}{\approx}_f y \quad \text{for} \quad |\mathsf{Prmt}_f(x) -_f \mathsf{Prmt}_f(y)|_1 \le \mathsf{Prmt}_f(e).$$

For length-N vectors $\boldsymbol{u}, \boldsymbol{v}$, use the same embedding convention and let f_0 be the maximum scale among e and all entries of $\boldsymbol{u}, \boldsymbol{v}$. For fixed $f \geq f_0$, write

$$\boldsymbol{u} \overset{e}{\approx}_f \boldsymbol{v} \quad \text{for} \quad \|\big(\mathsf{Prmt}_f(u_j) -_f \mathsf{Prmt}_f(v_j)\big)_{j<N}\|_\infty \leq \mathsf{Prmt}_f(e).$$

Both definitions use the norm and comparison circuits above. Omitting the subscript selects the default scale f_0. The subscript promotes both operands and the error bound to f for the check; the tolerance remains e. These auxiliary checks reuse the existing operand outputs, whose prescribed scales are unchanged. The promotion-of-error-check statements of Lemma C.1 prove equivalence with the default-scale check.

The rule for combining approximation checks is also stated in Lemma C.1.

C.2 Short-Word Inference Rules

This subsection gives big-step inference rules for arithmetic on scaled real and complex numbers and their vectors. For each statement below, fix the indicated scales and arbitrary widths for its operand words (available in Γ). The operand widths need not be equal; we omit the width of $\mathcal{D}_f$ and $\mathcal{C}_f$.

Assume that every scale, and input, intermediate, and output numerator width in the specified circuits is $O(W)$, with widths determined by the full-result operator rules.

Lemma C.1 (Short-word inference rules)**.** *Each big-step inference rule below has the displayed additional EF derivation cost.*

All big-step inference rules below incur polynomial additional EF derivation cost, and we do not attempt to optimize the fine-grained complexity of them. We state all big-step inference rules we need for completeness but will omit their EF proofs; this is because given suitable experience on proof complexity and bounded arithmetic (see, e.g., [Kra19, Li25]), the EF proofs of these rules are straightforward (though tedious).

In slightly more detail, we may first prove the corresponding rules in Cook's theory PV (see [Coo75] and also [Li25]), and by applying the propositional translation, we can obtain polynomial-sized EF proofs of the formulas corresponding to the inference rules. The formulas obtained from propositional translation might be different from the displayed big-step inference rules, as functions here (such as $\mathsf{Mul}, \mathsf{Add}$) are implemented in specific ways (described in Appendix A). Nevertheless, assuming straightforward formalization of the functions in PV, the equivalence between the formulas obtained from propositional translation and the big-step inference rules displayed here can be proven in size polynomial in W. In our actual application later, W is polylogarithmically bounded.

Power-of-two wiring and full multipliers. Fix $k \geq 0$. For a signed integer word A, the left shifts of Appendix A.1 satisfy

$$\Gamma \ \rhd_{\mathsf{EF}}^{W^{O(1)}} \ \mathsf{MulInt}(2^k, A) = 2^k A = A2^k = \mathsf{MulInt}(A, 2^k). \tag{59}$$

Unsigned words are zero-extended to signed words for this comparison. For $x \in \mathcal{D}_f$ or $x \in \mathcal{C}_f$, the scaled versions are

$$\Gamma \ \rhd_{\mathsf{EF}}^{W^{O(1)}}$$

$$\mathsf{Mul}_{0,f}(2^k, x) = 2^k x = x2^k = \mathsf{Mul}_{f,0}(x, 2^k), \tag{60a}$$

$$\mathsf{Mul}_{k,f}(2^{-k}, x) = x/2^k = \mathsf{Mul}_{f,k}(x, 2^{-k}). \tag{60b}$$

Here the middle expressions use the specialized wiring, whereas the explicit Mul expressions use full multipliers. Equality compares the outputs after scale alignment and width padding.

Whenever an arithmetic identity below or later is applied to a power-of-two scalar product, use (59) or (60) and Lemma B.4 to transfer the identity to the specified wiring. Each short-word application includes $W^{O(1)}$ additional cost for any missing multiplier and check gates and the transfer derivation; operand circuits are shared. The same convention applies to order and norm bounds, and to scalar instances of the network identities.

Exact division identities. For real or complex x, y and fixed $k \geq 0$, with omitted scales interpreted as in Appendix A.2, we have

$$\Gamma \ \rhd_{\mathsf{EF}}^{W^{O(1)}}$$

$$(2^k x)/2^k = x, \tag{61a}$$

$$(x \pm y)/2^k = x/2^k \pm y/2^k, \tag{61b}$$

$$(xy)/2^k = (x/2^k)y = x(y/2^k). \tag{61c}$$

Addition identities. For $x, y, z \in \mathcal{D}_f$ or $x, y, z \in \mathcal{C}_f$, write $0_f := \mathsf{Prmt}_f(0)$, with 0 initially at scale zero, and let $\mathsf{Neg}_f(x)$ denote full scale-f negation. Then

$$\Gamma \ \rhd_{\mathsf{EF}}^{W^{O(1)}}$$

$$x +_f y = y +_f x, \tag{62a}$$

$$(x +_f y) +_f z = x +_f (y +_f z), \tag{62b}$$

$$x +_f 0_f = x, \tag{62c}$$

$$0_f +_f x = x, \tag{62d}$$

$$x +_f \mathsf{Neg}_f(x) = 0_f. \tag{62e}$$

All additions retain their full numerators, and the equality checks sign-extend outputs to common widths. Negation retains an extra sign bit, so additive inversion holds for every input, including the most negative numerator.

Multiplication identities. For $x \in \mathcal{D}_f$, $y \in \mathcal{D}_g$, and $z \in \mathcal{D}_{g'}$ or $x \in \mathcal{C}_f$, $y \in \mathcal{C}_g$, and $z \in \mathcal{C}_{g'}$, we have

$$\Gamma \ \rhd_{\mathsf{EF}}^{W^{O(1)}}$$

$$\mathsf{Mul}_{f,g}(x, y) = \mathsf{Mul}_{g,f}(y, x), \tag{63a}$$

$$\mathsf{Mul}_{f+g,h}\big(\mathsf{Mul}_{f,g}(x, y), z\big) = \mathsf{Mul}_{f,g+h}\big(x, \mathsf{Mul}_{g,h}(y, z)\big), \tag{63b}$$

$$\mathsf{Mul}_{f,0}(x, 1) = x, \tag{63c}$$

$$\mathsf{Mul}_{0,f}(1, x) = x, \tag{63d}$$

$$\mathsf{Mul}_{f,0}(x, 0) = 0_f, \tag{63e}$$

$$\mathsf{Mul}_{0,f}(0, x) = 0_f. \tag{63f}$$

In the associativity identity, the inner products have scales $f + g$ and $g + h$; both final outputs have scale $f + g + h$.

Distributivity. Multiplication distributes over addition and subtraction in both arguments. For $x \in \mathcal{D}_f$ and $y, z \in \mathcal{D}_g$ or $x \in \mathcal{C}_f$ and $y, z \in \mathcal{C}_g$, we have

$$\Gamma \ \rhd_{\mathsf{EF}}^{W^{O(1)}} \ \mathsf{Mul}_{f,g}(x, y +_g z) = \mathsf{Mul}_{f,g}(x, y) +_{f+g} \mathsf{Mul}_{f,g}(x, z). \tag{64}$$

$$\Gamma \ \rhd_{\mathsf{EF}}^{W^{O(1)}} \ \mathsf{Mul}_{f,g}(x, y -_g z) = \mathsf{Mul}_{f,g}(x, y) -_{f+g} \mathsf{Mul}_{f,g}(x, z). \tag{65}$$

Right distributivity is a separate instance with $x, y \in \mathcal{D}_f$ and $z \in \mathcal{D}_g$ or $x, y \in \mathcal{C}_f$ and $z \in \mathcal{C}_g$; we have

$$\Gamma \ \rhd_{\mathsf{EF}}^{W^{O(1)}} \ \mathsf{Mul}_{f,g}(x +_f y, z) = \mathsf{Mul}_{f,g}(x, z) +_{f+g} \mathsf{Mul}_{f,g}(y, z). \tag{66}$$

$$\Gamma \ \rhd_{\mathsf{EF}}^{W^{O(1)}} \ \mathsf{Mul}_{f,g}(x -_f y, z) = \mathsf{Mul}_{f,g}(x, z) -_{f+g} \mathsf{Mul}_{f,g}(y, z). \tag{67}$$

Order transitivity. For real $x \in \mathcal{D}_f$, $y \in \mathcal{D}_g$, and $z \in \mathcal{D}_{g'}$, with comparisons aligned as specified above, we have

$$\Gamma : \ x \le y, \ y \le z \ \rhd_{\mathsf{EF}}^{W^{O(1)}} \ x \le z. \tag{68}$$

The mixed strict versions are

$$\Gamma : \ x < y, \ y \le z \ \rhd_{\mathsf{EF}}^{W^{O(1)}} \ x < z, \tag{69}$$

$$\Gamma : \ x \le y, \ y < z \ \rhd_{\mathsf{EF}}^{W^{O(1)}} \ x < z. \tag{70}$$

Order preservation. For addition, with real $x, y, z \in \mathcal{D}_f$, we have

$$\Gamma : \ x \le y \ \rhd_{\mathsf{EF}}^{W^{O(1)}} \ x +_f z \le y +_f z. \tag{71}$$

The strict version is

$$\Gamma : \ x < y \ \rhd_{\mathsf{EF}}^{W^{O(1)}} \ x +_f z < y +_f z. \tag{72}$$

For two pairs of real operands $x_1, x_2, y_1, y_2 \in \mathcal{D}_f$, we can add their inequalities directly:

$$\Gamma : \ x_1 \le y_1, \quad x_2 \le y_2 \ \rhd_{\mathsf{EF}}^{W^{O(1)}} \ x_1 +_f x_2 \le y_1 +_f y_2. \tag{73}$$

Two applications of (71), using commutativity (62a) and Lemma B.4 to add y_1 on the left in the second, give

$$x_1 +_f x_2 \le y_1 +_f x_2 \le y_1 +_f y_2.$$

Order transitivity (68) combines these checks. The constant number of rule applications and any missing intermediate arithmetic and check circuits have total additional cost $W^{O(1)}$.

For multiplication, with real $x, y \in \mathcal{D}_f$ and $c \in \mathcal{D}_g$, we have

$$\Gamma : \ x \le y, \ c \ge 0 \ \rhd_{\mathsf{EF}}^{W^{O(1)}} \ \mathsf{Mul}_{f,g}(x, c) \le \mathsf{Mul}_{f,g}(y, c). \tag{74}$$

The strict version has

$$\Gamma : \ x < y, \ c > 0 \ \rhd_{\mathsf{EF}}^{W^{O(1)}} \ \mathsf{Mul}_{f,g}(x, c) < \mathsf{Mul}_{f,g}(y, c). \tag{75}$$

These are order assertions about real words; the sign checks are part of the listed assertions.

Nonnegativity of absolute value. For a scaled real word $x \in \mathcal{D}_f$, including an integer word at scale zero,

$$\Gamma \ \rhd_{\mathsf{EF}}^{W^{O(1)}} \ 0 \le |x|. \tag{76}$$

Complex norm identities and bounds. For complex $x, y \in \mathcal{C}_f$, the triangle inequality is

$$\Gamma \ \rhd_{\mathsf{EF}}^{W^{O(1)}} \ |x +_f y|_1 \le |x|_1 + |y|_1. \tag{77}$$

The absolute values and the right-hand sum use the same full-result operators. Their widths follow from their input widths.

For complex $x \in \mathcal{C}_f$ and $y \in \mathcal{C}_g$, we also have

$$\begin{gathered}
\Gamma \ \rhd_{\mathsf{EF}}^{W^{O(1)}} \\
|\bar{x}|_1 = |x|_1, \qquad (78a) \\
|\mathsf{Neg}_f(x)|_1 = |x|_1, \qquad (78b) \\
|\mathsf{Mul}_{f,g}(x,y)|_1 \le |x|_1 |y|_1, \qquad (78c) \\
x\bar{x} = |x|_2^2 + \mathrm{i}0, \qquad (78d) \\
(|x|_1)^2 \le 2|x|_2^2, \qquad (78e) \\
|\mathsf{Mul}_{f,g}(x,y)|_2^2 = |x|_2^2 |y|_2^2. \qquad (78f)
\end{gathered}$$

All operations use the omitted-scale convention and retain full numerators. The conjugation and negation identities use scale f. The product bound for the complex 1-norm compares values at scale $f + g$. The conjugate-product identity and norm comparison use scale $2f$; the zero imaginary component embeds the squared norm into the complex representation. Multiplicativity of the squared 2-norm uses scale $2(f + g)$. The checks pad widths as needed.

Combining approximation checks. For real or complex scalars x, y, z and real error bounds e, e', fix a comparison scale f at least as large as all operand and bound scales. Then

$$\begin{gathered}
\Gamma : \ e, e' \ge 0, \quad x \overset{e}{\approx}_f y, \quad y \overset{e'}{\approx}_f z \ \rhd_{\mathsf{EF}}^{W^{O(1)}} \\
x \overset{e+e'}{\approx}_f z.
\end{gathered} \tag{79}$$

For length-N real or complex vectors $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$, with f at least as large as the scales of all entries and both bounds, we have

$$\begin{gathered}
\Gamma : \ e, e' \ge 0, \quad \boldsymbol{x} \overset{e}{\approx}_f \boldsymbol{y}, \quad \boldsymbol{y} \overset{e'}{\approx}_f \boldsymbol{z} \ \rhd_{\mathsf{EF}}^{NW^{O(1)}} \\
\boldsymbol{x} \overset{e+e'}{\approx}_f \boldsymbol{z}.
\end{gathered} \tag{80}$$

Complex embedding of real vectors. For a length-N real vector $\boldsymbol{v}$ and a nonnegative real bound e, use a common comparison scale f. Write $\boldsymbol{v} + \mathrm{i}0$ for the complex vector obtained by adjoining a hardwired zero imaginary component to each entry. Then

$$\begin{gathered}
\Gamma : \ \|\boldsymbol{v}\|_\infty \le e \ \rhd_{\mathsf{EF}}^{NW^{O(1)}} \\
\|\boldsymbol{v} + \mathrm{i}0\|_\infty \le e,
\end{gathered} \tag{81}$$

and conversely,

$$\begin{gathered}
\Gamma : \ \|\boldsymbol{v} + \mathrm{i}0\|_\infty \le e \ \rhd_{\mathsf{EF}}^{NW^{O(1)}} \\
\|\boldsymbol{v}\|_\infty \le e.
\end{gathered} \tag{82}$$

Promotion identities and error checks. For $x \in \mathcal{D}_f$ or $x \in \mathcal{C}_f$, and fixed $f \le g \le g'$, we have

$$\Gamma \ \rhd_{\mathsf{EF}}^{W^{O(1)}} \ \mathsf{Prmt}_{g'}(\mathsf{Prmt}_g(x)) = \mathsf{Prmt}_{g'}(x). \tag{83}$$

The same wiring proof establishes that $\mathsf{Prmt}_g(x)$ represents the same value as x: its numerator is the original numerator shifted left by $g - f$. Promotion increases the actual input width by $g - f$, so it is exact on every input word.

Promotion commutes with addition: for $x, y \in \mathcal{D}_f$ or $x, y \in \mathcal{C}_f$, and fixed $g \ge f$, we have

$$\Gamma \ \rhd_{\mathsf{EF}}^{W^{O(1)}} \ \mathsf{Prmt}_g(x +_f y) = \mathsf{Prmt}_g(x) +_g \mathsf{Prmt}_g(y). \tag{84}$$

Both sides compute the same full numerator, with $g - f$ low zero bits appended. Their equality uses only the shift and addition inference rules.

For multiplication, with $q \in \mathcal{D}_t$, $x \in \mathcal{D}_f$, or $q \in \mathcal{C}_t$, $x \in \mathcal{C}_f$, and fixed $g \ge f$, we have

$$\Gamma \ \rhd_{\mathsf{EF}}^{W^{O(1)}} \ \mathsf{Prmt}_{t+g}\big(\mathsf{Mul}_{t,f}(q, x)\big) = \mathsf{Mul}_{t,g}\big(q, \mathsf{Prmt}_g(x)\big). \tag{85}$$

For the same inputs and scales, promotion of the first operand satisfies

$$\Gamma \ \rhd_{\mathsf{EF}}^{W^{O(1)}} \ \mathsf{Mul}_{g,t}\big(\mathsf{Prmt}_g(x), q\big) = \mathsf{Prmt}_{g+t}\big(\mathsf{Mul}_{f,t}(x, q)\big). \tag{86}$$

In each identity, both sides compute the same full product numerator shifted by $g - f$. Applying the two identities successively moves both operand promotions outside the product.

Promotion also preserves error checks. Let A, B, E be signed integer numerator words available in Γ and let $d \ge 0$ be a fixed shift. Then

$$\Gamma : \ E \ge 0 \ \rhd_{\mathsf{EF}}^{W^{O(1)}} \ [\,|A - B| \le E\,] \ \longleftrightarrow \ [\,|2^d A - 2^d B| \le 2^d E\,]. \tag{87}$$

The brackets denote Boolean check outputs. All differences, absolute values, and shifts here retain full integer results, so no range premise is needed. After common-scale alignment, the complex 1-norm version uses component numerators $A_{\mathrm{re}}, A_{\mathrm{im}}, B_{\mathrm{re}}, B_{\mathrm{im}}$ and the same premise $E \ge 0$; it follows by adding the two full component absolute-value outputs.

In approximation notation, for real or complex scalars x, y and a real error bound e, let f_0 be their default comparison scale and fix $f \ge f_0$. Then

$$\Gamma : \ e \ge 0 \ \rhd_{\mathsf{EF}}^{W^{O(1)}} \ \big[x \overset{e}{\approx} y\big] \ \longleftrightarrow \ \big[x \overset{e}{\approx}_f y\big]. \tag{88}$$

For length-N real or complex vectors $\boldsymbol{u}, \boldsymbol{v}$ and a real error bound e, let f_0 be the maximum scale among all their entries and e. For fixed $f \ge f_0$, we have

$$\Gamma : \ e \ge 0 \ \rhd_{\mathsf{EF}}^{NW^{O(1)}} \ \big[\boldsymbol{u} \overset{e}{\approx} \boldsymbol{v}\big] \ \longleftrightarrow \ \big[\boldsymbol{u} \overset{e}{\approx}_f \boldsymbol{v}\big]. \tag{89}$$

Arithmetic modulo powers of two. Let x be a signed w-bit integer word available in Γ and fix $0 \le k \le w$. Using the quotient and remainder circuits defined in Appendix A.1, we have

$$\Gamma \ \rhd_{\mathsf{EF}}^{W^{O(1)}}$$

$$x = 2^k \lfloor x/2^k \rfloor + (x \bmod 2^k), \tag{90a}$$

$$0 \le x \bmod 2^k < 2^k. \tag{90b}$$

For signed integer words x, y, z and a fixed $k \geq 0$ with $k = O(W)$, adding a multiple of 2^k preserves the remainder:

$$\Gamma \ \rhd_{\mathsf{EF}}^{W^{O(1)}} \ (x + 2^k z) \bmod 2^k = x \bmod 2^k. \tag{91}$$

Reducing an operand before addition or multiplication also preserves the remainder:

$$\Gamma \ \rhd_{\mathsf{EF}}^{W^{O(1)}} \ \big(x + (y \bmod 2^k)\big) \bmod 2^k = (x + y) \bmod 2^k. \tag{92}$$

$$\Gamma \ \rhd_{\mathsf{EF}}^{W^{O(1)}} \ (xy) \bmod 2^k = \big((x \bmod 2^k)y\big) \bmod 2^k. \tag{93}$$

All arithmetic in these identities retains its full result. Sign-extend operands if needed before extracting their low k bits, and zero-extend unsigned remainders when using them as signed arithmetic inputs.

Comparison and addition at half the modulus. Let $1 \leq w = O(W)$ and let x be an unsigned w-bit word. Write $x = (x_0, \ldots, x_{w-1})$, listing its bits from lowest to highest. Then

$$\Gamma \ \rhd_{\mathsf{EF}}^{W^{O(1)}} \ \big[\mathsf{LT}_{\mathrm{u}}(x, 2^{w-1}) \longleftrightarrow \neg x_{w-1}\big]. \tag{94}$$

Let $b := (x + 2^{w-1}) \bmod 2^w$, using full integer addition followed by the low-w-bit remainder wiring. Then

$$\Gamma \ \rhd_{\mathsf{EF}}^{W^{O(1)}} \ [x < 2^{w-1}] \Longrightarrow \mathsf{EQ}\big(b, (x_0, \ldots, x_{w-2}, 1)\big). \tag{95}$$

The comparison in the antecedent is LT_{u}. The word on the right reuses the lower input bits and hardwires its highest bit to one; for $w = 1$, it is the one-bit word (1).

Rounding and integer gaps. For a real word $x \in \mathcal{D}_f$ and a real integer word $a \in \mathcal{D}_0$ available in Γ, we have

$$\Gamma \ \rhd_{\mathsf{EF}}^{W^{O(1)}} \ \big(|x - a| < \tfrac{1}{2}\big) \implies \mathsf{Round}(x) = a. \tag{96}$$

For the antecedent, promote to scale $\max(f, 1)$ and use full subtraction, absolute value, and comparison. The conclusion uses the rounding circuit of Appendix A.2, with its full integer quotient output. At $f = 0$ rounding is the identity.

For real integer words $a, b \in \mathcal{D}_0$ available in Γ, the integer-gap implication is

$$\Gamma \ \rhd_{\mathsf{EF}}^{W^{O(1)}} \ |a - b| < 1 \implies a = b. \tag{97}$$

The antecedent uses full-result subtraction and absolute value. In both this statement and the preceding one, the antecedent is part of the conclusion circuit.

The scale-zero instances give the integer versions of these rules. Different input scales are aligned by promotion before applying an equal-scale statement. General arithmetic uses the full-result operators, with the specified power-of-two wiring for scalar factors. No operation narrows a numerator.

D Inference Rules for Permutations, Routing, and Reordering

D.1 Routing Identities

Fixed permutation circuits. Let $m = 2^s$ and let each entry of $\boldsymbol{v}$ have w bits. This construction works for arbitrary bit words; for entries in $\mathcal{D}_f$, take w equal to their common padded numerator width, and for complex entries take twice their common padded component width. Routing preserves this type; the address words are separate s-bit integers. For a fixed permutation π of $[m]$, define the array circuit Π by $(\Pi \boldsymbol{v})_j := v_{\pi(j)}$. This

circuit consists of fixed wiring: output word j is connected to input word $\pi(j)$. The circuit $\mathsf{Sel}_{m,w}(\Pi \boldsymbol{v}, i)$ attaches the ordinary data selector to these permuted wires. The alternative circuit first computes the s-bit address $\pi(i)$ and then feeds it to $\mathsf{Sel}_{m,w}(\boldsymbol{v}, \pi(i))$. A general fixed permutation uses the hardwired address table

$$\pi(i) := \mathsf{Sel}_{m,s}((\pi(0), \ldots, \pi(m-1)), i),$$

which has $O(ms)$ gates. A rotation or reflection can instead use a short modular subtraction or negation circuit on s bits.

$$\boldsymbol{v} \longrightarrow (v_{\pi(0)}, \ldots, v_{\pi(m-1)}) \longrightarrow \mathsf{Sel}(\Pi \boldsymbol{v}, i)$$

fixed data wiring; selector address is i

$$i \longrightarrow \mathsf{Sel}((\pi(j))_{j<m}, i) = \pi(i) \longrightarrow \mathsf{Sel}(\boldsymbol{v}, \pi(i))$$

address lookup; data selector reads the original array $\boldsymbol{v}$

Figure 7: Two implementations whose output words are proved equal. The upper circuit permutes data wires; the lower circuit computes the address. A short address-arithmetic circuit may replace the lower lookup table.

Lemma D.1 (Selection through a fixed permutation)**.** *Let $\boldsymbol{v}$ be an array of $m = 2^s$ words of width w, let i be an s-bit address, and fix a permutation π of $[m]$. For the circuits defined above,*

$$\Gamma \rhd^{m \log^{O(1)} m \, w^{O(1)}}_{\mathsf{EF}} \mathsf{EQ}\big(\mathsf{Sel}_{m,w}(\Pi \boldsymbol{v}, i), \mathsf{Sel}_{m,w}(\boldsymbol{v}, \pi(i))\big) = 1.$$

Proof. For a fixed permutation, the EF proof works by brute-force enumerating over all possible addresses j. Fix one possible address j. If the input index i equals j, the first circuit selects position j of the rearranged array, which is the original word $v_{\pi(j)}$. The second circuit computes the address $\pi(j)$ and selects that same original word. For example, if $m = 4$ and the address table is $(2, 0, 3, 1)$, then under $i = 2$ the first circuit selects the third entry of (v_2, v_0, v_3, v_1), while the second computes address 3 and selects v_3. The bits of v_3 remain unknown: both outputs are connected, through their selector gates, to the same input wires. We prove this conditional agreement for every possible address j, then use the fact that i must equal one of them.

Introduce address-decoder bits $d_j \leftrightarrow [i = j]$ for every hardwired $j < m$. Here d_j is the Boolean flag saying that the variable input index equals the constant address j. If i has bits $i_{s-1}, \ldots, i_0$, its check is the AND of s literals: use i_r when bit r of j is one, and $\neg i_r$ when that bit is zero. For instance, with two index bits, $d_2 \leftrightarrow (i_1 \wedge \neg i_0)$. Apply Lemma B.6 to the permuted array and to the hardwired address table. Under d_j, these two applications give

$$\mathsf{Sel}(\Pi \boldsymbol{v}, i) = (\Pi \boldsymbol{v})_j = v_{\pi(j)}, \qquad \pi(i) = \pi(j).$$

Apply the same lemma to $\mathsf{Sel}(\boldsymbol{v}, \pi(i))$ at the fixed address $\pi(j)$, and use the second equality to obtain $\mathsf{Sel}(\boldsymbol{v}, \pi(i)) = v_{\pi(j)}$ under d_j. Combining the two output equalities, bitwise and then by Lemma B.2, gives $d_j \to \mathsf{EQ}(\mathsf{Sel}(\Pi \boldsymbol{v}, i), \mathsf{Sel}(\boldsymbol{v}, \pi(i)))$. The three lemma applications cost $O((w + s) \log(m + 1))$ per j; combining their conclusions adds $O(w + s)$, including the missing checks. Thus the additional cost per address is $w \log^{O(1)} m$. The full selector circuits, of sizes $O(mw)$ and $O(ms)$, are named once and shared across these proofs.

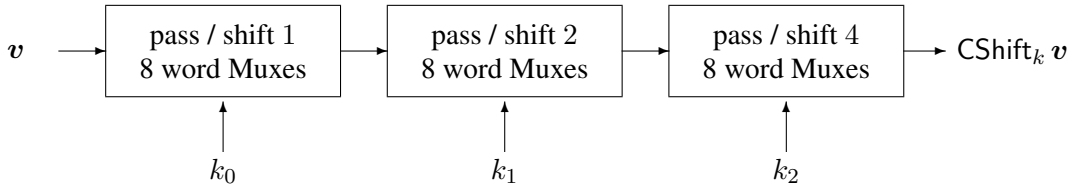
Let c name the output of $\mathsf{EQ}(\mathsf{Sel}(\Pi \boldsymbol{v}, i), \mathsf{Sel}(\boldsymbol{v}, \pi(i)))$. The preceding cases give $d_j \to c$ for all $j < m$. Lemma B.5 combines them to derive c, at additional cost $O(m \log(m + 1))$ for its auxiliary decoder, checks,

and small-step inferences. The index i remains free throughout. Including the applications of Lemma B.6, the total additional cost is $m \log^{O(1)} m\, w^{O(1)}$. Constructing the conclusion circuits from the supplied operand wires also fits this bound. For a single stage controlled by a bit β, carry out this construction under β and under $\neg\beta$, and apply Lemma B.5 to this one-bit address to combine the two conclusions at constant additional cost. □

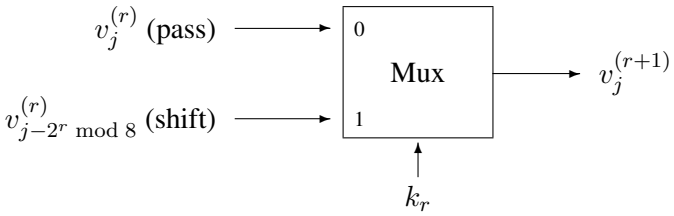
Variable shifts and reflection. Reflection Reflect is the fixed wiring $(\mathsf{Reflect}\, \boldsymbol{v})_j := v_{-j \bmod m}$, so it is a fixed permutation. A variable shift CShift_k uses a barrel circuit: writing $k = \sum_{r<s} 2^r k_r$, set $\boldsymbol{v}^{(0)} := \boldsymbol{v}$ and implement stage r by the bitwise word multiplexers

$$v_j^{(r+1)} := \mathsf{Mux}(k_r, v^{(r)}_{j-2^r \bmod m}, v_j^{(r)}).$$

The two candidate word connections are fixed wires, and the control is the input bit k_r. Each stage contains m word multiplexers, all sharing the control k_r; the fixed rotated input wires wrap around modulo m. Set $\mathsf{CShift}_k\, \boldsymbol{v} := \boldsymbol{v}^{(s)}$. There are s stages and $O(mws)$ gates.



One output word in stage r:



Example: $k = 5 = (101)_2$ selects shifts $1 + 0 + 4$.

Figure 8: Barrel implementation of CShift_k for $m = 8$. Each horizontal connection in the upper diagram carries the entire eight-word array. The lower diagram shows one of the eight word multiplexers inside a stage. A word multiplexer uses one Boolean multiplexer per bit. In general there are $\log_2 m$ stages, with shifts $1, 2, 4, \ldots, m/2$; the selected shifts add to k.

Lemma D.2 (Selection through shifts and reflection)**.** *Let $\boldsymbol{v}$ be an array of $m = 2^s$ words of width w, and let i, k be s-bit addresses. Then*

$$\Gamma \;\rhd^{m \log^{O(1)} m\, w^{O(1)}}_{\mathsf{EF}}$$

$$\mathsf{Sel}(\mathsf{CShift}_k\, \boldsymbol{v}, i) = \mathsf{Sel}(\boldsymbol{v}, i - k \bmod m), \tag{98}$$

$$\mathsf{Sel}(\mathsf{Reflect}\, \boldsymbol{v}, i) = \mathsf{Sel}(\boldsymbol{v}, -i \bmod m). \tag{99}$$

Proof of (98) *(cyclic shifts).* For $m = 1$, the shift and selectors are the identity wiring. Assume henceforth that $s \geq 1$. We follow the selected output backwards through the barrel circuit. Passing backwards through stage r subtracts $k_r 2^r$ from its address; after all stages, these subtractions add up to k. The arrays $\boldsymbol{v}^{(0)}, \ldots, \boldsymbol{v}^{(s)}$ are already present in the supplied shift circuit, with $\boldsymbol{v}^{(0)} = \boldsymbol{v}$ and $\boldsymbol{v}^{(s)} = \mathsf{CShift}_k \boldsymbol{v}$.

1. *Construct the intermediate addresses and selectors.* Starting with the existing address $i_s := i$, construct

$$i_r := (i_{r+1} - k_r 2^r) \bmod m \qquad (r = s-1, \dots, 0).$$

Each address uses a subtractor followed by low-s-bit wiring; the word $k_r 2^r$ has bit k_r in position r and zeros elsewhere. Construct $\mathsf{Sel}(\boldsymbol{v}^{(r)}, i_r)$ for $0 \le r < s$, reusing the final selector $\mathsf{Sel}(\boldsymbol{v}^{(s)}, i_s)$ from the conclusion. Each intermediate selector is built once and reused wherever it occurs in the stage comparisons. These selectors and addresses cost $O(smw) + s^{O(1)}$ additional gates.

2. *Prove the equality across each stage.* For each $r < s$, we prove

$$\mathsf{Sel}(\boldsymbol{v}^{(r+1)}, i_{r+1}) = \mathsf{Sel}(\boldsymbol{v}^{(r)}, i_r).$$

There are two cases. If $k_r = 0$, the stage multiplexers pass through $\boldsymbol{v}^{(r)}$ and $i_r = i_{r+1}$. If $k_r = 1$, the stage is the fixed permutation $j \mapsto j - 2^r \bmod m$, and $i_r = i_{r+1} - 2^r \bmod m$. Thus the controlled-permutation argument at the end of Lemma D.1 proves the displayed equality: apply the fixed permutation rule in each case and combine the cases by Lemma B.5 on the single bit k_r. The conditional forms are supplied by Lemma B.1. Each stage costs $m \log^{O(1)} m\, w^{O(1)}$, including its missing comparison circuits and proof auxiliaries. The $s = \log m$ applications therefore have total cost $m \log^{O(1)} m\, w^{O(1)}$.

3. *Join the equalities and identify the final address.* Equality transitivity, (55) of Lemma B.4, joins the stage conclusions into

$$\begin{aligned}\mathsf{Sel}(\mathsf{CShift}_k \boldsymbol{v}, i) &= \mathsf{Sel}(\boldsymbol{v}^{(s-1)}, i_{s-1}) \\ &= \cdots = \mathsf{Sel}(\boldsymbol{v}^{(0)}, i_0) = \mathsf{Sel}(\boldsymbol{v}, i_0).\end{aligned}$$

These are equalities between the shared selector outputs from Step 1; joining them and constructing the missing word checks costs $O(sw)$.

It remains to prove that i_0 is the desired address. The addition identities (62) and reduction-before-addition rule (92) of Lemma C.1 remove the intermediate reductions and combine the subtractions, giving

$$i_0 = \left(i - \sum_{r<s} k_r 2^r \right) \bmod m = (i - k) \bmod m.$$

For the last equality, the sum of the binary place values is k: each bit position has only its own bit k_r, so the adder cells produce no carries. The missing arithmetic circuits, checks, and derivations cost $s^{O(1)}$; their working widths are $O(s)$. Apply equality propagation (50) of Lemma B.4 through the selector on $\boldsymbol{v}$ to obtain

$$\mathsf{Sel}(\boldsymbol{v}, i_0) = \mathsf{Sel}(\boldsymbol{v}, i - k \bmod m).$$

This application costs $O(mw + s)$, including the missing checks, and transitivity now gives (98).

The three steps together cost $m \log^{O(1)} m\, w^{O(1)}$. □

Proof of (99) *(reflection).* Reflection is the fixed permutation $\pi(j) := -j \bmod m$. Apply Lemma D.1, using the modular negation circuit for $\pi(i)$, to obtain

$$\mathsf{Sel}(\mathsf{Reflect}\, \boldsymbol{v}, i) = \mathsf{Sel}(\boldsymbol{v}, -i \bmod m).$$

This costs $m \log^{O(1)} m\, w^{O(1)}$, so both conclusions together fit the displayed budget. □

Lemma D.3 (Cancellation of shifted reflections)**.** *Let $\boldsymbol{v}$ be an array of $m = 2^s$ words of width w, and let k be an s-bit address. Then*

$$\Gamma \;\rhd_{\mathsf{EF}}^{m\log^{O(1)} m\, w^{O(1)}}\; \mathsf{CShift}_k\,\mathsf{Reflect}\,\mathsf{CShift}_k\,\mathsf{Reflect}\,\boldsymbol{v} = \boldsymbol{v}.$$

Proof. Let i be an arbitrary s-bit index (Section 3.2). Let $A := \mathsf{CShift}_k\,\mathsf{Reflect}\,\mathsf{CShift}_k\,\mathsf{Reflect}\,\boldsymbol{v}$ name the existing routed output array; this introduces no gates.

Apply the two identities of Lemma D.2 successively, each time removing the outermost routing from the selected array. All address operations in the following display are modulo m:

$$\begin{aligned}
&\mathsf{Sel}(\mathsf{CShift}_k\,\mathsf{Reflect}\,\mathsf{CShift}_k\,\mathsf{Reflect}\,\boldsymbol{v}, i)\\
&\quad= \mathsf{Sel}(\mathsf{Reflect}\,\mathsf{CShift}_k\,\mathsf{Reflect}\,\boldsymbol{v}, i-k) && \text{by (98)}\\
&\quad= \mathsf{Sel}(\mathsf{CShift}_k\,\mathsf{Reflect}\,\boldsymbol{v}, -(i-k)) && \text{by (99)}\\
&\quad= \mathsf{Sel}(\mathsf{Reflect}\,\boldsymbol{v}, -(i-k)-k) && \text{by (98)}\\
&\quad= \mathsf{Sel}(\boldsymbol{v}, -(-(i-k)-k)) && \text{by (99)}\\
&\quad= \mathsf{Sel}(\boldsymbol{v}, i) && \text{by Lemmas B.4 and C.1.}
\end{aligned}$$

For the last equality, the addition identities and quotient-and-remainder statement of Lemma C.1 prove that the final modular address equals i. Lemma B.4 propagates this address equality through the selector on $\boldsymbol{v}$. Join the word equalities by transitivity, (55) of the same lemma.

The indexed conclusion supplies the two endpoint selectors. Construct the intermediate selectors, modular address circuits, and missing equality checks once, reusing the existing routed arrays. Their construction, the four applications of Lemma D.2, and the final address derivation give

$$\Gamma' \;\rhd_{\mathsf{EF}}^{S}\; \mathsf{EQ}(\mathsf{Sel}(A, i), \mathsf{Sel}(\boldsymbol{v}, i)) = 1, \qquad S = m\log^{O(1)} m\, w^{O(1)},$$

for every context $\Gamma' \supseteq \Gamma$ that contains the index i. All additional i-dependent circuits and derivations are included in S; the intermediate selectors have $O(mw)$ gates each and the address arithmetic has $s^{O(1)}$ gates and proof size.

Apply index-generalization (Lemma B.8) to the two arrays A and $\boldsymbol{v}$, with the local predicate $P(x, y) := \mathsf{EQ}(x, y)$. This check has size $|P| = O(w)$; it only compares the two selected words. Both arrays and the shift k belong to the base context Γ. Thus the lemma gives

$$\Gamma \;\rhd_{\mathsf{EF}}^{S+O(m(w+s))}\; [\mathsf{EQ}(A_j, v_j) = 1]_{j<m}.$$

Finally, Lemma B.2 folds the returned coordinate assertions into this check with $O(m)$ additional gates and small-step inferences. No unfolding into individual bits is needed. Since $|P| = O(w)$ and $s = \log m$, the total additional cost is $m\log^{O(1)} m\, w^{O(1)}$, as claimed. □

Lemma D.4 (Routing preserves entrywise checks)**.** *Let $\boldsymbol{v}$ be an array of $m = 2^s$ words of width w, and let k be an s-bit address. Let P be a single-output Boolean check of circuit size $|P|$ whose inputs are one word of width w and, optionally, an s-bit address. Besides these inputs, P may read further wires z of Γ, and all copies of P below read the same wires z. Then*

$$\Gamma :\; [P(v_j, j) = 1]_{j<m} \;\rhd_{\mathsf{EF}}^{m\log^{O(1)} m\,(|P|+w)}\; \big[P\big((\mathsf{CShift}_k\,\mathsf{Reflect}\,\boldsymbol{v})_j,\, k - j \bmod m\big) = 1\big]_{j<m}. \tag{100}$$

Here the address j in the premises is hardwired, and the address $k - j \bmod m$ in the conclusions is computed by a modular subtractor on s bits. If P does not read an address, the second argument is omitted. In particular, let $\boldsymbol{v}$ be a real or complex vector and V a real bound word, with all scales and working numerator widths $O(W)$. Then

$$\Gamma :\; \|\boldsymbol{v}\|_\infty \le V \;\rhd_{\mathsf{EF}}^{m\log^{O(1)} m\, W^{O(1)}}\; \|\mathsf{CShift}_k\,\mathsf{Reflect}\,\boldsymbol{v}\|_\infty \le V. \tag{101}$$

Proof. Write $\boldsymbol{u}^{(0)} := \mathsf{Reflect}\,\boldsymbol{v}$, and let $u_j^{(r+1)} := \mathsf{Mux}(k_r, u^{(r)}_{j-2^r \bmod m}, u_j^{(r)})$ be the stages of the barrel circuit, so that $\boldsymbol{u}^{(s)} = \mathsf{CShift}_k\,\mathsf{Reflect}\,\boldsymbol{v}$. For $r \le s$, let $k_{<r}$ be the s-bit address whose r low-order bits are those of k and whose other bits are zero; it is given by wiring. Let $a_j^{(r)}$ be the modular subtractor computing $k_{<r} - j \bmod m$, with j hardwired. This is the original address of the entry at position j after r stages, and $a_j^{(s)}$ is the address $k - j \bmod m$ of the conclusion. We derive the invariant

$$\big[P\big(u_j^{(r)}, a_j^{(r)}\big) = 1\big]_{j<m}$$

for $r = 0, \ldots, s$. At $r = s$, it is the conclusion. If P does not read an address, omit the addresses throughout.

Reflection. The word $u_j^{(0)}$ is the word $v_{-j \bmod m}$, so the premise at position $-j \bmod m$ is the check $P(u_j^{(0)}, -j \bmod m)$, with the address hardwired. The subtractor $a_j^{(0)}$ has only constant inputs. Evaluating it gate by gate derives that it outputs this hardwired address, and Lemma B.4 (52) gives $P(u_j^{(0)}, a_j^{(0)})$.

One stage. Fix $r < s$ and $j < m$, and put $j' := j - 2^r \bmod m$.

- *Under* k_r, the multiplexer definitions give $u_j^{(r+1)} = u_{j'}^{(r)}$ bit by bit. The address $k_{<r+1}$ agrees bit by bit with $k_{<r} + 2^r$, whose bit r is hardwired to one, so conditional equality propagation through the subtractor gives $a_j^{(r+1)} = k_{<r} + 2^r - j \bmod m$. The modular addition identities of Lemma C.1 identify the latter address with $a_{j'}^{(r)} = k_{<r} - j' \bmod m$.

- *Under* $\neg k_r$, likewise, $u_j^{(r+1)} = u_j^{(r)}$, and $a_j^{(r+1)} = a_j^{(r)}$ because $k_{<r+1}$ and $k_{<r}$ agree bit by bit.

In each case, the conditional substitution (53) of Lemma B.4, applied to the word and its address together, carries the invariant at j', respectively at j, to $P(u_j^{(r+1)}, a_j^{(r+1)})$ under the case assumption. Lemma B.5 (57), for the one-bit address k_r, combines the two cases. The further wires z are shared by all copies of P, so they need no derivation.

Cost. Each position at each stage costs $O(|P|+w)+s^{O(1)}$, including the new copies of P, the subtractors, and the missing equality checks. Over the m positions and s stages, and with the reflection step, the total is $O(ms(|P| + w)) + ms^{O(1)}$, which is at most $m \log^{O(1)} m\,(|P| + w)$.

Norm bounds. The check $\|\boldsymbol{v}\|_\infty \le V$ is the balanced AND of one check $|v_j|_1 \le V$ for each entry, which reads the bound V but no address. Routing preserves the entry type, so $\|\mathsf{CShift}_k\,\mathsf{Reflect}\,\boldsymbol{v}\|_\infty \le V$ is the balanced AND of the same checks on the routed entries. Unfold the premise by Lemma B.3, apply (100) to this check, of size $W^{O(1)}$, and fold the routed checks by Lemma B.2. This costs $m \log^{O(1)} m\, W^{O(1)}$. □

D.2 Reordering Sums and Products

Notation: balanced summation tree. When a summation denotes an arithmetic circuit, write $\sum_{j<m} x_j$ for the balanced binary addition tree with leaves $x_0, \ldots, x_{m-1}$ in index order. Recursively split each list of $k \ge 2$ leaves into consecutive halves of sizes $\lfloor k/2 \rfloor$ and $\lceil k/2 \rceil$. A one-leaf tree returns that leaf; an empty sum is zero. For $m = 2^s$, the tree is perfectly balanced, with every leaf at depth s. The leaves may have different fixed scales f_j and component widths w_j. Each internal addition uses the omitted-scale convention of Appendix A.2: promote its operands to their maximum scale, then add their full numerators. Thus a subtree has the maximum scale of its leaves. Put $f = \max_j f_j$ and $w = \max_j (w_j + f - f_j)$; the whole tree has output width at most $w + \lceil \log m \rceil$ for $m \ge 1$. For complex words, use one such tree for each component. When every leaf already has scale f, we may write $\sum_{j<m}^{[f]} x_j$ to display that common scale explicitly; every addition is then $+_f$. This annotation does not insert extra promotions.

Lemma D.5 (Promotion of a sum tree)**.** *Let $x = (x_j)_{j<m}$ be an array of $m = 2^s$ real or complex entries with fixed scales f_j and component widths w_j. Fix $g \geq \max_j f_j$. Assume $g = O(W)$ and $\max_j(w_j+g-f_j)+s = O(W)$. Then*

$$\Gamma \ \rhd_{\mathsf{EF}}^{mW^{O(1)}} \ \mathsf{Prmt}_g\left(\sum_{j<m} x_j\right) = \sum_{j<m} \mathsf{Prmt}_g(x_j).$$

Proof. At a node with child outputs u, v, put $f' = \max\{\mathrm{sc}(u), \mathrm{sc}(v)\}$. Its actual addition is $\mathsf{Prmt}_{f'}(u) +_{f'} \mathsf{Prmt}_{f'}(v)$. Promotion of addition and composition of promotions from Lemma C.1 give

$$\mathsf{Prmt}_g\big(\mathsf{Prmt}_{f'}(u) +_{f'} \mathsf{Prmt}_{f'}(v)\big) = \mathsf{Prmt}_g(u) +_g \mathsf{Prmt}_g(v).$$

Proceed from the leaves to the root, citing the two child equalities at each node and applying the width-padding identities from the same lemma. Each of the $m-1$ addition nodes costs $W^{O(1)}$, including any missing auxiliary gates; the child circuits and their proofs are shared. All its working widths are at most $\max_j(w_j+g-f_j)+s+O(1) = O(W)$. Promotion of the leaf array acts coordinatewise. The same nodewise construction works for any fixed binary parenthesization, with one proof block per addition node. □

For a fixed permutation of entries with different representations, Π still routes each entry together with its fixed scale and width, using only wiring.

Lemma D.6 (Fixed permutations preserve a perfectly balanced sum)**.** *Let $\boldsymbol{v} = (v_j)_{j<m}$ be an array of $m = 2^s$ real or complex entries with fixed scales f_j and component widths w_j. Put*

$$f = \max_j f_j, \qquad w = \max_j(w_j + f - f_j).$$

Fix a permutation π of $[m]$. Then

$$\Gamma \ \rhd_{\mathsf{EF}}^{m \log^{O(1)} m \, w^{O(1)}} \ \sum_{j<m} (\Pi \boldsymbol{v})_j = \sum_{j<m} v_j.$$

Proof. This lemma is a generalization of (9) for scaled real or complex numbers, so the proof follows the same leaf-swap approach. For completeness, below we briefly recall the derivation and highlight the differences from the proof of (9). We decompose π into at most $m-1$ transpositions of leaves and prove one identity per transposition. For a single transposition, follow the two paths from the chosen leaves to their lowest common ancestor; along each path, associativity and commutativity expose the chosen leaf as one summand and collect the sibling-subtree outputs as the other, so that the two children of the common ancestor read $x + A$ and $y + B$; one identity at the ancestor exchanges x and y; the exposing steps are then applied in reverse. All unaffected subtrees are retained, the equality created at a node is propagated to the root through the $O(s)$ additions above it, and the root equalities of the successive trees are joined by transitivity (55). Section 4.2 describes this construction in detail, with a worked example and Figure 4, for integers in w-bit words added at one common width. We verify it here for scaled real and complex short words, where three things change.

1. *The leaves must first be brought to a common scale.* Let $\widehat{\boldsymbol{v}}$ be $\mathsf{Prmt}_f(\boldsymbol{v})$ with every component sign-extended to width w. The nodewise promotion proof of Lemma D.5, together with the padding identities of Lemma C.1, compares $\sum_{j<m} v_j$ with $\sum_{j<m}^{[f]} \widehat{v}_j$, and likewise compares their fixed permutations. Their roots already have scale f, so no output scale change is needed. These comparisons cost $m(w+s)^{O(1)}$: each promotion adds at most w zero columns, and each working numerator has width $w + O(s)$. It remains to prove the fixed-permutation identity for $\widehat{\boldsymbol{v}}$, whose entries have the common scale f and width w; the scale remains f throughout these addition-only trees. Complex entries are pairs of real numerators, and every step below applies to each component.

2. *Widths are not fixed in advance.* Rather than adding at one width at which no partial sum overflows, every addition retains its full result. The balanced sum tree therefore has output width $w + s$, and its permuted version has the same shape, with leaf j wired to $\widehat{v}_{\pi(j)}$. Intermediate regroupings are compared after sign extension, and the inference rules are applied to the full numerator circuits, with padding wherever two intermediate widths differ. A leaf swap exposes only the two root-to-leaf paths, so its temporary expressions have height $O(s)$ and widths $w + O(s)$. Successive swaps are compared on trees built from the same original leaves; the numerical result of one whole tree is never added as a new leaf of the next, so widths do not accumulate over the $m - 1$ swaps.

3. *The local identities are short-word rules.* If the two exchanged leaves carry x, y and the sums of the subtrees beside their paths are A, B, the identity at their common ancestor is

$$(x +_f A) +_f (y +_f B) = (y +_f A) +_f (x +_f B),$$

a constant number of applications of associativity and commutativity of addition from Lemma C.1; a missing sibling sum at an endpoint of a path is a zero numerator at scale f. Propagation through the unchanged ancestors is Lemma B.4, with the sibling outputs as shared inputs. Every reassociation is thus an EF derivation about the actual Add_f gates.

Cost. A swap needs $O(s)$ new arithmetic nodes and short-word proof blocks; all other nodes are shared. The generator builds the corresponding succession of sum trees, changing only the affected paths at each swap, so the $m - 1$ swaps cost $O(ms)$ short-word blocks and new arithmetic nodes in addition to the initial tree. The initial tree is the original sum and the final leaves are exactly the specified permutation wires. If the endpoint circuit was already named separately, compare corresponding adder gates to identify the two copies in $O(m)$ further blocks. With working widths $w + O(s)$, these blocks and the initial promotion comparisons fit the displayed bound. □

Lemma D.7 (Cyclic shifts preserve a perfectly balanced sum)**.** *Let $\boldsymbol{v}$ be an array of $m = 2^s$ real or complex entries, all at scale f and with common component width w. Let k be an s-bit shift word. Then*

$$\Gamma \ \rhd_{\mathsf{EF}}^{m \log^{O(1)} m \, w^{O(1)}} \ \sum_{j<m} (\mathsf{CShift}_k \, \boldsymbol{v})_j = \sum_{j<m} v_j.$$

Proof. Use the barrel-stage arrays $\boldsymbol{v}^{(0)}, \ldots, \boldsymbol{v}^{(s)}$, with $\boldsymbol{v}^{(0)} = \boldsymbol{v}$ and $\boldsymbol{v}^{(s)} = \mathsf{CShift}_k \, \boldsymbol{v}$. Every stage preserves scale f and component width w. At stage r, under control $k_r = 0$ the leaf multiplexers select $\boldsymbol{v}^{(r)}$, and under $k_r = 1$ they select its fixed rotation by 2^r. Propagate these leaf equalities through the output sum tree in $O(m)$ short-word blocks in each case. In the second case, apply Lemma D.6 to that fixed rotation. Eliminate the control bit to derive

$$\sum_{j<m} v_j^{(r+1)} = \sum_{j<m} v_j^{(r)}.$$

Name one sum tree for each actual stage array and join these equalities by transitivity (55). All s stages cost $O(m(s+1)^2)$ short-word blocks, with working widths $w + O(s)$. The bound includes the missing intermediate sum trees, fixed-permutation proof circuits, comparison checks, and small-step inferences. □

Notation: summation with arbitrary parenthesization. For a fixed binary parenthesization T of the entries of $x = (x_j)_{j<M}$, write $\sum_{j<M}^{T} x_j$ for the corresponding circuit using the omitted-scale addition convention of Appendices A.2 and A.3. The terms occur in index order, and T specifies which additions are performed first. No balance assumption is imposed: in general, parenthesizations T, T' need not have logarithmic depth. The terms' fixed scales and widths may differ. Each node has the maximum scale of

its leaves, and every addition retains its full result. When all leaves have scale f, we may write $\sum_{j<M}^{[f,T]} x_j$. Omitting the parenthesization annotation specifies the balanced tree defined above. The special superscript left specifies left-to-right accumulation:

$$\sum_{j<M}^{\mathrm{left}} x_j := \big(((x_0 + x_1) + x_2) + \cdots + x_{M-1}\big).$$

For a single term this is x_0, and for no terms it is zero. With an explicit common scale, write $\sum_{j<M}^{[f,\mathrm{left}]} x_j$.

Lemma D.8 (Reordering an arbitrarily parenthesized sum)**.** *Let $x = (x_0, \ldots, x_{M-1})$ be an array of real or complex values with fixed scales f_j and component widths w_j. Put*

$$f = \max_j f_j, \qquad w = \max_j (w_j + f - f_j).$$

Fix binary parenthesizations T, T' and a permutation π of $[M]$. If $f = O(W)$ and $w + M = O(W)$, then

$$\Gamma \;\rhd_{\mathsf{EF}}^{M \log^2(M+1) W^{O(1)}} \; \sum_{j<M}^{T} x_j = \sum_{j<M}^{T'} x_{\pi(j)}.$$

Proof. The two sides are compared through a fixed balanced tree. We give the construction for leaves that already share a common scale, and then reduce the general case to that one.

Equate each side with the balanced tree. Assume first that every leaf has the scale f, so that each leaf has component width at most w and every addition in either tree has scale f. Let B be the balanced tree on the M leaf positions in index order, as defined above; its gates are constructed once and shared. When the positions $a, \ldots, b$ are the leaves of a subtree of B, write $\Sigma[a, b]$ for the output of that subtree; for example, $\Sigma[4, 5] = x_4 + x_5$ is the output of one of its nodes, and $\Sigma[j, j] = x_j$. We compare T with B, recording one equality at each node of T (Figure 9).

A node of T adds the terms in an interval I of consecutive positions, and I is the disjoint union of the maximal subtrees of B contained in it, listed from left to right. There are $O(\log(M + 1))$ of them: only the two boundary paths of I can prevent a contained subtree from belonging to a larger one. The equality recorded at the node, its *record*, states that its output equals the sum of the outputs of these subtrees, listed from left to right. We always add the entries $S_1, \ldots, S_k$ of such a list from the right, as $S_1 + (S_2 + (\cdots + S_k))$; a one-member list uses that wire directly. In the figure, every node of T is labeled with its record.

A node obtains its record from the records of its two children in three steps. First, since the node adds the outputs of its children, the records of the children show that its output equals the sum of the two lists, which is bracketed as a sum of two sums; Lemma B.4 carries the two child equalities through this node's adder, after padding its inputs to a common width. Second, associativity, $(P + Q) + R = P + (Q + R)$, moves the entries of the first list out of the outer parentheses one at a time, until the sum is a single list, the concatenation of the two. Third, whenever two adjacent entries of this list are the outputs of sibling subtrees, associativity groups them, and their sum is the output of their parent in the balanced tree, which replaces them. Once no two adjacent entries are siblings, the list consists of the maximal subtrees inside I, and it is the record of the node.

For example, when T is the left-to-right tree of Figure 9, its root adds x_7 to the output u of its left child. This child adds the terms in $\{0, \ldots, 6\}$, the disjoint union of the three orange subtrees in the figure, so its

record is $u = \Sigma[0,3] + (\Sigma[4,5] + x_6)$. The three steps give the chain

$$\begin{aligned}
u + x_7 &= \big(\Sigma[0,3] + (\Sigma[4,5] + x_6)\big) + x_7 && \text{record of the child} \\
&= \Sigma[0,3] + \big((\Sigma[4,5] + x_6) + x_7\big) && \text{associativity} \\
&= \Sigma[0,3] + \big(\Sigma[4,5] + (x_6 + x_7)\big) && \text{associativity} \\
&= \Sigma[0,3] + \big(\Sigma[4,5] + \Sigma[6,7]\big) && \text{siblings } x_6, x_7 \\
&= \Sigma[0,3] + \Sigma[4,7] && \text{siblings } \Sigma[4,5], \Sigma[6,7] \\
&= \Sigma[0,7] && \text{siblings } \Sigma[0,3], \Sigma[4,7].
\end{aligned}$$

The first transition cites the record of the child, the next two concatenate the two lists, and each of the last three replaces two siblings by their parent, one of the teal nodes in the figure. Here the siblings are always the last two entries of the list, so grouping them needs no further associativity. The record at the root of T is thus $\Sigma[0,7]$, the output of the balanced tree, which is the desired equality between the two trees.

Each list has $k = O(\log(M+1))$ entries, so a node needs $O(k)$ applications of associativity, taken from Lemma C.1. Each application changes the sum at one place in the list, and its equality is carried to the top through the $O(k)$ additions of the list by Lemma B.4, never through the possibly deep tree T. In the third line of the chain, for example, the change inside the parentheses is carried through the outer addition of $\Sigma[0,3]$. The parent of a node then cites the node's record instead of rederiving it, so intermediate rewrites are never propagated separately to the root of T. Each node therefore costs $O(k^2)W^{O(1)} = \log^2(M+1)W^{O(1)}$, and all nodes together cost $M\log^2(M+1)W^{O(1)}$, even when T is a chain of depth $M-1$. At the root of T the cover is the root of B, which proves $\sum_{j<M}^{[f,T]} x_j = \sum_{j<M} x_j$.

Permute the leaves of the balanced tree. Apply the same construction to T' on the permuted leaf list, which proves $\sum_{j<M}^{[f,T']} x_{\pi(j)} = \sum_{j<M} x_{\pi(j)}$. The two balanced sums are identified by the leaf-swap construction in the proof of Lemma D.6, which applies to a balanced tree on any number M of leaves and takes $O(M\log(M+1))$ blocks. Transitivity (55) joins the three equalities, which gives the statement when all leaves have scale f.

Reduction to a common scale. In general, put $y_j := \mathsf{Prmt}_f(x_j)$. The nodewise construction in the proof of Lemma D.5 applies to T and T' and proves

$$\sum_{j<M}^{T} x_j = \sum_{j<M}^{[f,T]} y_j, \qquad \sum_{j<M}^{T'} x_{\pi(j)} = \sum_{j<M}^{[f,T']} y_{\pi(j)}.$$

Both original roots already have scale f. These two comparisons cost $MW^{O(1)}$, including the missing promoted words, sum gates, and check gates. Each y_j has width at most w, so the case just proved applies to the promoted leaves, and transitivity gives the stated identity.

Widths. Both the original and promoted sums have width at most $w + M - 1$. The auxiliary block sums have depth $O(\log(M+1))$ above balanced subtrees, so their widths are $O(w+M) = O(W)$ as well. All additions in the promoted comparison trees have scale f. This wider allowance is used only in this lemma; Lemma D.6 retains its sharper width $w + O(\log m)$ for perfectly balanced sums. All missing adder and check gates are charged, including gatewise identification of separately constructed endpoint circuits. □

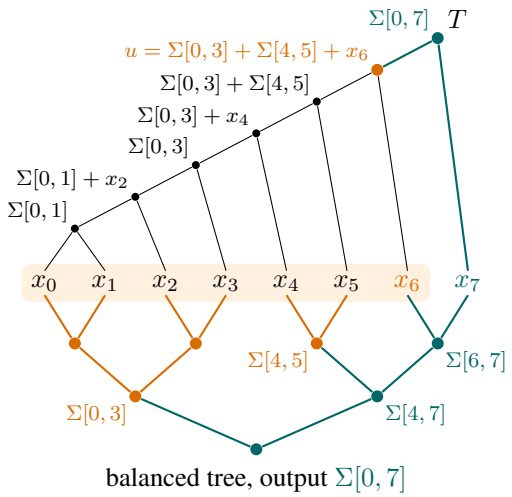


Figure 9: Comparing a sum of $M = 8$ terms with the balanced tree. The left-to-right tree T is drawn above the terms and the balanced tree below them, and $\Sigma[a, b]$ is the output of the subtree of the balanced tree on the positions $a, \dots, b$. Every node of T is labeled with its record, with parentheses omitted. The left child u of the root of T adds the terms in the shaded interval $\{0, \dots, 6\}$, and its record lists the three orange subtrees. The root appends x_7 and groups siblings at the three teal nodes of the balanced tree, which yields its record $\Sigma[0, 7]$.

Product circuits and notation. Write $\prod_{j<M}^{T} x_j$ for the circuit that multiplies the factors in index order, with the fixed binary parenthesization T. Every internal operation is the full-result complex multiplication circuit of Appendix A.3, using the omitted-scale convention: the operand scales determine the multiplier, and its output scale is their sum. The superscript T indicates the parenthesization. Without a parenthesization annotation, $\prod_{j<M} x_j$ uses the same balanced tree shape as $\sum_{j<M} x_j$, with multiplication at each internal node. The special superscript left specifies left-to-right accumulation:

$$\prod_{j<M}^{\mathrm{left}} x_j := \big(((x_0 x_1) x_2) \cdots x_{M-1}\big).$$

Each explicitly grouped subproduct is computed before it is used as a factor. A one-factor product returns that factor; the empty product is the scale-zero one. This indexed product notation is distinct from the fixed permutation circuit Π used in the routing lemmas.

If x_j has scale f_j and component width w_j, a subproduct with leaf set I has scale $\sum_{j\in I} f_j$ and component width $\sum_{j\in I} w_j + |I| - 1$. The inputs need not have the same scale or width. These rules determine the scale and width at every multiplication.

Lemma D.9 (Reordering an arbitrarily parenthesized product)**.** *Let $x_0, \dots, x_{M-1}$ be complex values of fixed scales f_j and component widths w_j, with $\sum_j f_j = O(W)$ and $\sum_j w_j + M = O(W)$. Fix binary parenthesizations T, T' and a permutation π of $[M]$. Then*

$$\Gamma \ \rhd_{\mathsf{EF}}^{M \log^2(M+1) W^{O(1)}} \ \prod_{j<M}^{T} x_j = \prod_{j<M}^{T'} x_{\pi(j)}.$$

Proof. Use the interval-cover and leaf-swap construction of Lemma D.8, replacing associativity and commutativity of addition by associativity and commutativity of multiplication from Lemma C.1. Replace

the additive identity zero by the multiplicative identity one. The factors retain their individual scales; the preliminary common-scale promotion used for sums is not needed for products.

More explicitly, the cover of an interval I in the balanced tree is the same fixed list of blocks. Its comparison circuit C_I now multiplies the block outputs. At a node, cite the two child equalities and propagate them through that node's multiplier. Associativity concatenates the two block products and exposes adjacent sibling blocks; their product is identified with the balanced tree's parent output. This again uses $O(\log^2(M+1))$ short-word blocks per original node. The balanced leaf-swap identity is $(xA)(yD) = (yA)(xD)$, proved by associativity and commutativity of multiplication, instead of the corresponding identity for addition. Its two paths and propagation to the balanced root cost $O(\log(M+1))$ blocks per transposition. These are precisely the counts in the sum proof, giving $M\log^2(M+1)W^{O(1)}$ overall.

The scale and width accounting uses the multiplication rules instead: each block has the sum of its leaf scales and the sum of its component widths plus one bit per multiplication. The stated assumptions keep all such values within $O(W)$, including the constant number of scale-zero identity factors used when exposing a leaf. A rebuilt path uses the scales of its new leaf set; all comparisons pad widths as needed. Operand circuits and proved node equalities are shared, and all missing multiplication and check gates are charged. The leaves may be outputs of selector circuits; the permutation and parenthesizations are fixed. □

D.3 Pairing Identities

For arrays $\boldsymbol{u} \in \mathcal{C}_f^m$ and $\boldsymbol{v} \in \mathcal{C}_g^m$, define the pairing by

$$\langle \boldsymbol{u}, \boldsymbol{v} \rangle_{[f+g]} := \sum_{j<m}^{[f+g]} u_j \cdot_{f,g} v_j.$$

Its product leaves already share scale $f+g$. More generally, a pairing annotated with scale f' requires every product leaf to have scale f'; any promotions needed to meet this requirement must be specified explicitly by the caller. An omitted pairing scale denotes the common scale of its product leaves, without inserting promotions.

Lemma D.10 (Common routing preserves pairings)**.** *Let $\boldsymbol{u} \in \mathcal{C}_f^m$ and $\boldsymbol{v} \in \mathcal{C}_g^m$, where $m = 2^s$, and let w bound their input component widths. Fix a permutation π of $[m]$, and let k be an s-bit shift word. Then*

$$\Gamma \ \rhd_{\mathsf{EF}}^{m\log^{O(1)} m\, w^{O(1)}}$$

$$\langle \Pi \boldsymbol{u}, \Pi \boldsymbol{v} \rangle_{[f+g]} = \langle \boldsymbol{u}, \boldsymbol{v} \rangle_{[f+g]}, \tag{102a}$$

$$\langle \mathsf{CShift}_k\, \mathsf{Reflect}\, \boldsymbol{u}, \mathsf{CShift}_k\, \mathsf{Reflect}\, \boldsymbol{v} \rangle_{[f+g]} = \langle \boldsymbol{u}, \boldsymbol{v} \rangle_{[f+g]}. \tag{102b}$$

Proof.

The pairing as a sum over the product array. Both identities compare two pairings, so we first record which circuit a pairing is. Each leaf $u_j \cdot_{f,g} v_j$ is a copy of the multiplier $\mathsf{Mul}_{f,g}$ and already has scale $f+g$, so the annotation in

$$\langle \boldsymbol{u}, \boldsymbol{v} \rangle = \langle \boldsymbol{u}, \boldsymbol{v} \rangle_{[f+g]} = \sum_{j<m}^{[f+g]} u_j \cdot_{f,g} v_j$$

inserts no promotion, and every addition of the sum tree is Add_{f+g}. A routing circuit acts on words of one fixed width, so the entries of $\boldsymbol{u}$ share a padded component width, as do those of $\boldsymbol{v}$; both are at most w. All m leaves are therefore copies of one multiplier: they share the scale $f+g$ and a common component width

$w' \le 2w + 1$. Full product and sum widths are retained, so the pairing is exact for arbitrary input entries. Write

$$\boldsymbol{P} := \big(u_j \cdot_{f,g} v_j\big)_{j<m}$$

for the array of these leaves. This names existing circuits and adds no gates, and it presents the right-hand side of both identities as the sum $\sum_{j<m}^{[f+g]} P_j$ of an array at the common scale $f + g$ and the common component width w', which is the setting of Lemmas D.6 and D.7.

Each identity is now proved in two steps: first that routing both factors in the same way produces the routed product array, coordinate by coordinate; then that routing does not change the sum of that array. Only the first step distinguishes the two identities, since Π is wiring while CShift_k is controlled by the word k. Each coordinate statement below concerns one multiplier and one word multiplexer, of size $w^{O(1)}$, so deriving one such statement per coordinate and stage stays inside the budget. No index-generalization is needed, and the coordinate statements are derived as explicit families over $j < m$. As elsewhere in this section, a word equality abbreviates the bitwise check $\mathsf{EQ} = 1$.

Proof of (102a). The circuit $(\Pi \boldsymbol{u})_j \cdot_{f,g} (\Pi \boldsymbol{v})_j$ is the multiplier $\mathsf{Mul}_{f,g}$ applied to the wires $u_{\pi(j)}$ and $v_{\pi(j)}$, which is the leaf $P_{\pi(j)} = (\Pi \boldsymbol{P})_j$. Permuting both factors by fixed wiring thus permutes the product array by the same wiring,

$$\big((\Pi \boldsymbol{u})_j \cdot_{f,g} (\Pi \boldsymbol{v})_j\big)_{j<m} = \Pi \boldsymbol{P},$$

as circuits; if the two sides were introduced as separately named copies, compare corresponding gates to identify them, in $O(m)$ blocks of size $w^{O(1)}$. The left-hand side of (102a) is therefore the sum $\sum_{j<m}^{[f+g]} (\Pi \boldsymbol{P})_j$, and Lemma D.6, applied to the array $\boldsymbol{P}$ and the permutation π, identifies this sum with $\sum_{j<m}^{[f+g]} P_j = \langle \boldsymbol{u}, \boldsymbol{v} \rangle_{[f+g]}$.

Pushing a product through one controlled stage. In (102a), the routing was performed through direct wiring. However, in (102b), the cyclic shift CShift_k is performed through a barrel shifter network consisting of $s = \log_2 m$ stages of word multiplexers, controlled by the bits of k. We first state the rule that exchanges one multiplexer with the multiplier, which one step of the induction below applies at each coordinate: for a Boolean control c and words x_1, x_0 and y_1, y_0 of the widths above,

$$\Gamma \;\rhd_{\mathsf{EF}}^{w^{O(1)}}\; \mathsf{Mux}(c, x_1, x_0) \cdot_{f,g} \mathsf{Mux}(c, y_1, y_0) = \mathsf{Mux}\big(c,\, x_1 \cdot_{f,g} y_1,\, x_0 \cdot_{f,g} y_0\big). \tag{103}$$

For complex entries, each multiplexer acts on the concatenated components and the equality is the componentwise check.

This rule is derived by proof by enumeration, Lemma B.5 (57), on the one-bit address c. Under each of its two values, the multiplexer identities (58) of Lemma B.6 replace all three multiplexers of (103) by their selected words; conditional equality propagation (51) carries the first two replacements through the multiplier that reads them, giving the selected product; and symmetry and transitivity (55) join that equality with the third replacement, which is the same selected product for the right-hand side. All circuits involved have $w^{O(1)}$ gates, so the two cases and their combination have size $w^{O(1)}$ in total.

Proof of (102b). Granted (103), the identity follows by an induction along the barrel stages, carried out on three routing networks at once. Write $\boldsymbol{u}^{(0)} := \mathsf{Reflect}\, \boldsymbol{u}$ and $u_j^{(r+1)} := \mathsf{Mux}\big(k_r, u^{(r)}_{j-2^r \bmod m}, u_j^{(r)}\big)$ for the barrel stages, so that $\boldsymbol{u}^{(s)} = \mathsf{CShift}_k\, \mathsf{Reflect}\, \boldsymbol{u}$, and define $\boldsymbol{v}^{(r)}$ and $\boldsymbol{p}^{(r)}$ by the same recursion from $\mathsf{Reflect}\, \boldsymbol{v}$ and $\mathsf{Reflect}\, \boldsymbol{P}$. The three barrel circuits have the same wiring pattern and read the same control bits

$k_0, \dots, k_{s-1}$; the first two occur in the conclusion, and the third is constructed and charged. Stage by stage, for $r = 0, \dots, s$, we derive the invariant

$$\big[u_j^{(r)} \cdot_{f,g} v_j^{(r)} = p_j^{(r)}\big]_{j<m}$$

saying that after r stages the products of the routed factors are the routed products.

At $r = 0$ the routing is wiring. The words $u_j^{(0)}$ and $v_j^{(0)}$ are $u_{-j \bmod m}$ and $v_{-j \bmod m}$, so their product is the leaf $P_{-j \bmod m} = p_j^{(0)}$, identified as in the previous part. For the step, fix $r < s$ and $j < m$ and put $j' := j - 2^r \bmod m$. Apply (103) with $c = k_r$ to the four words $u_{j'}^{(r)}$, $u_j^{(r)}$, $v_{j'}^{(r)}$, $v_j^{(r)}$:

$$u_j^{(r+1)} \cdot_{f,g} v_j^{(r+1)} = \mathsf{Mux}\big(k_r,\, u_{j'}^{(r)} \cdot_{f,g} v_{j'}^{(r)},\, u_j^{(r)} \cdot_{f,g} v_j^{(r)}\big).$$

The invariant at j' and at j identifies the two data inputs of this multiplexer with $p_{j'}^{(r)}$ and $p_j^{(r)}$, so equality propagation (50) through the multiplexer, applied to its two data words in turn, rewrites the right-hand side as $\mathsf{Mux}\big(k_r, p_{j'}^{(r)}, p_j^{(r)}\big)$, which is $p_j^{(r+1)}$. This is the invariant at j and stage $r+1$. At $r = s$ the invariant reads

$$\big[(\mathsf{CShift}_k\ \mathsf{Reflect}\ \boldsymbol{u})_j \cdot_{f,g} (\mathsf{CShift}_k\ \mathsf{Reflect}\ \boldsymbol{v})_j = (\mathsf{CShift}_k\ \mathsf{Reflect}\ \boldsymbol{P})_j\big]_{j<m}.$$

It remains to sum. These m coordinate equalities compare the leaves of the two sum trees below, so equality propagation (50) through the $m-1$ additions, nodewise from the leaves to the root as in the proof of Lemma D.5, gives

$$\langle \mathsf{CShift}_k\ \mathsf{Reflect}\ \boldsymbol{u}, \mathsf{CShift}_k\ \mathsf{Reflect}\ \boldsymbol{v}\rangle_{[f+g]} = \sum_{j<m}^{[f+g]} (\mathsf{CShift}_k\ \mathsf{Reflect}\ \boldsymbol{P})_j.$$

The array $\mathsf{Reflect}\ \boldsymbol{P}$ has the common scale $f+g$ and the common component width w', so Lemma D.7 identifies the right-hand side with $\sum_{j<m}^{[f+g]} (\mathsf{Reflect}\ \boldsymbol{P})_j$, and Lemma D.6, for the fixed permutation Reflect, identifies that in turn with $\sum_{j<m}^{[f+g]} P_j = \langle \boldsymbol{u}, \boldsymbol{v}\rangle_{[f+g]}$. Transitivity (55) joins the three equalities into (102b).

Cost. Each conclusion supplies its two pairings, hence the routed factor arrays and the leaves of their sum trees. Everything else is constructed and charged: the product arrays $\boldsymbol{P}$, $\Pi\boldsymbol{P}$ and $\mathsf{CShift}_k\ \mathsf{Reflect}\ \boldsymbol{P}$, the per-stage products $u_j^{(r)} \cdot_{f,g} v_j^{(r)}$, the intermediate sum trees, and the missing equality checks. There are $O(ms)$ such products and multiplexer words, each of width $O(w)$ and size $w^{O(1)}$, and one proof block per coordinate and stage, so the coordinate arguments cost $m \log^{O(1)} m\, w^{O(1)}$. Each leaf has component width $w' \le 2w+1$ and its sum tree adds s bits, so the propagation through the sum trees and the three applications of Lemmas D.6 and D.7 cost $m \log^{O(1)} m\, w^{O(1)}$ as well, including the proofs identifying separately named copies of a product array. □

Decoder vectors. For an s-bit address k and $m = 2^s$, define $(\boldsymbol{e}_k)_j := [k = j]$ for $j < m$. Each decoder bit is embedded in $\mathcal{C}_0$ with a two-bit signed real numerator and a zero imaginary numerator. For $\boldsymbol{v} \in \mathcal{C}_f^m$, every product in $\langle \boldsymbol{e}_k, \boldsymbol{v}\rangle_{[f]}$ uses $\mathsf{Mul}_{0,f}$ and has scale f; every sum-tree addition uses Add_f.

Lemma D.11 (Selection by a decoder pairing)**.** *Let $\boldsymbol{v} \in \mathcal{C}_f^m$ have common padded component width w, where $m = 2^s$, and let k be an s-bit address. Then*

$$\Gamma \ \rhd_{\mathsf{EF}}^{m \log^{O(1)} m\, w^{O(1)}} \ \langle \boldsymbol{e}_k, \boldsymbol{v}\rangle_{[f]} = \mathsf{Sel}(\boldsymbol{v}, k).$$

Proof. At every tree node, the conjunction of the address-prefix bits says whether its interval contains k. Outside that interval all decoder bits and all products with them are zero. Inside it, the next index bit chooses one child, and the other child's sum is zero. The multiplication-by-zero and multiplication-by-one inference rules identify the leaf products; the selector and zero-addition inference rules identify each parent sum with its selected leaf under the corresponding prefix condition. Compare words after sign extension when their widths differ. Unrolling this invariant proves the equality at the root. The full sum widths are $w + O(s)$, and the node derivations, including any missing auxiliary gates, cost $m \log^{O(1)} m\, w^{O(1)}$. This also justifies the impulse reductions in Appendix F. □

E EF Verification of FFT Constants

E.1 Twiddle Computation Circuit

Recall that in our FFT/IFFT implementations, the roots $z_1, \ldots, z_L$ and twiddle table $(q_m(j))_{j<m}$ are hardwired constants precomputed in Appendix A.4. However, to prove properties about these factors in EF, we introduce an auxiliary circuit whose output on address a is denoted by $\mathfrak{q}_m(a)$. We reserve $q_m(j)$ for hardwired table entries; $\mathfrak{q}_m(a)$ denotes a circuit output even when its address is fixed. Lemma E.1 derives in EF, for each fixed address j, that $a = j$ implies equality of the computed twiddle $\mathfrak{q}_m(a)$ with the hardwired coefficient $q_m(j)$.

The circuit evaluates the same product recipe used to precompute the hardwired twiddles in Appendix A.4. For $m = 2^s$ and address bits $a = \sum_{r<s} 2^r a_r$, the circuit $\mathfrak{q}_m(a)$ evaluates

$$\mathfrak{q}_m(a) := \mathsf{Prmt}_t\left(\prod_{r<s}^{\text{left}} \begin{cases} \mathsf{Prmt}_p(1), & a_r = 0, \\ z_{s-r}, & a_r = 1. \end{cases}\right).$$

Twiddle computation circuit: $\mathfrak{q}_m(a)$. For $m = 2^s$, this circuit computes the twiddle at a variable s-bit address a from the hardwired roots, using the product recipe in Appendix A.4. For $m = 1$, set $\mathfrak{q}_1(0) := \mathsf{Prmt}_t(1)$. A one-bit multiplexer chooses each factor between z_{s-r} and $\mathsf{Prmt}_p(1)$, so both alternatives have scale p. Addresses are reduced modulo m by retaining their low s bits. For $s \geq 1$, the first selected factor is the initial running product. For $2 \leq j \leq s$, the successive multiplication gates have input scales $(j-1)p, p$ and output scale jp, followed by promotion to t. These products retain full results, then sign-extend to the common twiddle width w_q. At a fixed exponent this is the same computation used to generate the corresponding hardwired twiddle; at a variable exponent it is an auxiliary circuit in the EF proof of Lemma E.3. Its size is polynomial in L and the word width.

Lemma E.1 (Hardwired twiddle certificates)**.** *For $m = 2^s \leq N$ and an s-bit address a, the twiddle computation circuit and the hardwired coefficients satisfy*

$$\Gamma \;\rhd^{m \log^{O(1)} N}_{\mathsf{EF}}\; \{[a = j] \Longrightarrow \mathfrak{q}_m(a) = q_m(j)\}_{j<m}.$$

Proof. For each fixed j, introduce the closed twiddle computation $\mathfrak{q}_m(j)$ and evaluate its gates to prove $\mathfrak{q}_m(j) = q_m(j)$, because this is the product recipe used to precompute that hardwired coefficient. The gate evaluations and equality check cost $\log^{O(1)} N$. Under the condition $a = j$, the conditional unfolding statement of Lemma B.3 gives equality of the corresponding address bits. The conditional equality-substitution statement of Lemma B.4 identifies $\mathfrak{q}_m(a)$ with the closed copy, gate by gate. Combine this with the evaluated output equality and retain the condition $a = j$ to obtain the stated implication. The conditional checks, proof auxiliaries, and small-step inferences also cost $\log^{O(1)} N$ per j.

Repeating for the m fixed addresses costs $m \log^{O(1)} N$ in total; all implications use the same variable-address circuit $\mathfrak{q}_m(a)$. Constructing that circuit and the conclusion's address, output-equality, and implication checks from scratch also fits this bound. The lengths of the tables for $m = 2, 4, \ldots, N$ sum to less than $2N$, so constructing and proving one such family per size costs $N \log^{O(1)} N$. For a common L-bit address, use its low s bits at size 2^s; recursive calls of that size share the computation and its conditional certificates. The closed computation certificates for each hardwired entry can also be reused.

□

E.2 Constant Certificates

For the error estimates, let

$$\varepsilon_{\mathrm{root}} = N^6 h, \quad \varepsilon_{\mathrm{tw}} = \varepsilon_{\mathrm{root}} N^{10}$$

where $h = 2^{-p}$ is the precomputation grid spacing from Appendix A.4. These error constants are used only in the proof. The named constants $\varepsilon_{\mathrm{root}}, \varepsilon_{\mathrm{tw}}$ belong to $\mathcal{D}_p$, since multiplication by their power-of-two integer factors shifts the numerator and leaves the scale unchanged. All constants in the estimates that follow are scaled real or complex numbers. An inequality containing only parameters is proved by evaluating its closed comparator circuit, gate by gate. The chosen p in particular gives

$$100 L \varepsilon_{\mathrm{root}} < 1, \qquad \varepsilon_{\mathrm{tw}} N^{60} < 2^{-20}.$$

The proof generator includes the evaluations of these two comparisons, together with the table construction.

Lemma E.2 (Hardwired root certificates)**.** *For the hardwired roots of Appendix A.4, EF derives the following assertions:*

$$\Gamma \ \rhd_{\mathsf{EF}}^{\log^{O(1)} N}$$

$$z_s \bar{z}_s \overset{\varepsilon_{\mathrm{root}}}{\approx} 1 \quad (1 \le s \le L), \tag{104a}$$

$$z_s^2 \overset{\varepsilon_{\mathrm{root}}}{\approx} z_{s-1} \qquad (2 \le s \le L). \tag{104b}$$

Proof.

Construction of the EF certificates. For each s, substitute the hardwired root bits into the displayed check circuits and evaluate their gates in order. Each gate value follows from its defining Boolean equivalence and the already evaluated inputs. The mathematical argument below shows that every check evaluates to one. Recording these gate evaluations therefore gives the required EF derivations. There are $O(L)$ checks, each with $O(p)$-bit working numerators and $\log^{O(1)} N$ gates. Their construction and all gate evaluations therefore cost $\log^{O(1)} N$, even when the checks must be built from scratch.

Mathematical justification outside EF. The following argument establishes that the hardwired checks evaluate to one for every input length. It justifies the proof generator and is not translated into small-step EF inferences. The initial roots are exact. If $z_s = a + \mathrm{i}b$ with $0 \le a, b \le 1$, the recipe gives $1/2 \le u \le 1$ and $0 \le v \le 1$. Put $e = u^2 - (1+a)/2$ and $r = 2uv - b$. The defining grid intervals (44) and (45) imply $-3h \le e \le 0$ and $-2h \le r \le 0$, and direct algebra gives

$$4u^2(u^2 - v^2 - a) = 1 - a^2 - b^2 + 4e + 4e^2 - 2br - r^2,$$
$$4u^2(u^2 + v^2 - 1) = a^2 + b^2 - 1 + 4ae + 4e^2 + 2br + r^2.$$

The extra terms in each line have magnitude at most $24h$, and $4u^2 \ge 1$. Consequently the norm defect increases by at most $24h$ at each step, and the complex square defect is at most the previous norm defect plus

$26h$: the first line bounds $|u^2 - v^2 - a|$ by the previous norm defect plus $24h$, and $|2uv - b| = |r| \le 2h$. Starting at $z_2 = \mathrm{i}$ gives norm defect at most $32(s-2)h$ for z_s; all displayed defects are bounded by $\varepsilon_{\mathrm{root}} = N^6 h$. Thus every closed check evaluates to one, as required by the gate-evaluation construction above. □

Address arithmetic. In the lemma below, operations on addresses supplied to $\mathfrak{q}_m$, such as $a+b$, $-a$, $2a$, and $a + m/2$, are performed modulo m. Since $m = 2^s$, use the integer arithmetic circuits and retain the low s bits of their results, including for signed negation. This reduction is wiring and needs no division circuit. In particular, doubling an address uses a left shift before retaining the low bits. The address of $\mathfrak{q}_{m/2}$ is similarly reduced modulo $m/2$ by retaining $s-1$ bits; for $m/2 = 1$, its only address is zero. Address products used later, such as aj, follow the same convention.

Lemma E.3 (Variable-address twiddle identities)**.** *For fixed $m = 2^s \le N$ with $1 \le s \le L$ and s-bit words a, b, the twiddle circuits satisfy*

$$\Gamma \ \triangleright_{\mathsf{EF}}^{\log^{O(1)} N}$$

$$|\mathfrak{q}_m(a)|_1 \le 2, \tag{105a}$$

$$\mathfrak{q}_m(a)\mathfrak{q}_m(b) \overset{\varepsilon_{\mathrm{tw}}}{\approx} \mathfrak{q}_m(a+b), \tag{105b}$$

$$\mathfrak{q}_m(a)\bar{\mathfrak{q}}_m(a) \overset{\varepsilon_{\mathrm{tw}}}{\approx} 1, \tag{105c}$$

$$\bar{\mathfrak{q}}_m(a) \overset{\varepsilon_{\mathrm{tw}}}{\approx} \mathfrak{q}_m(-a), \tag{105d}$$

$$[a < m/2] \implies \mathfrak{q}_m(a + m/2) = -\mathfrak{q}_m(a), \tag{105e}$$

$$\mathfrak{q}_m(2a) = \mathfrak{q}_{m/2}(a). \tag{105f}$$

The error checks use the common-scale promotion convention. In the product comparison $\mathfrak{q}_m(a)\mathfrak{q}_m(b) \overset{\varepsilon_{\mathrm{tw}}}{\approx} \mathfrak{q}_m(a+b)$, the product $\mathfrak{q}_m(a)\mathfrak{q}_m(b)$, single twiddle $\mathfrak{q}_m(a+b)$, and error bound $\varepsilon_{\mathrm{tw}}$ have scales $2t, t, p$, respectively. Thus the subtraction and inequality check are performed by promoting these quantities to the common scale $\max\{2t, p\}$. The proof of Lemma E.3 is deferred to Appendix E.3.

E.3 Proof of Lemma E.3

First derive the root certificates of Lemma E.2, at total additional cost $\log^{O(1)} N$, including any missing certificate circuits. Later steps reuse these assertions without copying their proofs. Put $c_r := z_{s-r}$ for $0 \le r < s$, and write c_r^e for the scale-p complex selector that outputs c_r when the bit e is one and $\mathsf{Prmt}_p(1)$ when it is zero, so that its value is the e-th power of c_r. Thus the already defined twiddle circuit is

$$\mathfrak{q}_m(a) := \mathsf{Prmt}_t\left(\prod_{r<s}^{\mathrm{left}} c_r^{a_r}\right).$$

The product starts from $c_0^{a_0}$ and uses the existing multipliers to include the remaining factors from left to right; its output before promotion has scale sp. In this notation, the root certificates (104a) and (104b) of Lemma E.2 give the first two assertions of

$$c_r\bar{c}_r \overset{\varepsilon_{\mathrm{root}}}{\approx} 1, \qquad c_r^2 \overset{\varepsilon_{\mathrm{root}}}{\approx} c_{r+1} \quad (r < s-1), \qquad c_{s-1} = -1,$$

the ranges matching under $c_r = z_{s-r}$, and the third is a closed check on z_1. The equality $c_{s-1} = -1$ and the comparisons $2L\varepsilon_{\mathrm{root}} \le 1$ and $0 \le \varepsilon_{\mathrm{root}} \le 1$ have only hardwired inputs. Construct their check circuits once and evaluate their gates to obtain EF assertions, at total additional cost $\log^{O(1)} N$.

For the cost accounting throughout the rest of this proof, recall that W is the common upper bound on numerator widths and scales introduced in Appendix A.5.4, with $W = O(L^4)$. Each application of Lemma C.1 costs $W^{O(1)}$ when its working widths and scales are $O(W)$. We also charge for any missing auxiliary circuits; the arguments below verify the required width and scale bounds for their constructions. For fixed power-of-two factors in the estimates, use Lemma C.1 (60) and Lemma B.4 when applying multiplication, order, or norm identities to their wiring implementations. These transfers are included in the $W^{O(1)}$ cost per short-word step.

The norm and conjugate-product bounds: $|\mathfrak{q}_m(a)|_1 \le 2$ and $\mathfrak{q}_m(a)\bar{\mathfrak{q}}_m(a) \overset{\varepsilon_{\mathrm{tw}}}{\approx} 1$. Each selected factor $c_j^{a_j}$ is either $\mathsf{Prmt}_p(1)$ or the root $c_j = z_{s-j}$. We bound the squared 2-norm of their running product, then use that one estimate for both conclusions.

Let u_j name the existing running-product outputs in $\mathfrak{q}_m(a)$:

$$u_0 := c_0^{a_0}, \qquad u_j := \mathsf{Mul}_{jp,p}\big(u_{j-1}, c_j^{a_j}\big) \quad (1 \le j < s).$$

Thus u_j has scale $(j+1)p$ and $\mathfrak{q}_m(a) = \mathsf{Prmt}_t(u_{s-1})$. These names introduce no gates. Upfront, construct the missing norm circuits

$$|c_j|_2^2, \qquad |c_j^{a_j}|_2^2, \qquad |u_j|_2^2 \quad (0 \le j < s),$$

and the final norms $|u_{s-1}|_1$, $|\mathfrak{q}_m(a)|_2^2$, the square $(|u_{s-1}|_1)^2$, and the real absolute values

$$\big||c_j|_2^2 - 1\big| \quad (j < s), \qquad \big||u_{s-1}|_2^2 - 1\big|.$$

Reuse $|u_0|_2^2$ as $|c_0^{a_0}|_2^2$. The two conclusion checks already supply $|\mathfrak{q}_m(a)|_1$ and $|\mathfrak{q}_m(a)\bar{\mathfrak{q}}_m(a) - 1|_1$. Every later occurrence of one of these expressions refers to the same output wires.

There are $O(s+1)$ new circuits here, each costing $W^{O(1)}$, for total construction cost $O(s+1)W^{O(1)}$. A squared norm uses two real multipliers and an adder; a complex 1-norm uses two absolute values and an adder. The roots and selected factors have $O(p)$-bit numerators; u_j has width $O((j+1)p + j + 1)$, and promotion to t gives width $O(t+s) = O(W)$. Squaring, adding, and taking these absolute differences keeps all widths and scales $O(W)$. The costs below cover the remaining arithmetic, checks, proof auxiliaries, and small-step inferences; the norm circuits just constructed are reused.

1. *Bound the squared norm of each factor.* For each $j < s$, we derive

$$1 - \varepsilon_{\mathrm{root}} \le |c_j|_2^2 \le 1 + \varepsilon_{\mathrm{root}}, \qquad 1 - \varepsilon_{\mathrm{root}} \le |c_j^{a_j}|_2^2 \le 1 + \varepsilon_{\mathrm{root}}.$$

   The shared certificate from Lemma E.2 (104a) gives $c_j \bar{c}_j \overset{\varepsilon_{\mathrm{root}}}{\approx} 1$. Apply the conjugate-product identity of Lemma C.1 (78d), substitute through the error check using Lemma B.4 (52), and extract the real bound using Lemma C.1 (82) on the length-one difference. This gives $\big||c_j|_2^2 - 1\big| \le \varepsilon_{\mathrm{root}}$. To obtain the interval, split on the sign bit of $|c_j|_2^2 - 1$ in its absolute-value circuit, which returns the difference or its negation. The addition identities of Lemma C.1 (62b), (62c), (62e), and order preservation (71) translate the two cases into the stated lower and upper bounds.

   For the selected factor, Lemma B.6 gives $c_j^{a_j} = \mathsf{Prmt}_p(1)$ under $a_j = 0$ and $c_j^{a_j} = c_j$ under $a_j = 1$. The first case uses the closed checks for the squared norm of one and $1 - \varepsilon_{\mathrm{root}} \le 1 \le 1 + \varepsilon_{\mathrm{root}}$; the second uses the root interval. Conditional substitution by Lemma B.4 (53) transfers these bounds to the selected factor, and Lemma B.5 (57) combines the two cases for each bound. For each j, there are a constant number of short-word lemma applications and two selector cases. Their missing checks, auxiliary gates, and small-step inferences cost $W^{O(1)}$, hence $O(s)W^{O(1)}$ in total. The root certificates and all norm outputs are reused.

2. *Propagate the bounds through the running products.* We derive, successively for $0 \le j < s$,

$$1 - 2(j+1)\varepsilon_{\mathrm{root}} \le |u_j|_2^2 \le 1 + 2(j+1)\varepsilon_{\mathrm{root}}.$$

For $j = 0$, use $u_0 = c_0^{a_0}$, Step 1, and the closed comparisons $1 - 2\varepsilon_{\mathrm{root}} \le 1 - \varepsilon_{\mathrm{root}}$ and $1 + \varepsilon_{\mathrm{root}} \le 1 + 2\varepsilon_{\mathrm{root}}$. Order transitivity, Lemma C.1 (68), gives the base case.

For $1 \le j < s$, evaluate the closed checks $0 \le 1 - 2j\varepsilon_{\mathrm{root}}$ and $0 \le 1 - \varepsilon_{\mathrm{root}}$. Together with the preceding intervals, these give nonnegative operands for multiplication. The lower and upper bounds follow from

$$\begin{aligned}
1 - 2(j+1)\varepsilon_{\mathrm{root}} &\le (1 - 2j\varepsilon_{\mathrm{root}})(1 - \varepsilon_{\mathrm{root}}) && \text{closed parameter check} \\
&\le |u_{j-1}|_2^2 \, |c_j^{a_j}|_2^2 && \text{Lemma C.1 (74)} \\
&= |u_j|_2^2 && \text{Lemma C.1 (78f),}
\end{aligned}$$

$$\begin{aligned}
|u_j|_2^2 &= |u_{j-1}|_2^2 \, |c_j^{a_j}|_2^2 && \text{Lemma C.1 (78f)} \\
&\le (1 + 2j\varepsilon_{\mathrm{root}})(1 + \varepsilon_{\mathrm{root}}) && \text{Lemma C.1 (74)} \\
&\le 1 + 2(j+1)\varepsilon_{\mathrm{root}} && \text{closed parameter check.}
\end{aligned}$$

Each product inequality uses order preservation twice. For multiplication on the left, use commutativity (63a) of Lemma C.1 and substitution (52) of Lemma B.4. Order transitivity (68) combines the comparisons. The closed upper comparison holds because $2j\varepsilon_{\mathrm{root}} \le 2L\varepsilon_{\mathrm{root}} \le 1$; its cross term $2j\varepsilon_{\mathrm{root}}^2$ is therefore at most $\varepsilon_{\mathrm{root}}$. The closed lower comparison follows by expanding its product, whose cross term is nonnegative. Both checks are certified by gate evaluation.

At the last prefix, reverse the two-sign-case conversion in Step 1 and evaluate $1 + 2s\varepsilon_{\mathrm{root}} \le 2$. Using the same addition identities and Lemma C.1 (71), (68), this gives

$$\big||u_{s-1}|_2^2 - 1\big| \le 2s\varepsilon_{\mathrm{root}}, \qquad |u_{s-1}|_2^2 \le 2.$$

The new circuitry consists of the products of squared norms, parameter expressions, and missing comparison and equality checks in these chains. Each stage uses a constant number of $W^{O(1)}$-cost constructions and lemma applications, including common-scale alignment. The preceding bounds and the norm outputs are reused, so all stages and the final conversion cost $O(s)W^{O(1)}$.

3. *Derive the* 1*-norm bound.* The final squared-norm estimate gives

$$\begin{aligned}
(|u_{s-1}|_1)^2 &\le 2|u_{s-1}|_2^2 && \text{Lemma C.1 (78e)} \\
&\le 4 && \text{Step 2; Lemma C.1 (74); evaluate } 2 \cdot 2 = 4.
\end{aligned}$$

To exclude the opposite comparator case $2 < |u_{s-1}|_1$, apply strict multiplication monotonicity, Lemma C.1 (75), twice, using commutativity (63a) to obtain

$$4 = 2 \cdot 2 < 2|u_{s-1}|_1 < (|u_{s-1}|_1)^2.$$

Strict/non-strict transitivity (69) contradicts the preceding bound by four. The comparator definitions in Appendix C.1 therefore give $|u_{s-1}|_1 \le 2$. Read this bound as $u_{s-1} \overset{2}{\approx} 0$, using the addition-by-zero identity (62c) of Lemma C.1. The error-check promotion equivalence (88) then transfers it through the existing final promotion to $|\mathfrak{q}_m(a)|_1 \le 2$. Check substitutions use Lemma B.4 (52). The norms and their square were constructed upfront; the remaining products, checks, and constant number of lemma applications cost $W^{O(1)}$.

4. *Derive the conjugate-product bound.* Use the same final squared-norm estimate and the exact identity

$$\begin{aligned} \mathfrak{q}_m(a)\bar{\mathfrak{q}}_m(a) &= |\mathfrak{q}_m(a)|_2^2 + \mathrm{i}0 && \text{Lemma C.1 (78d)} \\ &= \mathsf{Prmt}_{2t}\big(|u_{s-1}|_2^2\big) + \mathrm{i}0 && \text{Lemma C.1 (83)–(86); Lemma B.4 (50)}. \end{aligned}$$

The second equality moves the component promotions through the two squares and their sum. Promote Step 2's check $\big||u_{s-1}|_2^2 - 1\big| \le 2s\varepsilon_{\mathrm{root}}$ to scale $2t$ using Lemma C.1 (88). Apply the real-to-complex bound conversion (81) to the length-one difference, and substitute the displayed identity using Lemma B.4 (52). Finally evaluate $2s\varepsilon_{\mathrm{root}} \le \varepsilon_{\mathrm{tw}}$ and use order transitivity, Lemma C.1 (68), to obtain

$$|\mathfrak{q}_m(a)\bar{\mathfrak{q}}_m(a) - 1|_1 \le 2s\varepsilon_{\mathrm{root}} \le \varepsilon_{\mathrm{tw}}. \tag{106}$$

Retain the stronger $2s\varepsilon_{\mathrm{root}}$ bound for the conjugation argument. The product on the left is supplied by the conclusion, and both squared norms are already available. The missing promotions, intermediate arithmetic, equality checks, and constant number of lemma applications cost $W^{O(1)}$; their widths and scales remain $O(W)$.

Including the upfront norm circuits, the shared additional cost for the two conclusions is $O(s+1)W^{O(1)}$. This excludes the root certificates and initial parameter checks already charged at the start of the proof. Including that one-time cost gives $\log^{O(1)} N$, since $s \le L$ and $W = \log^{O(1)} N$.

The same argument also gives the factor bounds needed in the product law. For arbitrary bits $e_0, \dots, e_{s-1}$, with the shared certificates and parameter checks available, we have

$$\Gamma \ \rhd_{\mathsf{EF}}^{O(s)W^{O(1)}} \ \big[\, |c_r|_1 \le 2, \quad |c_r^{e_r}|_1 \le 2 \,\big]_{r<s}. \tag{107}$$

Step 1 gives $|c_r|_2^2 \le 1 + \varepsilon_{\mathrm{root}} \le 2$, with the last comparison certified by closed evaluation and Lemma C.1 (68). Repeat Step 3's conversion, using Lemma C.1 (78e), (74), (75), (63a), and (69), to obtain $|c_r|_1 \le 2$. The selector identities of Lemma B.6, conditional substitution (53) of Lemma B.4, and case combination (57) of Lemma B.5 give $|c_r^{e_r}|_1 \le 2$. The new 1-norm and selector circuits, checks, and these derivations cost $W^{O(1)}$ per r, hence $O(s)W^{O(1)}$ in total. The root squared norms and certificates are reused.

The product law: $\mathfrak{q}_m(a)\mathfrak{q}_m(b) \overset{\varepsilon_{\mathrm{tw}}}{\approx} \mathfrak{q}_m(a+b)$. Multiplying the two twiddles combines their selected root factors. We process one address position at a time, replacing two copies of a root by the root at the next position, as prescribed by binary addition. Each replacement has a small error; the factor bounds proved above control its effect on the whole product.

Observe that the expanded context already contains the outputs

$$\mathfrak{q}_m(a), \qquad \mathfrak{q}_m(b), \qquad \mathfrak{q}_m(a+b), \qquad \mathfrak{q}_m(a)\mathfrak{q}_m(b),$$

and the norm check on $\mathfrak{q}_m(a)\mathfrak{q}_m(b) - \mathfrak{q}_m(a+b)$ in the conclusion. Inside these twiddle circuits, it also contains the selected factors and every left-to-right prefix product

$$c_j^{x_j}, \qquad \prod_{0 \le k \le j}^{\mathrm{left}} c_k^{x_k} \quad (0 \le j < s,\ x \in \{a, b, a+b\}).$$

The prefix product has scale $(j+1)p$; its final promotion to t supplies $\mathfrak{q}_m(x)$. The modular sum $a+b$ and its carry bits are also already available. We reuse these outputs below without giving them new names. An underbrace "in context" marks existing output wires, including auxiliary outputs already constructed in the proof. Their construction cost is included in the step that introduces them. The superscript left specifies left-to-right accumulation. Products of the displayed blocks are also left-associated, with each displayed input pair multiplied first; an empty product is the scale-zero constant one.

1. *Move the final promotions outside the product.* The unpromoted products inside $\mathfrak{q}_m(a)$ and $\mathfrak{q}_m(b)$ both have scale sp. The first- and second-operand multiplication-promotion identities and composition of promotions from Lemma C.1 give

$$\begin{aligned}\underbrace{\mathfrak{q}_m(a)\mathfrak{q}_m(b)}_{\text{in context}} &= \mathsf{Prmt}_{2t}\left(\underbrace{\prod_{j<s}^{\text{left}} c_j^{a_j}}_{\text{in context}}\ \underbrace{\mathsf{Prmt}_t\left(\prod_{j<s}^{\text{left}} c_j^{b_j}\right)}_{\text{in context}}\right)\\ &= \mathsf{Prmt}_{2t}\left(\mathsf{Prmt}_{sp+t}\left(\underbrace{\left(\prod_{j<s}^{\text{left}} c_j^{a_j}\right)\left(\prod_{j<s}^{\text{left}} c_j^{b_j}\right)}_{\text{new }(\star)}\right)\right)\\ &= \mathsf{Prmt}_{2t}(\star)\,.\end{aligned}$$

   The first equality applies the first-operand promotion identity; the second applies the second-operand identity; the third composes promotions. Propagating these equalities uses Lemma B.4. Constructing the missing products, promotions, and equality checks costs $W^{O(1)}$. The constant number of lemma applications adds $W^{O(1)}$ for auxiliary gates and small-step inferences.

2. *Interleave the two factor lists.* We want to prove

$$\underbrace{\mathfrak{q}_m(a)\mathfrak{q}_m(b)}_{\text{in context}} = \mathsf{Prmt}_{2t}\left(\underbrace{\prod_{j<s}^{\text{left}}\left(\underbrace{c_j^{a_j}}_{\text{in context}}\ \underbrace{c_j^{b_j}}_{\text{in context}}\right)}_{\text{new }(\square)}\right). \tag{108}$$

   Keep every factor slot, including those whose selector outputs one. The permutation that interleaves the two lists is then fixed, independently of a, b. Apply Lemma D.9 to derive

$$\underbrace{\left(\prod_{j<s}^{\text{left}} c_j^{a_j}\right)\left(\prod_{j<s}^{\text{left}} c_j^{b_j}\right)}_{(\star)} = \underbrace{\prod_{j<s}^{\text{left}}\left(c_j^{a_j} c_j^{b_j}\right)}_{(\square)}.$$

   The left-hand side is the subcircuit marked $(\star)$ inside $\mathsf{Prmt}_{2t}(\cdot)$ in the last expression of the preceding step; both sides have scale $2sp$. Apply Lemma B.4 to propagate this equality through the outer promotion Prmt_{2t}. Combining the resulting equality with the preceding step's equality proves (108).

   Constructing $(\square)$, including its internal pair and prefix outputs, uses s pair multipliers and $s-1$ accumulating multipliers and costs $O(s)W^{O(1)}$; promotion uses wiring. Each prefix contains at most $2s$ scale-p factors, giving $O(sp+s) = O(W)$ widths before promotion and $O(t+s) = O(W)$ afterwards. Constructing the missing equality checks costs $W^{O(1)}$. The regrouping lemma, including its proof auxiliaries, costs $O(s\log^2(s+1))W^{O(1)}$. Equality substitution through the promotion and combination with the preceding equality add $W^{O(1)}$. Including the $O(s)W^{O(1)}$ paired-product construction, the total additional cost of this step is $O(s\log^2(s+1))W^{O(1)}$.

3. *Prove the sum-and-carry relations.* Write γ_r for the existing carry bits of the modular adder, with $\gamma_0 = 0$; its sum bits are $(a+b)_r$. The full-adder derivation in the proof of Lemma C.1 gives, for every $0 \le r < s$,

$$a_r + b_r + \gamma_r = \big(\underbrace{a+b}_{\text{in context}}\big)_r + 2\gamma_{r+1}.$$

Here the bits are interpreted as small unsigned integers. Each assertion uses only a constant number of bits, so its check and derivation cost $O(1)$, hence $O(s)$ for all positions. The low s sum bits represent $a + b \bmod m$; the final carry γ_s is discarded by the address wiring.

4. *Construct suffix products and carry-stage products.* For $1 \le r < s$, construct the suffix product

$$\underbrace{\prod_{r \le j < s}^{\text{left}} \underbrace{\big(c_j^{a_j} c_j^{b_j}\big)}_{\text{from } (\square)}}_{\text{new } \lozenge_r},$$

accumulating the existing pairs from left to right starting at pair r. At the endpoints, $\lozenge_0$ is the product $(\square)$ constructed in Step 2, and $\lozenge_s$ is the empty product, the scale-zero constant one. The suffix starting at r has scale $2(s-r)p$. Only the suffixes with $1 \le r < s$ need new multipliers: there are $O(s^2)$ such multipliers in total, costing $O(s^2)W^{O(1)}$. Supplying the empty suffix requires no arithmetic gates.

Construct the carry selectors $c_r^{\gamma_r}$ at cost $O(s)W^{O(1)}$. Let $f := 2t + p$ and construct the carry-stage outputs

$$X_0 := \mathsf{Prmt}_f\left(\underbrace{c_0^{\gamma_0}}_{\text{in context}} \underbrace{\prod_{j<s}^{\text{left}} \big(c_j^{a_j} c_j^{b_j}\big)}_{\lozenge_0}\right),$$

$$X_r := \mathsf{Prmt}_f\left(\left(\underbrace{\prod_{j<r}^{\text{left}} c_j^{(a+b)_j}}_{\text{in context}} \underbrace{c_r^{\gamma_r}}_{\text{in context}}\right) \underbrace{\prod_{r \le j < s}^{\text{left}} \big(c_j^{a_j} c_j^{b_j}\big)}_{\lozenge_r}\right) \quad (1 \le r < s),$$

$$X_s := \mathsf{Prmt}_f\left(\underbrace{\prod_{j<s}^{\text{left}} c_j^{(a+b)_j}}_{\text{in context}}\right).$$

Each intermediate stage combines the processed sum-bit prefix, one incoming-carry factor, and the remaining input-pair suffix. These $O(s)$ new multipliers cost $O(s)W^{O(1)}$. Before promotion, the product at stage $r < s$ has scale $(2s - r + 1)p$; the final stage reuses the scale-sp product inside $\mathfrak{q}_m(a+b)$. These promotions append low zero bits by wiring. Each product contains at most $2s+1$ scale-p factors, so its width is $O(sp + s)$ before promotion and $O(t + s) = O(W)$ afterwards. The total new-circuit cost of this step is $O(s^2)W^{O(1)}$; the previously constructed factors and products are reused.

5. *Prove the endpoint equalities.* We will derive in EF

$$X_0 = \mathsf{Prmt}_f\left(\underbrace{\mathfrak{q}_m(a)\mathfrak{q}_m(b)}_{\text{in context}}\right), \qquad X_s = \mathsf{Prmt}_f\left(\underbrace{\mathfrak{q}_m(a+b)}_{\text{in context}}\right).$$

All product outputs in these assertions are now available. Supply the right-hand promotion wiring and construct the two equality checks, after sign extension to matching widths, at cost $W^{O(1)}$. For the first endpoint, $\gamma_0 = 0$ gives $c_0^{\gamma_0} = \mathsf{Prmt}_p(1)$. Multiplication by one and the promotion identities of Lemma C.1, followed by equality substitution from Lemma B.4, therefore identify X_0 with

$$\mathsf{Prmt}_f\left(\underbrace{\prod_{j<s}^{\text{left}}\left(c_j^{a_j}c_j^{b_j}\right)}_{\Diamond_0}\right).$$

Cite (108) and compose promotions to obtain $X_0 = \mathsf{Prmt}_f(\mathfrak{q}_m(a)\mathfrak{q}_m(b))$. For the second endpoint, X_s and $\mathfrak{q}_m(a+b)$ promote the same existing prefix-product output to f and t, respectively, so composition of promotions gives the result. These checks and the constant number of short-word lemma applications cost $W^{O(1)}$ in total. Equation (108) is cited at its existing wires; its product circuit and derivation are not charged again.

6. *Prove the error bound for one local carry replacement.* For each $0 \le r < s$, construct the three-factor block

$$C_r := \underbrace{c_r^{\gamma_r}}_{\text{in context}}\ \underbrace{\left(c_r^{a_r}c_r^{b_r}\right)}_{\text{from }(\square)}$$

at scale $3p$, reusing the pair output constructed in Step 2 (interleaving). For $r < s-1$, construct the replacement

$$C'_r := \underbrace{c_r^{(a+b)_r}}_{\text{in context}}\ \underbrace{c_{r+1}^{\gamma_{r+1}}}_{\text{in context}}$$

at scale $2p$. At the last position, define

$$C'_{s-1} := \underbrace{c_{s-1}^{(a+b)_{s-1}}}_{\text{in context}}$$

at scale p, reusing the selector output already supplied by $\mathfrak{q}_m(a+b)$. For every $0 \le r < s$, construct the error circuit

$$E_r := C_r -_{3p} \mathsf{Prmt}_{3p}(C'_r).$$

We will derive in EF

$$|E_r|_1 \le \mathsf{Prmt}_{3p}(2\varepsilon_{\text{root}}) \quad (0 \le r < s), \qquad E_{s-1} = \mathsf{Prmt}_{3p}(0).$$

Thus every local replacement has error at most $2\varepsilon_{\text{root}}$, and the last replacement is exact.

At each position, the three case bits a_r, b_r, γ_r have eight assignments. Their small integer sum $n_r := a_r + b_r + \gamma_r$ takes four possible values. For $r < s-1$, under each bit assignment, the previously proved adder assertion and the selector gates identify the block outputs with the following values:

n_r	0	1	2	3
C_r	1	c_r	c_r^2	c_r^3
C'_r	1	c_r	c_{r+1}	$c_r c_{r+1}$.

The table records represented values, compared after promotion to scale $3p$. For $n_r = 0, 1$, multiplication by one from Lemma C.1 gives exact equality. For $n_r = 2$, use the root-square certificate Lemma E.2 (104b), which in this notation is $|c_r^2 - c_{r+1}|_1 \le \varepsilon_{\mathrm{root}}$. For $n_r = 3$, subtraction distributivity (65) and multiplication associativity (63b) from Lemma C.1 give

$$c_r^3 - c_r c_{r+1} = c_r(c_r^2 - c_{r+1}).$$

The product bound $|xy|_1 \le |x|_1 |y|_1$ in "Complex norm identities and bounds" of Lemma C.1, together with order preservation and $|c_r|_1 \le 2$, bounds $|c_r^3 - c_r c_{r+1}|_1$ by $2\varepsilon_{\mathrm{root}}$. Promotion of multiplication and error checks from Lemma C.1, followed by equality substitution from Lemma B.4, transfers these bounds to $|E_r|_1$. Order preservation and $\varepsilon_{\mathrm{root}} \ge 0$ put the zero and $\varepsilon_{\mathrm{root}}$ bounds from the other cases under the same bound $2\varepsilon_{\mathrm{root}}$.

For $r = s - 1$, the root certificate gives $c_r = -1$, and the adder assertion gives $n_r = (a + b)_r + 2\gamma_s$. In each of the eight bit cases, identify the selected factors with 1 or -1 and evaluate the resulting closed products. This gives the same sign for C_r and C'_r: two copies of -1 multiply to one. The promotion and additive-inverse identities therefore give $E_{s-1} = \mathsf{Prmt}_{3p}(0)$. This also implies its norm bound, using $\varepsilon_{\mathrm{root}} \ge 0$. Thus the outgoing carry disappears, matching the discarded γ_s in addition modulo m. At every position, apply Lemma B.5 to the three-bit address (a_r, b_r, γ_r) to combine the eight conditional conclusions. This adds constant cost per stated assertion.

All the new blocks, error circuits, and checks have $O(W)$ working widths and scales. Their construction costs $W^{O(1)}$ per position. Each of the eight cases uses a constant number of applications of Lemmas B.4 and C.1; their proof auxiliaries, small-step inferences, and the combination of cases add $W^{O(1)}$ per position. The total additional cost is therefore $O(s)W^{O(1)}$. The existing selector and pair outputs are shared, and the previously proved root-square certificates and root norm bounds are cited without copying their proofs.

7. *Bound the effect of a local replacement on the whole product.* The surrounding products are the existing sum-bit prefix through position $r - 1$ and input-pair suffix strictly after position r. Their scales are rp and $2(s - r - 1)p$, respectively. To bound them, apply (107) to the three bit arrays $(a_j)_{j<s}$, $(b_j)_{j<s}$, and $((a + b)_j)_{j<s}$ to derive in EF, for each $0 \le j < s$,

$$|c_j^{a_j}|_1,\ |c_j^{b_j}|_1,\ |c_j^{(a+b)_j}|_1 \le 2.$$

These three applications cost $O(s)W^{O(1)}$ in total, including all missing norm and comparison circuits, proof auxiliaries, and small-step inferences, reusing the root certificates and parameter checks. Apply the product bound for the complex 1-norm and order preservation from Lemma C.1 once at each multiplication node, using these bounds, to derive

$$\left| \underbrace{\prod_{j<r}^{\text{left}} c_j^{(a+b)_j}}_{\text{in context}} \right|_1 \le 2^r, \qquad \left| \underbrace{\prod_{r<j<s}^{\text{left}} \left(c_j^{a_j} c_j^{b_j}\right)}_{\Diamond_{r+1}} \right|_1 \le 2^{2(s-r-1)}.$$

There are $O(s)$ nodes per position, so the new norm and comparison checks and their derivations cost $O(s^2)W^{O(1)}$ over all r.

For each $0 \le r < s$, define

$$U_r := \underbrace{\left(\prod_{j<r}^{\text{left}} c_j^{(a+b)_j}\right)}_{\text{in context}} \underbrace{C_r}_{\text{in context}} \underbrace{\left(\prod_{r<j<s}^{\text{left}} \left(c_j^{a_j} c_j^{b_j}\right)\right)}_{\Diamond_{r+1}},$$

$$U'_r := \underbrace{\left(\prod_{j<r}^{\text{left}} c_j^{(a+b)_j}\right)}_{\text{in context}} \underbrace{\mathsf{Prmt}_{3p}(C'_r)}_{\text{in context}} \underbrace{\left(\prod_{r<j<s}^{\text{left}} \left(c_j^{a_j} c_j^{b_j}\right)\right)}_{\Diamond_{r+1}}.$$

We prove in EF that

$$X_r = \mathsf{Prmt}_f(U_r), \qquad X_{r+1} = \mathsf{Prmt}_f(U'_r).$$

Apply Lemma D.9 to regroup the factors of X_r and X_{r+1} into the displayed blocks. Multiplication by one handles the empty prefix or suffix. The promotion identities of Lemma C.1 move the promotion in $\mathsf{Prmt}_{3p}(C'_r)$ outside the product and compose it with the final promotion to f, giving the second equality. Each regrouping has $O(s)$ factors, so Lemma D.9 gives cost $O(s \log^2(s+1))W^{O(1)}$, including its missing product and check gates. The bound includes propagation through the product; unaffected factors and previously derived equalities are shared. On these existing operands, the two distributivity statements for subtraction (65) and (67), with multiplication associativity (63b), from Lemma C.1 derive

$$U_r - U'_r = \underbrace{\left(\prod_{j<r}^{\text{left}} c_j^{(a+b)_j}\right)}_{\text{in context}} \underbrace{E_r}_{\text{in context}} \underbrace{\left(\prod_{r<j<s}^{\text{left}} \left(c_j^{a_j} c_j^{b_j}\right)\right)}_{\Diamond_{r+1}}.$$

Every term here has scale $(2s - r + 1)p \le f$. The product bound for the complex 1-norm, order preservation, and promotion of the error check from Lemma C.1, using the prefix and suffix bounds and the bound on E_r, then give

$$\underbrace{|X_r -_f X_{r+1}|_1}_{\text{new } d_r} \le \mathsf{Prmt}_f\left(2^r\, 2\varepsilon_{\text{root}}\, 2^{2(s-r-1)}\right) \le \mathsf{Prmt}_f\left(2^{2s}\varepsilon_{\text{root}}\right).$$

The last coefficient comparison is a closed check, certified by gate evaluation. Equality substitution through the error checks uses Lemma B.4. After the regroupings, only a constant number of short-word applications and new arithmetic checks is needed per r, at cost $W^{O(1)}$. Thus this whole step costs $O(s^2 \log^2(s+1))W^{O(1)}$.

8. *Accumulate the errors across carry positions.* Starting with $r = 0$, derive the invariant

$$\underbrace{|X_0 -_f X_r|_1}_{\text{new } D_r} \le \mathsf{Prmt}_f\left(r 2^{2s}\varepsilon_{\text{root}}\right) \qquad (0 \le r \le s).$$

The base case follows from the additive-inverse and norm statements of Lemma C.1, at cost $W^{O(1)}$ including its check circuit. For each transition, the addition identities and triangle inequality of Lemma C.1 give

$$\underbrace{|X_0 -_f X_{r+1}|_1}_{D_{r+1}} \le \underbrace{|X_0 -_f X_r|_1}_{D_r} +_f \underbrace{|X_r -_f X_{r+1}|_1}_{d_r}.$$

The rule for adding two inequalities in Lemma C.1 (73) combines the preceding invariant with the local error bound; (68) then bounds D_{r+1}. The sum of their bounds equals $\mathsf{Prmt}_f((r+1)2^{2s}\varepsilon_{\mathrm{root}})$; construct and evaluate this closed check for each fixed r. All missing differences, norms, adders, and comparison checks and the constant number of lemma applications cost $W^{O(1)}$ per transition. Each transition cites the preceding invariant without copying its derivation. The base case and all transitions therefore cost $O(s+1)W^{O(1)}$.

9. *Recover the product-law conclusion.* Put $\beta := s2^{2s}\varepsilon_{\mathrm{root}}$, represented at scale p. At $r = s$, substitute the endpoint equalities into the invariant using Lemma B.4 to obtain

$$\mathfrak{q}_m(a)\mathfrak{q}_m(b) \overset{\beta}{\approx}_f \mathfrak{q}_m(a+b).$$

Promotion of addition, composition of promotions, and the error-check equivalence from Lemma C.1 transfer the resulting bound from scale f to the comparison scale $\max\{2t, p\}$, giving

$$\underbrace{\mathfrak{q}_m(a)\mathfrak{q}_m(b)}_{\text{in context}} \overset{\beta}{\approx} \underbrace{\mathfrak{q}_m(a+b)}_{\text{in context}}. \tag{109}$$

Construct and evaluate the closed comparisons $\beta \le \varepsilon_{\mathrm{root}}N^4 \le \varepsilon_{\mathrm{tw}}$, then use order transitivity from Lemma C.1 (68) to obtain the stated product bound. This final step costs $W^{O(1)}$, including the missing bound and comparison circuits. Retain the stronger bound β for the conjugation argument below.

The total additional cost of the product-law construction is $O(s^2\log^2(s+1))W^{O(1)} = \log^{O(1)} N$, beyond the already charged root certificates and root norm bounds.

The conjugation bound: $\bar{\mathfrak{q}}_m(a) \overset{\varepsilon_{\mathrm{tw}}}{\approx} \mathfrak{q}_m(-a)$. Write $b := -a \bmod m$ for the address already computed in the conclusion. The expanded context supplies $\mathfrak{q}_m(a)$, $\bar{\mathfrak{q}}_m(a)$, $\mathfrak{q}_m(b) = \mathfrak{q}_m(-a)$, and the target error check. Reuse also the product $\mathfrak{q}_m(a)\bar{\mathfrak{q}}_m(a)$, its stronger error bound (106), and the bound $|\mathfrak{q}_m(a)|_1 \le 2$ from the joint norm proof.

1. *Apply the product law to opposite addresses.* Recall $\beta = s2^{2s}\varepsilon_{\mathrm{root}}$ from the product-law proof. We derive

$$\begin{aligned}
\mathfrak{q}_m(a)\mathfrak{q}_m(b) &\overset{\beta}{\approx} \mathfrak{q}_m(a+b) && \text{(109)}\\
&= \mathfrak{q}_m(0) && \text{Lemma C.1 (92), (62e); Lemma B.4}\\
&= 1 && \text{zero-address selectors; Lemma C.1: multiplication and promotion identities}.
\end{aligned}$$

For the first equality, the reduction-before-addition rule gives $(a+b) \bmod m = (a-a) \bmod m = 0$. For the second, every selector in $\mathfrak{q}_m(0)$ outputs $\mathsf{Prmt}_p(1)$. Multiplication by one and promotion, using (63c) and (83)–(86) of Lemma C.1, identify its final output with $\mathsf{Prmt}_t(1)$. Equality substitution transfers these equalities through the product-law check.

The new displayed circuitry consists of $\mathfrak{q}_m(a+b)$, $\mathfrak{q}_m(0)$, $\mathfrak{q}_m(a)\mathfrak{q}_m(b)$, and their missing address, norm, and comparison checks. These circuits reuse the twiddles at a, b and cost $O(s+1)W^{O(1)}$ to construct. The product-law proof at (a, b) adds $O(s^2\log^2(s+1))W^{O(1)}$ for its proof auxiliaries and small-step inferences. The zero-address argument and check substitutions add $O(s+1)W^{O(1)}$, reusing the root certificates and parameter checks. Thus this step costs $O(s^2\log^2(s+1))W^{O(1)}$ in total.

2. *Compare the conjugate with the opposite twiddle.* Put $f' := \max\{3t, p\}$. The derivation is

$$\begin{aligned}
\bar{\mathfrak{q}}_m(a) &= \bar{\mathfrak{q}}_m(a) \cdot 1 && \text{Lemma C.1 (63c)} \\
&\overset{2\beta}{\approx}_{f'} \bar{\mathfrak{q}}_m(a)\big(\mathfrak{q}_m(a)\mathfrak{q}_m(b)\big) && \text{first error bound below} \\
&= \big(\mathfrak{q}_m(a)\bar{\mathfrak{q}}_m(a)\big)\mathfrak{q}_m(b) && \text{Lemma C.1 (63b), (63a); Lemma B.4} \\
&\overset{4s\varepsilon_{\mathrm{root}}}{\approx}_{f'} 1 \cdot \mathfrak{q}_m(b) && \text{second error bound below} \\
&= \mathfrak{q}_m(b) && \text{Lemma C.1 (63d)} \\
&= \mathfrak{q}_m(-a) && \text{definition of } b.
\end{aligned}$$

For the first approximate transition, derive

$$\begin{aligned}
&\big|\bar{\mathfrak{q}}_m(a) \cdot 1 - \bar{\mathfrak{q}}_m(a)(\mathfrak{q}_m(a)\mathfrak{q}_m(b))\big|_1 \\
&\quad = \big|\bar{\mathfrak{q}}_m(a)(1 - \mathfrak{q}_m(a)\mathfrak{q}_m(b))\big|_1 && \text{Lemma C.1 (65), (85); Lemma B.4} \\
&\quad \le |\bar{\mathfrak{q}}_m(a)|_1 \, |1 - \mathfrak{q}_m(a)\mathfrak{q}_m(b)|_1 && \text{Lemma C.1 (78c)} \\
&\quad = |\mathfrak{q}_m(a)|_1 \, |\mathfrak{q}_m(a)\mathfrak{q}_m(b) - 1|_1 && \text{Lemma C.1 (78a), (78b), (62); Lemma B.4} \\
&\quad \le 2\beta && \text{available norm and product bounds; Lemma C.1 (74)}.
\end{aligned}$$

For the second approximate transition, derive

$$\begin{aligned}
&\big|(\mathfrak{q}_m(a)\bar{\mathfrak{q}}_m(a))\mathfrak{q}_m(b) - 1 \cdot \mathfrak{q}_m(b)\big|_1 \\
&\quad = \big|(\mathfrak{q}_m(a)\bar{\mathfrak{q}}_m(a) - 1)\mathfrak{q}_m(b)\big|_1 && \text{Lemma C.1 (67), (86); Lemma B.4} \\
&\quad \le |\mathfrak{q}_m(a)\bar{\mathfrak{q}}_m(a) - 1|_1 \, |\mathfrak{q}_m(b)|_1 && \text{Lemma C.1 (78c)} \\
&\quad \le (2s\varepsilon_{\mathrm{root}})\, 2 && \text{(106) and (105a) at } b\text{; Lemma C.1 (74)} \\
&\quad = 4s\varepsilon_{\mathrm{root}} && \text{Lemma C.1 (63); evaluate } 2 \cdot 2 = 4.
\end{aligned}$$

The differences use the omitted-scale convention. The promotion identities in the factoring steps align multiplication by the scale-zero constant one with the triple products, which have scale $3t$. Promote the two error checks to f' using Lemma C.1 (88). Combining approximation checks by Lemma C.1 (79), and substituting the exact equalities using Lemma B.4, gives

$$\bar{\mathfrak{q}}_m(a) \overset{2\beta+4s\varepsilon_{\mathrm{root}}}{\approx}_{f'} \mathfrak{q}_m(-a).$$

Evaluate the closed comparison $2\beta + 4s\varepsilon_{\mathrm{root}} \le \varepsilon_{\mathrm{tw}}$ and apply Lemma C.1 (68) to enlarge the tolerance to $\varepsilon_{\mathrm{tw}}$. The error-check promotion equivalence (88) then gives the desired check at its default scale.

The new circuitry is dominated by the two triple products in the main chain and the factored differences in the error calculations. They reuse the twiddles, conjugate, and pair products already available. Construct these intermediate arithmetic outputs once, together with their missing norms, promotions, tolerance words, equality checks, and comparisons; every later occurrence refers to the same wires. There are only a constant number of these short-word operations and applications of Lemmas B.4 and C.1. The products have scale at most $3t$ and $O(W)$ numerator widths; promotion to $f' = O(W)$ preserves this width bound. Consequently the new circuits, proof auxiliaries, and small-step inferences in this step cost $W^{O(1)}$ in total.

Excluding the previously charged root certificates, parameter checks, and available norm bounds at a, b and conjugate-product bound at a, the total additional cost is $O(s^2 \log^2(s+1))W^{O(1)} = \log^{O(1)} N$. The product-law proof at (a, b) is instantiated once; the earlier norm assertions and all subsequently constructed outputs are reused.

The half-period identity: $[a < m/2] \Longrightarrow \mathfrak{q}_m(a + m/2) = -\mathfrak{q}_m(a)$**.** Write $b := (a + m/2) \bmod m$ for the address already computed in this conclusion. Adding $m/2$ changes only the highest address bit under the stated condition. Thus the two twiddle circuits select the same first $s - 1$ factors, while one selects 1 and the other selects -1 as its last factor. We retain $a < m/2$ as the antecedent throughout the following derivation.

1. *Identify the address bits.* We first derive

$$[a < m/2] \Longrightarrow \big[a_{s-1} = 0, \quad b_r = a_r \ (r < s - 1), \quad b_{s-1} = 1\big].$$

Apply Lemma C.1 (94) and (95) with $x = a$ and $w = s$. Since $m/2 = 2^{s-1}$, the first gives $a_{s-1} = 0$ under the antecedent; the second gives the stated bits of b after conditional unfolding by Lemma B.3. Reuse the supplied address adder and comparator. The two rule instances, missing checks, and unfolding steps together cost $W^{O(1)}$.

2. *Identify the selected factors and the common prefix product.* From these address-bit assertions, derive

$$\begin{aligned} c_r^{b_r} &= c_r^{a_r} && (r < s - 1), \\ c_{s-1}^{a_{s-1}} &= \mathsf{Prmt}_p(1), \\ c_{s-1}^{b_{s-1}} &= z_1 = \mathsf{Prmt}_p(-1). \end{aligned}$$

For $r < s - 1$, apply the conditional equality-substitution statement of Lemma B.4 through the two selectors $c_r^{a_r}$ and $c_r^{b_r}$. For the last factor, use Lemma B.6 with a two-entry array and the fixed addresses zero and one. The equality $z_1 = \mathsf{Prmt}_p(-1)$ is the hardwired definition in Appendix A.4; its closed check was already certified at the start of Appendix E.3.

For $s \geq 2$, let u name the existing running-product output just before the last factor in $\mathfrak{q}_m(a)$:

$$u := \prod_{r<s-1}^{\text{left}} c_r^{a_r}.$$

It has scale $(s-1)p$; naming it introduces no gates. Apply conditional equality substitution (Lemma B.4) through the corresponding prefix circuits of $\mathfrak{q}_m(a)$ and $\mathfrak{q}_m(b)$. Their first $s - 1$ factors have just been identified, so this derives equality of their prefix outputs. Traverse these existing circuits once, reusing the preceding product equality at each multiplier. The selectors and prefix multipliers are all supplied by the two twiddle circuits. The new conditional checks, proof auxiliaries, and small-step inferences cost $O(s + 1)W^{O(1)}$ in total.

3. *Extract the minus sign from the last factor.* For $s \geq 2$, the matched prefix and last-factor assertions give the chain

$$\begin{aligned} \mathfrak{q}_m(b) &= \mathsf{Prmt}_t\big(u\mathsf{Prmt}_p(-1)\big) && \text{matched prefix and last factor; Lemma B.4} \\ &= \mathsf{Prmt}_t\big(u(-1)\big) && \text{Lemma C.1 (85), (83); Lemma B.4} \\ &= \mathsf{Prmt}_t(u)(-1) && \text{Lemma C.1 (86)} \\ &= -\mathsf{Prmt}_t(u) && \text{multiplication by } -1\text{, derived below} \\ &= -\mathsf{Prmt}_t\big(u\mathsf{Prmt}_p(1)\big) && \text{Lemma C.1 (63c), (85), (83); Lemma B.4} \\ &= -\mathfrak{q}_m(a) && \text{last-factor assertion; Lemma B.4}. \end{aligned}$$

For the second equality, (85) first gives $u\mathsf{Prmt}_p(-1) = \mathsf{Prmt}_{sp}(u(-1))$; composition of promotions then removes this intermediate promotion inside Prmt_t. The same argument with 1 and multiplication

by one gives the penultimate equality. To justify multiplication by -1, use

$$\begin{aligned} x(-1) &= x(0-1) && \text{evaluate } -1 = 0-1;\ \text{Lemma B.4} \\ &= x0 - x1 && \text{Lemma C.1 (65)} \\ &= -x && \text{Lemma C.1 (63e), (63c), (62); Lemma B.4}. \end{aligned}$$

Here x stands for the already constructed output $\mathsf{Prmt}_t(u)$. All equality substitutions use the conditional version of Lemma B.4, retaining $a < m/2$. Since b names the address $a + m/2 \bmod m$, the final equality is the claimed implication.

For $s = 1$, there is no prefix multiplier. The selector assertions give $\mathfrak{q}_2(a) = \mathsf{Prmt}_t(\mathsf{Prmt}_p(1))$ and $\mathfrak{q}_2(b) = \mathsf{Prmt}_t(\mathsf{Prmt}_p(-1))$ under $a < 1$. Use Lemma C.1 (83) and evaluate the closed equality $\mathsf{Prmt}_t(-1) = -\mathsf{Prmt}_t(1)$, then apply conditional equality substitution to obtain the same conclusion.

The new circuitry in this step consists of the intermediate products, promotions, negations, and conditional equality checks in the displays. Reuse u, both twiddle outputs, and the selected constants from the preceding step; construct every missing intermediate output once. There are a constant number of short-word operations and applications of Lemmas B.4 and C.1, costing $W^{O(1)}$ in total, including the $s = 1$ case when applicable. The prefix contains $s - 1$ scale-p factors, so its width is $O(sp + s) = O(W)$. Adding the last factor and promoting to t keeps every working width and scale $O(W)$.

Including all new circuits, checks, proof auxiliaries, and small-step inferences, the half-period argument costs $O(s + 1)W^{O(1)} = \log^{O(1)} N$ beyond the shared certificates. Each prefix circuit is traversed once, and the twiddle circuits and previously derived assertions are reused throughout.

The doubling identity: $\mathfrak{q}_m(2a) = \mathfrak{q}_{m/2}(a)$. Doubling shifts the address bits left: modulo m, a zero enters the lowest position and the highest bit disappears. The circuit for $\mathfrak{q}_m(2a)$ therefore selects an initial factor $\mathsf{Prmt}_p(1)$, followed by the factors of $\mathfrak{q}_{m/2}(a)$ in the same order. We identify these factors, remove the initial one while tracking its extra scale p, and compare the final promotions to t.

1. *Identify the address bits and selected factors.* Write $(2a)_r$ for the bits of the reduced address already supplied to $\mathfrak{q}_m(2a)$. The shift and low-bit wiring of Appendix A.1 gives

$$(2a)_0 = 0, \qquad (2a)_{r+1} = a_r = (a \bmod (m/2))_r \quad (0 \le r < s - 1).$$

The quotient-and-remainder assertions of Lemma C.1 (90) certify the interpretation of these low-bit outputs as the two reduced addresses. Their bit equalities are wiring identities, so their missing checks and EF derivations cost $W^{O(1)}$.

For $s \ge 2$, the factor at position r in $\mathfrak{q}_{m/2}(a)$ chooses between $\mathsf{Prmt}_p(1)$ and $z_{(s-1)-r} = c_{r+1}$. Thus $c_{r+1}^{a_r}$ names this existing selector output in the smaller twiddle circuit. Derive

$$\begin{aligned} c_0^{(2a)_0} &= \mathsf{Prmt}_p(1) && \text{Lemma B.6} \\ c_{r+1}^{(2a)_{r+1}} &= c_{r+1}^{a_r} \quad (0 \le r < s-1) && \text{Lemma B.4}. \end{aligned}$$

The first assertion opens the two-entry selector at address zero. For the second, substitute the equal address bits through the two copies of the same selector, sharing their hardwired inputs. Both sets of selectors already belong to the conclusion's twiddle circuits. The $O(s)$ selector comparisons, their missing checks, and proof auxiliaries cost $O(s + 1)W^{O(1)}$, including the address-bit step.

2. *Remove the initial factor from the running products.* Suppose $s \geq 2$. For this step, let

$$u_r := \prod_{j<r+2}^{\text{left}} c_j^{(2a)_j}, \qquad v_r := \prod_{j<r+1}^{\text{left}} c_{j+1}^{a_j} \quad (0 \leq r < s-1)$$

name the existing prefix outputs in $\mathfrak{q}_m(2a)$ and $\mathfrak{q}_{m/2}(a)$, respectively. The superscript left retains their prescribed left-to-right multiplication order. These names introduce no gates. Their scales are $(r+2)p$ and $(r+1)p$. We will derive, successively for all these prefixes,

$$u_r = \mathsf{Prmt}_{(r+2)p}(v_r) \qquad (0 \leq r < s-1).$$

The extra factor $\mathsf{Prmt}_p(1)$ changes the encoding by one scale-p promotion, while preserving the represented value.

For the first prefix, use the factor equalities from Step 1 to obtain

$$\begin{aligned} u_0 &= \mathsf{Prmt}_p(1)v_0 && \text{Step 1; Lemma B.4} \\ &= \mathsf{Prmt}_{2p}(1v_0) && \text{Lemma C.1 (86)} \\ &= \mathsf{Prmt}_{2p}(v_0) && \text{Lemma C.1 (63d); Lemma B.4}. \end{aligned}$$

For $0 \leq r < s-2$, the next prefix follows from

$$\begin{aligned} u_{r+1} &= u_r c_{r+2}^{a_{r+1}} && \text{Step 1; Lemma B.4} \\ &= \mathsf{Prmt}_{(r+2)p}(v_r) c_{r+2}^{a_{r+1}} && \text{preceding prefix equality; Lemma B.4} \\ &= \mathsf{Prmt}_{(r+3)p}\big(v_r c_{r+2}^{a_{r+1}}\big) && \text{Lemma C.1 (86)} \\ &= \mathsf{Prmt}_{(r+3)p}(v_{r+1}) && \text{definition of } v_{r+1}. \end{aligned}$$

The last product is the existing next-prefix output in the smaller twiddle circuit. In each instance, construct only the missing promoted prefix, intermediate products, and equality checks displayed above; reuse all selected factors and both original prefix circuits. Each prefix uses a constant number of applications of Lemmas B.4 and C.1, costing $W^{O(1)}$. Propagate its equality through only the next multiplier, reusing the recorded assertion rather than copying the earlier prefix proof. There are $s-1$ prefixes, so the total additional cost of this step is $O(s)W^{O(1)}$. Every prefix contains at most s scale-p factors, of width $O(sp+s)$; the auxiliary promotions and products also have width and scale $O(W)$.

3. *Identify the final twiddle outputs.* For $s \geq 2$, the last prefix equality gives

$$\begin{aligned} \mathfrak{q}_m(2a) &= \mathsf{Prmt}_t(u_{s-2}) && \text{definition of } \mathfrak{q}_m(2a) \\ &= \mathsf{Prmt}_t\big(\mathsf{Prmt}_{sp}(v_{s-2})\big) && \text{Step 2; Lemma B.4} \\ &= \mathsf{Prmt}_t(v_{s-2}) && \text{Lemma C.1 (83)} \\ &= \mathfrak{q}_{m/2}(a) && \text{definition of } \mathfrak{q}_{m/2}(a). \end{aligned}$$

Here $sp \leq t$, so both final promotions are defined. For $s = 1$, there are no prefix multipliers and the smaller twiddle is $\mathfrak{q}_1(0)$. Directly,

$$\begin{aligned} \mathfrak{q}_2(2a) &= \mathsf{Prmt}_t\big(\mathsf{Prmt}_p(1)\big) && \text{Step 1; Lemma B.4} \\ &= \mathsf{Prmt}_t(1) && \text{Lemma C.1 (83)} \\ &= \mathfrak{q}_1(0) && \text{definition of } \mathfrak{q}_1. \end{aligned}$$

Reuse the supplied twiddle outputs and the last promoted prefix from Step 2 when $s \geq 2$. The missing composition circuit, equality checks, and a constant number of lemma applications cost $W^{O(1)}$. Promotion to t leaves every working width and scale $O(W)$.

The doubling argument therefore adds $O(s+1)W^{O(1)} = \log^{O(1)} N$ for all new circuits, checks, proof auxiliaries, and small-step inferences. The two twiddle circuits are reused throughout, and each prefix comparison is proved once. This proves the exact doubling identity.

Total cost. Summing the costs of the joint norm proof, the product-law proof, the conjugation proof, and the two exact identities gives $\log^{O(1)} N$ additional cost for all six conclusions, since $s \le L$ and $W = \log^{O(1)} N$. This includes all instantiated proofs and missing auxiliary circuits charged above, with the shared root certificates and parameter checks constructed and proved only once. Constructing the conclusion circuits also costs $\log^{O(1)} N$. The hardwired-entry certificates are charged separately in Lemma E.1.

F EF Verification of FFT Network Identities

In this section U, V denote supplied nonnegative scaled real range bounds at most N^{32}. Their scales and widths are fixed independently as part of each instance. Bound arithmetic retains full results; the proofs account for their widths when constructing the auxiliary circuits.

F.1 A Row Identity with a Variable Row Number

For an input array $\boldsymbol{v} \in \mathcal{C}_{f_0}^N$ and an L-bit word a, the circuit

$$D_a(\boldsymbol{v}) := \sum_{j<N}^{[f_0+t]} \mathsf{Mul}_{t,f_0}(\mathfrak{q}_N(aj), v_j)$$

uses short integer multiplication to compute each twiddle address $aj \bmod N$, followed by the twiddle computation circuit, exact complex multiplication, and a balanced adder tree. Each summand lies in $\mathcal{C}_{f_0+t}$ and every addition is Add_{f_0+t}; the stored output has that same scale. Numerator widths follow the full-result operator rules. For input component widths $O(W)$, the circuit has $N \log^{O(1)} N$ gates, including when a is variable.

Lemma F.1 (Selected Fourier row)**.** *Let $\boldsymbol{v} \in \mathcal{C}_{f_0}^N$, let V be a real bound word at fixed scale g, and let a be an L-bit address. Take the input component widths and the fixed scales f_0, g to be $O(W)$. Then*

$$\Gamma : \ \|\boldsymbol{v}\|_\infty \le V, \quad 0 \le V \le N^{32} \ \rhd_{\mathsf{EF}}^{N \log^{O(1)} N}$$
$$\mathsf{Sel}(\mathsf{FFT}(\boldsymbol{v}), a) \overset{\varepsilon_{\mathrm{tw}} N^3 V}{\approx} D_a(\boldsymbol{v}).$$

The same bound holds when every twiddle in both FFT *and D_a is replaced by its complex conjugate.*

Proof. The direct sum splits into an even part and an odd part. The even part agrees exactly with the smaller direct sum. In the odd part, factoring out one twiddle introduces a small error. We compare these two parts with the two child FFT outputs, then combine their already proved error bounds at the parent butterfly.

1. *Name the supplied circuits and prepare shared assertions.* The expanded context already contains

   $$\mathsf{FFT}(\boldsymbol{v}), \qquad \mathsf{Sel}(\mathsf{FFT}(\boldsymbol{v}), a), \qquad D_a(\boldsymbol{v}), \qquad \varepsilon_{\mathrm{tw}} N^3 V,$$

   and the conclusion's subtraction, norm, and comparison gates. In particular, every recursive child FFT is a subcircuit of the supplied $\mathsf{FFT}(\boldsymbol{v})$, and the twiddle computations, products, and sum-tree nodes

inside $D_a(\boldsymbol{v})$ are supplied as well. We reuse these gates. The input subarrays for recursive calls are fixed even/odd selections of the original input wires; naming them introduces no gates.

Apply Lemma B.3 once to $\|\boldsymbol{v}\|_\infty \le V$ to obtain the list $[|v_j|_1 \le V]_{j<N}$, at additional cost $O(N)$. Every recursive call reuses the appropriate entries of this list. Construct any missing root checks and apply Lemma E.2, at cost $\log^{O(1)} N$. These root certificates are shared throughout the proof. Construct the product $\varepsilon_{\mathrm{tw}} V$ and its sign check if they are absent. Recall $W = O(L^4)$ from Appendix A.5.4. The product and its sign check have $O(W)$-bit working numerators, since the full product width is the sum of its operand widths. The closed parameter checks and the order-preservation and multiplication-by-zero statements of Lemma C.1 give $\varepsilon_{\mathrm{tw}} \ge 0$ and $\varepsilon_{\mathrm{tw}} V \ge 0$, at cost $W^{O(1)}$ including these gates.

For each size $m = 2^s \ge 2$, let a_s name the low s bits of the original address a. Construct $\mathfrak{q}_m(a_s)$ and the conditional-check circuits of Lemma E.1, reusing any identical circuits already available, and derive

$$\{[a_s = j] \Longrightarrow \mathfrak{q}_m(a_s) = q_m(j)\}_{j<m}\,.$$

By that lemma, all these constructions and derivations cost $N \log^{O(1)} N$ over the sizes. All recursive calls of size m share the computation, address checks, and conditional certificates. Including the preceding input bounds and root and parameter checks, the total new cost of this step is $N \log^{O(1)} N$.

2. *Construct the auxiliary row circuits.* Consider one size-$m = 2^s$ recursive FFT call within the supplied FFT_N circuit. In the formulas for this call, use $\boldsymbol{v} = (v_j)_{j<m}$ as a local name for the array of existing entry wires fed into that call. At the root this is the lemma's original input array. The two size-$N/2$ calls receive the original entries at indices $0, 2, 4, \ldots$ and $1, 3, 5, \ldots$, respectively; deeper calls take the even or odd entries of their parent's input array in the same way. Thus this local $\boldsymbol{v}$ consists of original input-entry wires selected by fixed wiring. The original entries may be free input wires or outputs of earlier circuits in Γ; this local renaming introduces no new input wires or gates. All these entries retain their input scale f_0.

   Also use a locally for the low s bits of the original address, which requires only wiring. On this call's input array, define

$$X := \mathsf{Sel}(\mathsf{FFT}_m(\boldsymbol{v}), a), \qquad D_{m,a}(\boldsymbol{v}) := \sum_{j<m}^{[f_1]} \mathfrak{q}_m(aj) v_j, \qquad f_1 := f_0 + t.$$

   At the root, X and $D_{N,a}(\boldsymbol{v}) = D_a(\boldsymbol{v})$ name supplied outputs and introduce no gates. At every other recursive FFT call, construct X on that call's existing output array $\mathsf{FFT}_m(\boldsymbol{v})$, and construct $D_{m,a}(\boldsymbol{v})$ using the same input-entry wires v_j that feed the call. These circuits are constructed once for that call and reused.

   If $w = O(W)$ bounds the input component widths, the FFT widths are $w + O(s(t + L))$. Recall from Appendix A.4 that $w_q = t + 3L + 2$ is the component numerator width of a twiddle. Each direct-sum leaf product has component width at most $w + w_q + 1$; the balanced sum tree adds at most $\log m$ bits. Thus the selector and direct sum constructed here have $O(W)$-bit working numerators. The selector costs $O(mW)$ gates. The m modular address products, twiddle computations, complex products, and the balanced sum tree together cost $m \log^{O(1)} N$ gates. This is the total additional cost of this step for a non-root call; at the root both circuits are already supplied.

3. *Identify the selected FFT butterfly.* For each size-$m = 2^s \ge 2$ call, put $b := a \bmod (m/2)$, using the existing low address bits. As in Appendix A.5.1, write $\boldsymbol{E} := \mathsf{FFT}_{m/2}(\boldsymbol{v}^{\mathrm{e}})$ and $\boldsymbol{O} := \mathsf{FFT}_{m/2}(\boldsymbol{v}^{\mathrm{o}})$ for

the output arrays of the two child calls, and consider the outputs $\mathsf{Sel}(\boldsymbol{E}, b)$ and $\mathsf{Sel}(\boldsymbol{O}, b)$ of the child selectors constructed in Step 2. Both have scale $f_{s-1} := f_0 + (s-1)t$. These names introduce no gates. Put $f_s := f_0 + st = f_{s-1} + t$. We will prove the exact identity

$$X = \mathsf{Prmt}_{f_s}(\mathsf{Sel}(\boldsymbol{E}, b)) +_{f_s} \mathsf{Mul}_{t,f_{s-1}}(\mathfrak{q}_m(a), \mathsf{Sel}(\boldsymbol{O}, b)).$$

The outputs X, $\mathsf{Sel}(\boldsymbol{E}, b)$, and $\mathsf{Sel}(\boldsymbol{O}, b)$ are already available. Let $\mathfrak{q} := \mathfrak{q}_m(a)$ name the shared twiddle output constructed in Step 1 for this size. The local address a names the same low bits as a_s, so the conditional certificates from that step apply to this output. These names introduce no gates. Construct the right-hand arithmetic circuit and its equality check against X. If $w' = O(W)$ bounds the component widths of $\mathsf{Sel}(\boldsymbol{E}, b)$ and $\mathsf{Sel}(\boldsymbol{O}, b)$, the product has width at most $w' + w_q + 1$, the promoted $\mathsf{Sel}(\boldsymbol{E}, b)$ has width $w' + t$, and the final sum adds one bit to the larger width. Thus these new circuits have $O(W)$-bit working numerators and cost $W^{O(1)}$. Apply the norm bound of Lemma E.3 to this same $\mathfrak{q}$ to record $|\mathfrak{q}|_1 \le 2$ for the following steps, at additional cost $\log^{O(1)} N$ including its check and proof auxiliaries.

The FFT construction in Appendix A.5.1 already supplies, for each fixed $j < m$,

$$(\mathsf{FFT}_m(\boldsymbol{v}))_j = \mathsf{Prmt}_{f_s}(E_{j \bmod (m/2)}) +_{f_s} \mathsf{Mul}_{t,f_{s-1}}(q_m(j), O_{j \bmod (m/2)}),$$

where the entries $E_{j \bmod (m/2)}$ and $O_{j \bmod (m/2)}$ of the existing child outputs are read by wiring, and $q_m(j)$ is the hardwired coefficient defined in Appendix A.4.

Reuse Step 1's address checks $[a = j]$ for the m address cases. Under $a = j$, the low-bit wiring gives $b = j \bmod (m/2)$. Apply Lemma B.6 to the parent selector at j and to each child selector at $j \bmod (m/2)$ to prove

$$X = (\mathsf{FFT}_m(\boldsymbol{v}))_j, \qquad \mathsf{Sel}(\boldsymbol{E}, b) = E_{j \bmod (m/2)}, \qquad \mathsf{Sel}(\boldsymbol{O}, b) = O_{j \bmod (m/2)}.$$

These three applications add $O(W \log(m+1))$ for their auxiliary checks and small-step inferences per address case. The recorded conditional certificate from Step 1 gives $\mathfrak{q}_m(a) = q_m(j)$ under this same condition. Apply conditional equality substitution from Lemma B.4 through the local promotion, multiplier, and adder. Together with the displayed FFT gate definitions, these equalities prove

$$[a = j] \Longrightarrow X = \mathsf{Prmt}_{f_s}(\mathsf{Sel}(\boldsymbol{E}, b)) +_{f_s} \mathsf{Mul}_{t,f_{s-1}}(\mathfrak{q}_m(a), \mathsf{Sel}(\boldsymbol{O}, b)).$$

The child FFTs producing $\boldsymbol{E}$ and $\boldsymbol{O}$ are shared; this substitution does not copy their circuits.

Apply Lemma B.5 to these m conditional equalities to prove the target equality without an address premise. Its auxiliary decoder, checks, and small-step inferences cost $O(m \log(m+1))$. Together with the applications of Lemma B.6 and local substitutions above, the total additional cost is

$$m \log^{O(1)} m \, W^{O(1)}.$$

The parent and child selectors from Step 2 and the conditional twiddle certificates from Step 1 are reused. Adding the target-circuit cost, the twiddle-norm proof, and the address-case derivations gives total additional cost $m \log^{O(1)} N$ for the step. All FFT, child-selector, and twiddle-computation outputs are shared across the address cases.

4. *Set the invariant and prove the size-one cases.* Put $g' := p + g$ and $f'_s := \max\{f_1, g'\} + st$. These specify the error-word scale and the comparison scale at size 2^s. Construct the scale-g' error words

$$\varepsilon_1 := \mathsf{Prmt}_{g'}(0), \qquad \varepsilon_m := 3\varepsilon_{m/2} +_{g'} m\varepsilon_{\mathrm{tw}} V \quad (m = 2, 4, \ldots, N).$$

There is one such circuit per size, shared by all calls of that size. Multiplying the preceding error word by 3 adds only a constant number of bits; multiplication of $\varepsilon_{\mathrm{tw}}V$ by $m = 2^s$ adds s bits, and the final addition adds one bit to the larger operand width. All these words have scale g', so the recurrence needs no further scale promotion. Over the L levels, the widths therefore grow by only $O(L)$ beyond the larger of the widths of ε_1 and $\varepsilon_{\mathrm{tw}}V$, both already $O(W)$. Consequently all error words have width $O(W)$, and constructing the products and additions in this list costs $O(L+1)W^{O(1)}$ in total. Using $\varepsilon_{\mathrm{tw}}V \geq 0$, the addition identities and order preservation of Lemma C.1 derive $\varepsilon_m \geq 0$ once per size, at total cost $O(L+1)W^{O(1)}$ including the sign checks. We will derive, separately for each call, the assertion

$$X \overset{\varepsilon_m}{\approx}_{f'_s} D_{m,a}(\boldsymbol{v}).$$

At $m = 1$, $X = v_0$ and $D_{1,0}(\boldsymbol{v}) = \mathfrak{q}_1(0)v_0$. For these size-one calls, construct the invariant's missing promotions, subtraction, norm, and comparison gates on the existing operands. Promotion to $f'_0 = \max\{f_0 + t, p + g\} = O(W)$ adds $O(W)$ bits; the subtraction and norm add only a constant number of bits. These new check circuits therefore cost $W^{O(1)}$ per leaf. Since $\mathfrak{q}_1(0) = \mathsf{Prmt}_t(1)$, multiplication by one and the multiplication-promotion identities of Lemma C.1 give $D_{1,0}(\boldsymbol{v}) = \mathsf{Prmt}_{f_1}(v_0)$. Composition of promotions identifies the two outputs at f'_0. Additive inverses and the absolute-value gates then give zero error. Lemma B.4 transfers these equalities to the error check. Each leaf uses $W^{O(1)}$ additional gates and small-step inferences, hence $NW^{O(1)}$ for all size-one calls. Together with the error words and their nonnegativity assertions, the total cost of this step is $O(L+1)W^{O(1)} + NW^{O(1)}$.

The remaining steps process a size-m call, $m \geq 2$, after its two children. Reuse b, $\mathsf{Sel}(\boldsymbol{E}, b)$, and $\mathsf{Sel}(\boldsymbol{O}, b)$ from Step 3, and name the already constructed child direct-sum outputs

$$A := D_{m/2,b}(\boldsymbol{v}^{\mathrm{e}}), \qquad B := D_{m/2,b}(\boldsymbol{v}^{\mathrm{o}}).$$

Both have scale f_1. These names introduce no gates. The child error assertions

$$\mathsf{Sel}(\boldsymbol{E}, b) \overset{\varepsilon_{m/2}}{\approx}_{f'_{s-1}} A, \qquad \mathsf{Sel}(\boldsymbol{O}, b) \overset{\varepsilon_{m/2}}{\approx}_{f'_{s-1}} B$$

are already proved and will be cited without copying their derivations.

5. *Split the direct sum and identify its even part.* In this step, we derive in EF $D_{m,a}(\boldsymbol{v}) = A + C$ where C is defined below. The derivation works as follows.

$$\begin{aligned}
D_{m,a}(\boldsymbol{v}) &= \underbrace{\sum_{j<m/2}^{[f_1]} \mathfrak{q}_m(2aj)v_{2j}}_{\text{even part}} +_{f_1} \underbrace{\sum_{j<m/2}^{[f_1]} \mathfrak{q}_m(a(2j+1))v_{2j+1}}_{C\text{: odd part}} && \text{Lemma D.6} \\
&= \sum_{j<m/2}^{[f_1]} \mathfrak{q}_{m/2}(aj \bmod (m/2))v_{2j} +_{f_1} C && \text{Lemma E.3 (105f); Lemma B.4} \\
&= \sum_{j<m/2}^{[f_1]} \mathfrak{q}_{m/2}(bj \bmod (m/2))v_{2j} +_{f_1} C && \text{Lemma C.1 (93); Lemma B.4} \\
&= D_{m/2,b}(\boldsymbol{v}^{\mathrm{e}}) +_{f_1} C && \text{definitions of } D_{m/2,b} \text{ and } \boldsymbol{v}^{\mathrm{e}} \\
&= A +_{f_1} C && \text{definition of } A.
\end{aligned}$$

All additions in the display have scale f_1.

The new circuitry is dominated by the reordered sum

$$\left(\sum_{j<m/2}^{[f_1]} \mathfrak{q}_m(2aj)v_{2j}\right) +_{f_1} \left(\sum_{j<m/2}^{[f_1]} \mathfrak{q}_m(a(2j+1))v_{2j+1}\right)$$

and the intermediate sum $\sum_{j<m/2}^{[f_1]} \mathfrak{q}_{m/2}(aj \bmod (m/2))v_{2j}$. The reordered sum reuses all its leaves from terms available in $D_{m,a}(\boldsymbol{v})$ and adds only addition gates; its odd subtree supplies C. The intermediate sum reuses the inputs v_{2j} and requires any missing address, twiddle, product, and addition gates. The final even sum $D_{m/2,b}(\boldsymbol{v}^{\mathrm{e}}) = A$ and its leaves are already available. Together with the outer adders involving C and the equality checks, these new circuits have size $m \log^{O(1)} N$. The product widths are $w + w_q + O(1)$, and balanced addition adds $O(\log(m+1))$ bits, so all working widths are $O(W)$.

Apply Lemma D.6 once to the fixed even/odd permutation, and apply Lemmas C.1 and E.3 $O(m)$ times to the short address calculations and twiddle identities. For each transition, apply Lemma B.4 through the twiddle and product circuits and then once per sum-tree node, sharing the child equalities. Retain the twiddle equalities $\mathfrak{q}_m(2aj) = \mathfrak{q}_{m/2}(bj)$ for the next step. These inference rules, including all their auxiliary circuits and small-step inferences, have total additional cost at most $m \log^{O(1)} m\, W^{O(1)} + m \log^{O(1)} N = m \log^{O(1)} N$. Including the displayed circuit constructions, the total additional cost of this step is therefore $m \log^{O(1)} N$.

6. *Bound the error from factoring the odd sum.*

Put $f'' := \max\{f_1 + t, g'\}$. We derive in EF

$$\mathfrak{q}_m(a)B \overset{(m/2)\varepsilon_{\mathrm{tw}}V}{\approx}_{f''} C$$

as follows:

$$\begin{aligned}
\mathfrak{q}_m(a)B &= \sum_{j<m/2}^{[f_1+t]} \big(\mathfrak{q}_m(a)\mathfrak{q}_{m/2}(bj)\big)v_{2j+1} && \text{Lemma C.1 (63b), (64); Lemma B.4}\\
&= \sum_{j<m/2}^{[f_1+t]} \big(\mathfrak{q}_m(a)\mathfrak{q}_m(2aj)\big)v_{2j+1} && \text{Step 5; Lemma B.4}\\
&\overset{(m/2)\varepsilon_{\mathrm{tw}}V}{\approx}_{f''} \sum_{j<m/2}^{[f_1]} \mathfrak{q}_m\big(a + (2aj \bmod m)\big)v_{2j+1} && \text{Lemma E.3 (105b); error propagation below}\\
&= \sum_{j<m/2}^{[f_1]} \mathfrak{q}_m(a + 2aj)v_{2j+1} && \text{Lemma C.1 (92); Lemma B.4}\\
&= \sum_{j<m/2}^{[f_1]} \mathfrak{q}_m(a(2j+1))v_{2j+1} && \text{Lemma C.1 (62a), (63), (64); Lemma B.4}\\
&= C && \text{definition of } C.
\end{aligned}$$

The approximate row displays the product law's second address $2aj \bmod m$ explicitly. The next equality applies (92) with $x = a$, $y = 2aj$, and $2^k = m$. To rewrite $a + 2aj$ as $a(2j+1)$, use

Lemma C.1 (59) before applying the multiplication identities to the factor two, and transfer back by Lemma B.4. This adds $W^{O(1)}$ per address, within the step's accounting below. Each address equality is transferred through the twiddle computation, multiplication by v_{2j+1}, and the sum tree using Lemma B.4.

For the approximate transition, name the two existing summand outputs

$$\xi_j := \big(\mathfrak{q}_m(a)\mathfrak{q}_m(2aj)\big)v_{2j+1}, \qquad \zeta_j := \mathfrak{q}_m\big(a + (2aj \bmod m)\big)v_{2j+1}.$$

They have scales $f_1 + t$ and f_1, respectively; these names introduce no gates. For each $j < m/2$, derive

$$\begin{aligned}
|\xi_j - \zeta_j|_1 &= |\big(\mathfrak{q}_m(a)\mathfrak{q}_m(2aj) - \mathfrak{q}_m\big(a + (2aj \bmod m)\big)\big)v_{2j+1}|_1 && \text{Lemma C.1 (67), (86); Lemma B.4}\\
&\le |\mathfrak{q}_m(a)\mathfrak{q}_m(2aj) - \mathfrak{q}_m\big(a + (2aj \bmod m)\big)|_1\,|v_{2j+1}|_1 && \text{Lemma C.1 (78c)}\\
&\le \varepsilon_{\mathrm{tw}}|v_{2j+1}|_1 && \text{Lemma E.3 (105b); Lemma C.1 (74)}\\
&\le \varepsilon_{\mathrm{tw}}V && \text{Step 1: } |v_{2j+1}|_1 \le V\text{; Lemma C.1 (74)}.
\end{aligned}$$

Here the subtraction uses the omitted-scale convention. The promotion identity in the first line aligns the product scales before factoring.

The error between the two sums is bounded at scale f'' by

$$\begin{aligned}
&\left|\mathsf{Prmt}_{f''}\left(\sum_{j<m/2}^{[f_1+t]}\xi_j\right) -_{f''} \mathsf{Prmt}_{f''}\left(\sum_{j<m/2}^{[f_1]}\zeta_j\right)\right|_1\\
&= \left|\sum_{j<m/2}^{[f'']}\big(\mathsf{Prmt}_{f''}(\xi_j) -_{f''} \mathsf{Prmt}_{f''}(\zeta_j)\big)\right|_1 && \text{Lemma D.5; Lemma C.1 (62); Lemma B.4}\\
&\le \sum_{j<m/2}^{[f'']}|\mathsf{Prmt}_{f''}(\xi_j) -_{f''} \mathsf{Prmt}_{f''}(\zeta_j)|_1 && \text{Lemma C.1 (77), (73)}\\
&\le \sum_{j<m/2}^{[f'']}\mathsf{Prmt}_{f''}(\varepsilon_{\mathrm{tw}}V) && \text{the summand bounds above; Lemma C.1 (88), (73)}\\
&= \mathsf{Prmt}_{f''}((m/2)\varepsilon_{\mathrm{tw}}V) && \text{Lemma C.1 (66), (63d); Lemma D.5}.
\end{aligned}$$

The tree equalities and inequalities are derived once per node, reusing the two child assertions. This proves the approximate transition with tolerance $(m/2)\varepsilon_{\mathrm{tw}}V$.

The new circuitry is dominated by the intermediate product sums in the chain from $\mathfrak{q}_m(a)B$ to C. They reuse $\mathfrak{q}_m(a)$, the child twiddles $\mathfrak{q}_{m/2}(bj)$ from B, the twiddles $\mathfrak{q}_m(2aj)$ from Step 5, and the input entries v_{2j+1}. The sum defining C is already available. Construct $\mathfrak{q}_m(a)B$ and any missing address and twiddle circuits, products, adders, promotions, tolerance words, and comparison checks, including the difference and tolerance sum trees in the error calculation. Each product contains one input entry and at most two twiddles, so the sum widths are $w + 2w_q + O(\log(m+1)) = O(W)$ before comparison promotion, and remain $O(W)$ afterwards because $f'' = \max\{f_0 + 2t, p + g\} = O(W)$. These constructions cost $m \log^{O(1)} N$.

The derivations use $O(m)$ applications of Lemmas C.1 and E.3, together with Lemmas B.4 and D.5, sharing the available root certificates. For the distributive transition, construct the products of $\mathfrak{q}_m(a)$ with the existing node outputs of B's sum tree, reuse $\mathfrak{q}_m(a)B$ at the root, and prove one equality

per node. Equality and error propagation likewise reuse the two child assertions at each parent. Use Lemma B.4 through the final error check to identify its operands with $\mathfrak{q}_m(a)B$ and C. All proof auxiliaries and small-step inferences therefore cost $m \log^{O(1)} N$. Including the circuit constructions, the total additional cost of this step is $m \log^{O(1)} N$.

7. *Combine the two child errors at the parent.* Construct the auxiliary comparison output

$$Y := \mathsf{Prmt}_{f'_s}(A) +_{f'_s} \mathsf{Mul}_{t,f'_{s-1}}(\mathfrak{q}, \mathsf{Prmt}_{f'_{s-1}}(B)).$$

Reuse $\mathfrak{q}, A, B$ and the promoted word $\mathsf{Prmt}_{f'_{s-1}}(B)$ from the odd child's error check. The new arithmetic in Y is one multiplier and one adder, together with any missing promotion wiring. Construct the analogous expression on $\mathsf{Sel}(\boldsymbol{E}, b)$ and $\mathsf{Sel}(\boldsymbol{O}, b)$, reusing $\mathsf{Prmt}_{f'_{s-1}}(\mathsf{Sel}(\boldsymbol{O}, b))$ from that same child check. The two assertions to be proved are

$$X \overset{3\varepsilon_{m/2}}{\approx}_{f'_s} Y,$$
$$Y \overset{(m/2)\varepsilon_{\mathrm{tw}}V}{\approx}_{f'_s} D_{m,a}(\boldsymbol{v}).$$

Construct their missing error-check gates and the parent invariant's check gates. Here $f'_s = \max\{f_0 + t, p+g\} + st = O(W)$, since $s \le L$. Promoting the existing operands adds $O(W)$ bits, multiplication by $\mathfrak{q}$ adds at most $w_q + 1$ bits, and the remaining additions, differences, and norms add only a constant number of bits. Thus these new arithmetic and check circuits have $O(W)$-bit working numerators and cost $W^{O(1)}$ altogether.

For the first check, the promotion identities of Lemma C.1 turn Step 3's exact equality into

$$\mathsf{Prmt}_{f'_s}(X) = \mathsf{Prmt}_{f'_s}(\mathsf{Sel}(\boldsymbol{E}, b)) +_{f'_s} \mathsf{Mul}_{t,f'_{s-1}}(\mathfrak{q}, \mathsf{Prmt}_{f'_{s-1}}(\mathsf{Sel}(\boldsymbol{O}, b))).$$

Here $f'_s = f'_{s-1} + t$. Promote the even child error to f'_s. For the odd child, subtraction distributivity (65), the complex-norm product bound, and order preservation from Lemma C.1 multiply its tolerance by $|\mathfrak{q}|_1 \le 2$, proved in Step 3. The triangle inequality then gives $\varepsilon_{m/2} + 2\varepsilon_{m/2} = 3\varepsilon_{m/2}$. Use Lemma B.4 through the short error checks to identify their operands with the displayed outputs.

For the second check, promote the equality from Step 5 to obtain

$$\mathsf{Prmt}_{f'_s}(D_{m,a}(\boldsymbol{v})) = \mathsf{Prmt}_{f'_s}(A) +_{f'_s} \mathsf{Prmt}_{f'_s}(C).$$

Promotion of multiplication also identifies Y with $\mathsf{Prmt}_{f'_s}(A) +_{f'_s} \mathsf{Prmt}_{f'_s}(\mathfrak{q}B)$. Cancel the common summand using the addition identities of Lemma C.1, and promote the error check from Step 6; this is permitted because $f'_s \ge f''$ for $s \ge 1$.

A final application of "Combining approximation checks" in Lemma C.1 combines the two checks. Order preservation, $m/2 \le m$, and $\varepsilon_{\mathrm{tw}}V \ge 0$ give $3\varepsilon_{m/2} + (m/2)\varepsilon_{\mathrm{tw}}V \le 3\varepsilon_{m/2} + m\varepsilon_{\mathrm{tw}}V = \varepsilon_m$. This proves the invariant of Step 4 at the parent. Each argument uses a constant number of short-word lemma applications, at additional cost $W^{O(1)}$ for their proof auxiliaries and small-step inferences. Including the circuits constructed in this step, its total additional cost is $W^{O(1)}$. The child circuits and their proofs are reused. Once this parent check is recorded, the next call cites it.

8. *Bound the tolerances and obtain the stated conclusion.* For each size, derive $\varepsilon_m \le \varepsilon_{\mathrm{tw}} m^3 V$. The base word is zero. At $m \ge 2$, order preservation, distributivity, and the addition and multiplication identities of Lemma C.1 give

$$\varepsilon_m = 3\varepsilon_{m/2} +_{g'} m\varepsilon_{\mathrm{tw}}V \le \big(3(m/2)^3 + m\big)\varepsilon_{\mathrm{tw}}V \le m^3\varepsilon_{\mathrm{tw}}V.$$

The last step evaluates the closed comparison $3(m/2)^3 + m \le m^3$ and multiplies it by the nonnegative word $\varepsilon_{\mathrm{tw}} V$. Construct the missing envelope products and comparison checks once per size. The integer factors $3(m/2)^3 + m$ and m^3 have $O(L)$ bits, so their products with $\varepsilon_{\mathrm{tw}} V$ and the comparisons have width $O(W)$. All $L+1$ envelope derivations cost $O(L+1)W^{O(1)}$; every call of a given size shares that assertion.

At the root, order transitivity replaces ε_N by the already supplied tolerance $\varepsilon_{\mathrm{tw}} N^3 V$, giving

$$\mathsf{Sel}(\mathsf{FFT}(\boldsymbol{v}), a) \overset{\varepsilon_{\mathrm{tw}} N^3 V}{\approx}_{f'_L} D_a(\boldsymbol{v}).$$

The default scale of the conclusion is $\max\{f_L, g'\} \le f'_L$. The "Promotion identities and error checks" of Lemma C.1 convert the proved check at f'_L to the supplied check at that scale. Both scales are $O(W)$, so the conversion has $O(W)$-bit working numerators. This final conversion costs $W^{O(1)}$, including its proof auxiliaries; the FFT output, direct sum, tolerance, and target check are reused. The total additional cost of this step is $O(L+1)W^{O(1)}$.

9. *Obtain the conjugated-twiddle version.* For this version, write $\overline{\mathsf{FFT}}_m$ for FFT_m with every hardwired twiddle replaced by its complex conjugate, and put $\overline{\mathsf{FFT}} := \overline{\mathsf{FFT}}_N$. The corresponding conclusion supplies this network and its direct sum with conjugated twiddles. The complex arithmetic definitions and the addition, multiplication, and distributivity identities of Lemma C.1 prove that conjugation commutes with sums and products. Its "Complex norm identities and bounds" show that conjugation preserves the 1-norm. Conjugating each twiddle identity therefore preserves its equality or error tolerance, at additional cost $W^{O(1)}$ per identity. The conditional version of Lemma B.4 gives

$$[a_s = j] \Longrightarrow \overline{\mathfrak{q}_m(a_s)} = \overline{q_m(j)}$$

from each hardwired-twiddle certificate in Step 1. Reuse the root certificates, and perform the same steps on the conjugated network. Each conjugation negates one component, adding at most one bit; the width calculations in the preceding steps therefore still give $O(W)$. The added conjugation circuits and derivations preserve the $m \log^{O(1)} N$ cost per call and $O(L+1)W^{O(1)}$ envelope cost.

Total cost. Step 1's input-bound unfolding, shared root and parameter certificates, twiddle computations, and conditional hardwired-twiddle certificates cost $N \log^{O(1)} N$ in total. The shared error-word and envelope circuits and their derivations cost $O(L+1)W^{O(1)}$. For a size-m call, the new row circuits, regrouping, selector proofs, odd-branch comparison, and parent error derivation in Steps 2–3 and 5–7 together cost $m \log^{O(1)} N$; Step 4's size-one checks and proofs cost $W^{O(1)}$ each. There are $N/2^s$ calls of size 2^s, so the total is

$$N \log^{O(1)} N + O(L+1)W^{O(1)} + \sum_{s=0}^{L} \frac{N}{2^s} 2^s \log^{O(1)} N = N \log^{O(1)} N.$$

Every newly constructed circuit is charged at its construction step; later steps share its output wires and recorded assertions. Each call expands one selected row. Every child derivation is constructed once; its parent cites the resulting assertion without copying the derivation. The conjugated version changes the total by at most a constant factor. □

F.2 Exact Linearity and Transpose Rules

The following big-step inference rules concern the full-result circuits FFT, $\overline{\mathsf{FFT}}$, and IFFT. All operand scales and widths are fixed, and every working width is $O(W)$. They use the arithmetic, padding, and sum-tree inference rules of Lemma C.1 and appendix D.2. They require no range or accuracy premise. Each

equality includes promotion and sign extension to a common comparison type. The proof constructions specify their intermediate circuits; all definitions absent from the initial expanded context and all new proof steps are charged to the displayed additional budget.

Lemma F.2 (Exact network linearity in EF)**.** *Let arrays $\boldsymbol{u}, \boldsymbol{v} \in \mathcal{C}_{f_0}^N$ and a scalar $c \in \mathcal{C}_g$ be available in Γ. Then*

$$\Gamma \ \rhd_{\mathsf{EF}}^{N \log^{O(1)} N}$$

$$\mathsf{FFT}(\boldsymbol{u} +_{f_0} \boldsymbol{v}) = \mathsf{FFT}(\boldsymbol{u}) +_{f_0+Lt} \mathsf{FFT}(\boldsymbol{v}), \qquad \mathsf{FFT}(c\boldsymbol{u}) = c(\mathsf{FFT}(\boldsymbol{u})), \tag{110}$$

$$\mathsf{IFFT}(\boldsymbol{u} +_{f_0} \boldsymbol{v}) = \mathsf{IFFT}(\boldsymbol{u}) +_{f_0+Lt+L} \mathsf{IFFT}(\boldsymbol{v}), \qquad \mathsf{IFFT}(c\boldsymbol{u}) = c(\mathsf{IFFT}(\boldsymbol{u})). \tag{111}$$

The scalar equalities have common output scales $f_0 + g + Lt$ and $f_0 + g + Lt + L$, respectively; scalar products use Mul *except that fixed power-of-two scalars use the wiring of Appendix A.2. The two assertions for* FFT *also hold for* $\overline{\mathsf{FFT}}$*. Inputs at different scales are explicitly promoted to a common input scale first. All newly introduced auxiliary gates and instantiated proof blocks are included in the bound.*

Proof. The networks and endpoint arithmetic in the conclusions are already in the expanded context. We reuse their internal wires and construct only the intermediate arithmetic and checks needed to compare corresponding layers. Recall $L = \log N$ and $W = O(L^4)$ from Appendix A.5.4. All equalities below compare full results; no twiddle-accuracy or range certificate is needed.

Additivity of FFT. We prove the first identity in (110) by maintaining additivity at every corresponding wire of the three supplied networks.

1. *Name the layer outputs and construct the intermediate checks.* Let $\boldsymbol{U}^{(i)}, \boldsymbol{V}^{(i)}, \boldsymbol{S}^{(i)}$ name the arrays after i levels of $\mathsf{FFT}(\boldsymbol{u})$, $\mathsf{FFT}(\boldsymbol{v})$, and $\mathsf{FFT}(\boldsymbol{u} +_{f_0} \boldsymbol{v})$, respectively. These names introduce no gates. Each entry has scale $f_i := f_0 + it$. For every $0 \le i \le L$ and fixed $j < N$, we will derive

$$S_j^{(i)} = U_j^{(i)} +_{f_i} V_j^{(i)}. \tag{112}$$

   Construct the coordinate additions on the right for $1 \le i < L$; the additions at levels zero and L are already supplied. Construct the missing coordinate-equality checks as well. The input identities follow from the input-sum wiring, with sign extension where the network's boundary width requires it.

   Each new addition takes two existing $O(W)$-bit numerator words and adds only one bit to their maximum width. Thus these additions and checks cost $O(NL)W^{O(1)}$ in total; the input wiring identities cost $O(NW)$. No new FFT instance is constructed.

2. *Derive one output equality from the two input equalities.* Fix a layer $i < L$ and one output of a butterfly in a size-m recursive FFT call. Let q be its hardwired coefficient, the same in all three network copies. For an output at position a of that call, $q = q_m(a)$. Let x, y name its two input words from $\boldsymbol{U}^{(i)}$, and x', y' the corresponding words from $\boldsymbol{V}^{(i)}$. Put $f = f_i$ and $f' = f + t$. If this output occupies position j in the next layer, derive

$$\begin{aligned}
S_j^{(i+1)} &= \mathsf{Prmt}_{f'}(x +_f x') +_{f'} q(y +_f y') && \text{(112); Lemma B.4 (50)} \\
&= \big(\mathsf{Prmt}_{f'}(x) +_{f'} \mathsf{Prmt}_{f'}(x')\big) +_{f'} (qy +_{f'} qy') && \text{Lemma C.1 (84), (64)} \\
&= \big(\mathsf{Prmt}_{f'}(x) +_{f'} qy\big) +_{f'} \big(\mathsf{Prmt}_{f'}(x') +_{f'} qy'\big) && \text{Lemma C.1 (62a), (62b)} \\
&= U_j^{(i+1)} +_{f'} V_j^{(i+1)} && \text{the supplied butterfly outputs.}
\end{aligned}$$

Every product by q uses $\mathsf{Mul}_{t,f}$. For the first transition, construct a local butterfly output on the already available sums $x +_f x'$ and $y +_f y'$ and substitute the two incoming equalities through this block. The products qy, qy' and promotions of x, x' are already in the two supplied FFTs. Only missing local arithmetic, regrouping, and equality-check gates are constructed. The width-padding argument in the proof of Lemma C.1 aligns any differing operand widths before applying Lemma B.4.

Each displayed expression has a fixed number of short-word operations. Its full numerators have width $O(W)$, including the promotion by t. All new local gates and the constant number of inference-rule applications therefore cost $W^{O(1)}$ per output.

3. *Traverse the layers and collect the output equalities.* Apply the preceding local derivation to every output in layer $i + 1$, then proceed to the next layer. The intervening fixed permutations route the same positions in all three networks, so they reuse the corresponding recorded equalities. Each layer assertion is proved once and reused at the next layer; equality is propagated only through the next local block. After L levels, (112) gives every coordinate of the desired FFT equality. Use Lemmas B.2 and B.3 to extract the coordinate bit equalities and combine them into the supplied vector-equality check.

   There are NL local output comparisons, costing $O(NL)W^{O(1)}$ altogether. Collecting the final equality costs $O(NW)$. Including the intermediate additions and checks in Step 1, the total additional cost for FFT additivity is $O(NL)W^{O(1)}$. The same construction proves additivity of $\overline{\mathsf{FFT}}$ using its supplied conjugated coefficients; the argument needs only that corresponding blocks use the same coefficient. Its additional cost is again $O(NL)W^{O(1)}$.

Additivity of IFFT. We prove the first identity in (111) by the same layer comparison, followed by exact division by N. Use the three supplied unnormalized inverse subcircuits on $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{u} + \boldsymbol{v}$. Name their layer outputs as above and construct their intermediate coordinate sums and equality checks. At a reversed block with input scale f, put $f' = f + t$ and let $\bar{q}$ be its shared hardwired coefficient. On input pairs (x, y) and (x', y'), its two output identities are

$$\begin{aligned}
\mathsf{Prmt}_{f'}\big((x +_f x') +_f (y +_f y')\big) &= \mathsf{Prmt}_{f'}\big((x +_f y) +_f (x' +_f y')\big) && \text{Lemma C.1 (62a), (62b)}\\
&= \mathsf{Prmt}_{f'}(x +_f y) +_{f'} \mathsf{Prmt}_{f'}(x' +_f y') && \text{Lemma C.1 (84),}
\end{aligned}$$

and

$$\begin{aligned}
\bar{q}\big((x +_f x') -_f (y +_f y')\big) &= (\bar{q}x +_{f'} \bar{q}x') -_{f'} (\bar{q}y +_{f'} \bar{q}y') && \text{Lemma C.1 (65), (64)}\\
&= (\bar{q}x -_{f'} \bar{q}y) +_{f'} (\bar{q}x' -_{f'} \bar{q}y') && \text{Lemma C.1 (62)}\\
&= \bar{q}(x -_f y) +_{f'} \bar{q}(x' -_f y') && \text{Lemma C.1 (65).}
\end{aligned}$$

The middle transition in the second chain uses associativity, commutativity, and additive inverses, with subtraction interpreted as addition of the negated operand. Construct the missing local expressions in these chains, reusing the supplied inverse outputs wherever present. Lemma B.4 (50) transfers the incoming equalities through each local block, and the same layer traversal establishes

$$\mathsf{IFFT}^{\mathrm{raw}}_N(\boldsymbol{u} +_{f_0} \boldsymbol{v}) = \mathsf{IFFT}^{\mathrm{raw}}_N(\boldsymbol{u}) +_{f_0+Lt} \mathsf{IFFT}^{\mathrm{raw}}_N(\boldsymbol{v}). \tag{113}$$

Every local expression has $O(W)$-bit numerators. The intermediate sums, local arithmetic and checks, and all local derivations together cost $O(NL)W^{O(1)}$.

Finally, construct the normalized sum on the right of the first transition below and derive, coordinatewise,

$$\begin{aligned}
\mathsf{IFFT}(\boldsymbol{u} +_{f_0} \boldsymbol{v}) &= \left(\mathsf{IFFT}^{\mathrm{raw}}_N(\boldsymbol{u}) +_{f_0+Lt} \mathsf{IFFT}^{\mathrm{raw}}_N(\boldsymbol{v})\right)/N && \text{(113); Lemma B.4 (50)}\\
&= \mathsf{IFFT}(\boldsymbol{u}) +_{f_0+Lt+L} \mathsf{IFFT}(\boldsymbol{v}) && \text{Lemma C.1 (61b).}
\end{aligned}$$

The raw sum is already available. Division by $N = 2^L$ reuses its numerator bits and raises its scale by L, adding no arithmetic gates. The N normalization identities and their missing checks cost $NW^{O(1)}$, including the final folding by Lemma B.2. Thus IFFT additivity has total additional cost $O(NL)W^{O(1)}$.

Scalar linearity. We prove the second identities in (110) and (111), and their conjugated-forward counterpart, by moving the scalar c through one layer at a time.

1. *Construct the intermediate scalar products and checks.* For each of the forward, conjugated-forward, and unnormalized inverse networks, let $\boldsymbol{U}^{(i)}$ name its supplied layer outputs on $\boldsymbol{u}$, and let $M^{(i)}$ name those on $c\boldsymbol{u}$. The scales are $f_i = f_0 + it$ and $f_i + g$, respectively. Construct the products $cU_j^{(i)}$ and the checks needed to prove
$$M_j^{(i)} = cU_j^{(i)} \qquad (0 \le i \le L,\ j < N). \tag{114}$$
Reuse the input products cu_j and any supplied endpoint products. In particular, $c\mathsf{IFFT}_N^{\mathrm{raw}}(\boldsymbol{u})$ is auxiliary, whereas $c\mathsf{IFFT}(\boldsymbol{u})$ is supplied by the conclusion. Each new product uses the existing scalar c and one existing network word, so its numerator width is $O(W)$. All missing products and checks cost $O(NL)W^{O(1)}$; the input identities follow from the input-product wiring at cost $O(NW)$.

2. *Move the scalar through each block.* Let x, y be inputs at scale f, put $f' = f + t$, and let q denote one forward output's hardwired coefficient. After substituting the two incoming instances of (114), derive
$$\begin{aligned} \mathsf{Prmt}_{f'+g}(cx) +_{f'+g} q(cy) &= c\mathsf{Prmt}_{f'}(x) +_{f'+g} c(qy) && \text{Lemma C.1 (85), (63b), (63a)} \\ &= c\big(\mathsf{Prmt}_{f'}(x) +_{f'} qy\big) && \text{Lemma C.1 (64).} \end{aligned}$$
This proves the next scalar invariant for the forward block, including when its coefficient is conjugated. For the inverse block, derive
$$\begin{aligned} \mathsf{Prmt}_{f'+g}(cx +_{f+g} cy) &= \mathsf{Prmt}_{f'+g}\big(c(x +_f y)\big) && \text{Lemma C.1 (64)} \\ &= c\mathsf{Prmt}_{f'}(x +_f y) && \text{Lemma C.1 (85),} \\ \bar{q}(cx -_{f+g} cy) &= \bar{q}\big(c(x -_f y)\big) && \text{Lemma C.1 (65)} \\ &= c\big(\bar{q}(x -_f y)\big) && \text{Lemma C.1 (63b), (63a).} \end{aligned}$$
Construct the missing local arithmetic and equality checks in these chains. Their operands are the existing network wires and the scalar products from Step 1. Substitution through the surrounding local gates uses Lemma B.4 (50), after the width-padding argument from Lemma C.1 where needed. Each chain costs $W^{O(1)}$, including all new gates and inference steps, and every working numerator has width $O(W)$.

   For a fixed power-of-two scalar implemented by wiring, first use Lemma C.1 (60) to compare each local scalar product with its full-multiplier version. Apply the identities above to those multiplier outputs, then transfer back with Lemma B.4. The missing multiplier and check gates and these transfers add $W^{O(1)}$ per local comparison.

   Traversing the L layers establishes (114) at the outputs, with the same sharing of recorded equalities as in the additivity proof. This proves the two forward scalar identities and the raw inverse identity, at total additional cost $O(NL)W^{O(1)}$ for this step.

3. *Normalize the inverse output and collect the conclusions.* The raw inverse invariant gives, coordinatewise,
$$\begin{aligned} \mathsf{IFFT}(c\boldsymbol{u}) &= \frac{c\mathsf{IFFT}_N^{\mathrm{raw}}(\boldsymbol{u})}{N} && \text{(114) at } i = L\text{; Lemma B.4 (50)} \\ &= c\mathsf{IFFT}(\boldsymbol{u}) && \text{Lemma C.1 (61c).} \end{aligned}$$

Only the normalized intermediate products and missing checks are new; the raw products were constructed in Step 1, and both endpoint arrays are supplied. Normalization leaves their numerator widths unchanged. For a power-of-two scalar, use the same multiplier-to-wiring transfers as in Step 2. The N applications of the exact-division identity, equality propagation, and any such transfers cost $NW^{O(1)}$ in total. Use Lemmas B.2 and B.3 to collect the coordinate equalities for all three scalar identities, at cost $O(NW)$. Including all three steps, scalar linearity has total additional cost $O(NL)W^{O(1)}$.

Total cost. The additivity and scalar-linearity proofs together construct $O(NL)$ auxiliary short-word arithmetic blocks and checks and use $O(NL)$ local inference-rule applications. Network outputs, hardwired coefficients, and previously proved layer equalities are reused throughout. All new gates, proof auxiliaries, and small-step inferences therefore have total cost

$$O(NL)W^{O(1)} = N \log^{O(1)} N.$$

This is one shared additional budget for all the stated identities. □

Transpose FFT. To define FFT^T as a circuit, reverse the order of the linear butterfly layers and invert their intervening wiring permutations. For a butterfly pair at positions $j, j+m/2$ of a size-m call, write $q_0 := q_m(j)$ and $q_1 := q_m(j + m/2)$ for its two hardwired coefficients. Its unscaled form is $(x, y) \mapsto (x + q_0 y, x + q_1 y)$. Replace it by $(a, b) \mapsto (a + b, q_0 a + q_1 b)$, implemented on scale-$g$ inputs as

$$(\mathsf{Prmt}_{g+t}(a +_g b), \quad \mathsf{Mul}_{t,g}(q_0, a) +_{g+t} \mathsf{Mul}_{t,g}(q_1, b)) .$$

The component widths are at most $w + t + 1$ and $w + w_q + 2$ when the input widths are w. Conjugating the coefficients adds at most one bit, so the same boundary width $w + \Delta w$ suffices in either case. Sign-extend both outputs to this width. Thus FFT^T has output scale $g + Lt$ and width $w + L\Delta w$. This construction transposes the network as a linear map over the scaled words. Conjugating both coefficients defines $\overline{\mathsf{FFT}}^T$. The implemented inverse in Appendix A.5.3 instead uses $\bar{q}_0(a -_g b)$ in its second branch. The next lemma proves its equality with the conjugate-transpose network using the exact half-period identity for the hardwired constants. The pairing is the bilinear sum circuit of Appendix D.3, without conjugation.

Lemma F.3 (Exact transpose identity in EF)**.** *For arrays $\boldsymbol{u} \in \mathcal{C}_f^N$ and $\boldsymbol{v} \in \mathcal{C}_g^N$ available in Γ, put $f' = f + g + Lt$. Then*

$$\Gamma \ \rhd_{\mathsf{EF}}^{N \log^{O(1)} N}$$
$$\langle \mathsf{FFT}(\boldsymbol{u}), \boldsymbol{v} \rangle_{[f']} = \langle \boldsymbol{u}, \mathsf{FFT}^T(\boldsymbol{v}) \rangle_{[f']},$$

and the same assertion holds with FFT replaced by $\overline{\mathsf{FFT}}$. Also,

$$\Gamma \ \rhd_{\mathsf{EF}}^{N \log^{O(1)} N}$$
$$\mathsf{IFFT}(\boldsymbol{v}) = \frac{1}{N} \overline{\mathsf{FFT}}^T(\boldsymbol{v}), \tag{115a}$$
$$\langle \boldsymbol{u}, \mathsf{IFFT}(\boldsymbol{v}) \rangle_{[f'+L]} = \frac{1}{N} \langle \overline{\mathsf{FFT}}(\boldsymbol{u}), \boldsymbol{v} \rangle_{[f']}. \tag{115b}$$

Division by N is exact scale change. The network equality has scale $g + Lt + L$, and the pairing equality has scale $f' + L$.

Proof. We first move a pairing through the butterfly layers, then identify the conjugated transpose with the unnormalized inverse, and finally combine these two identities. Recall $L = \log N$ and $W = O(L^4)$ from Appendix A.5.4. The network instances, endpoint pairings, and equality checks in the conclusions are already in the expanded context. The constructions below reuse their gates and name new auxiliary circuits explicitly.

Transpose identities. We prove

$$\underbrace{\langle \mathsf{FFT}(\boldsymbol{u}), \boldsymbol{v}\rangle_{[f']}}_{\mathsf{Pair}_L} = \underbrace{\langle \boldsymbol{u}, \mathsf{FFT}^T(\boldsymbol{v})\rangle_{[f']}}_{\mathsf{Pair}_0}, \tag{116}$$

and the same identity for $\overline{\mathsf{FFT}}$. The idea is to move the weights on the forward outputs backward through one butterfly layer at a time, preserving their pairing with the forward values.

1. *Name the existing layer outputs and construct the intermediate pairings.* For $0 \le i \le L$, let $\boldsymbol{U}^{(i)}$ name the array after i forward levels in the supplied $\mathsf{FFT}(\boldsymbol{u})$. Let $\boldsymbol{V}^{(i)}$ name the array after the first $L-i$ transposed levels of the supplied $\mathsf{FFT}^T(\boldsymbol{v})$. These correspond to the last $L-i$ forward levels, traversed in reverse. Match their positions at the same boundary of the forward network, including its fixed wiring permutations. In particular,

$$\boldsymbol{U}^{(0)} = \boldsymbol{u}, \quad \boldsymbol{U}^{(L)} = \mathsf{FFT}(\boldsymbol{u}), \qquad \boldsymbol{V}^{(L)} = \boldsymbol{v}, \quad \boldsymbol{V}^{(0)} = \mathsf{FFT}^T(\boldsymbol{v}).$$

   These are names for existing wires. Their entry scales are $f + it$ and $g + (L-i)t$, respectively. Construct the auxiliary pairing circuits

$$\mathsf{Pair}_i := \langle \boldsymbol{U}^{(i)}, \boldsymbol{V}^{(i)}\rangle_{[f']} \qquad (1 \le i < L).$$

   Also use Pair_L and Pair_0 as names for the two endpoint pairings in (116), already in the context. Every product leaf has scale $(f+it)+(g+(L-i)t) = f'$, so all these sum trees use $+_{f'}$. If w_u, w_v bound the input component widths, each pairing has output width at most $w_u + w_v + L\Delta w + 1 + L = O(W)$, by the network signatures and the full-result product and sum-tree rules. Each new pairing adds N complex products and a balanced sum tree on existing operands, for total construction cost $O(NL)W^{O(1)}$ over the intermediate levels.

2. *Prove the local identity for one butterfly.* Fix a layer $i < L$ and a size-m recursive FFT call at this layer. Choose $0 \le j < m/2$, and consider the butterfly that computes entries j and $j + m/2$ of this call's output array. Its two hardwired coefficients are

$$q_0 := q_m(j), \qquad q_1 := q_m(j + m/2).$$

   Let x, y name this butterfly's forward inputs in $\boldsymbol{U}^{(i)}$, and let a, b name the transpose inputs at the corresponding output positions $j, j + m/2$, among the wires of $\boldsymbol{V}^{(i+1)}$. Put $f_i = f + it$ and $g_i = g + (L-i-1)t$, so $f_i + g_i + t = f'$, and let $\widehat{x} := \mathsf{Prmt}_{f_i+t}(x)$ name its existing promoted input. The two forward outputs are $\widehat{x} +_{f_i+t} q_0 y$ and $\widehat{x} +_{f_i+t} q_1 y$. Construct the subtotal circuit for their contribution to the pairing with a, b:

$$Z_{\text{left}} := (\widehat{x} +_{f_i+t} q_0 y)a +_{f'} (\widehat{x} +_{f_i+t} q_1 y)b.$$

   Construct also the subtotal circuit

$$Z_{\text{right}} := x\,\mathsf{Prmt}_{g_i+t}(a +_{g_i} b) +_{f'} y(q_0 a +_{g_i+t} q_1 b),$$

   and their equality check. The butterfly expressions refer to the existing network outputs, including their prescribed boundary padding. The four outer products are already leaves of Pair_{i+1} and Pair_i; reuse them. Only the subtotal adders and any missing checks are new at this point. We will derive $Z_{\text{left}} = Z_{\text{right}}$.

Construct any missing arithmetic gates in the intermediate expressions of the following chain, using the named operands above:

$$\begin{aligned} Z_{\text{left}} &= \big(\mathsf{Prmt}_{f'}(xa) +_{f'} \mathsf{Prmt}_{f'}(xb)\big) +_{f'} \big(y(q_0 a) +_{f'} y(q_1 b)\big) \\ &= \mathsf{Prmt}_{f'}\big(x(a +_{g_i} b)\big) +_{f'} y(q_0 a +_{g_i+t} q_1 b) \\ &= Z_{\text{right}}. \end{aligned}$$

The first equality uses "Distributivity," associativity and commutativity in "Addition identities" and "Multiplication identities," and promotion of multiplication from Lemma C.1. The second factors the two groups using distributivity and promotion of addition from the same lemma. The third moves the promotion from the product to its second operand, using "Promotion identities and error checks" of that lemma. Apply Lemma B.4 through the surrounding constant-size arithmetic expressions when replacing an equal subexpression. Width-padding identities from Lemma C.1 align the compared numerator words. Each intermediate expression has a fixed number of operations on $O(W)$-bit words; the promotions add at most $t = O(W)$ bits. Thus construction of all missing arithmetic and check gates, and the constant number of lemma applications, costs $W^{O(1)}$ per butterfly. All $O(NL)$ local identities together cost $O(NL)W^{O(1)}$.

3. *Combine the local identities, then traverse the layers.* For each layer, construct two balanced sum trees that group the product leaves of Pair_{i+1} and Pair_i by butterfly. Choose the same order of butterflies on both sides. Their bottom additions are the subtotal circuits just constructed, which are reused. The leaf permutations are fixed by the layer wiring. Lemma D.6 identifies each original pairing with its regrouped tree. Its applications include the construction of any missing permutation and intermediate sum-tree circuits.

   Unfold the local equalities with Lemma B.3, and apply Lemma B.4 successively through corresponding adders above the subtotal outputs. Each child equality is recorded once and reused at its parent. Folding the root's bit equalities with Lemma B.2 gives

$$\mathsf{Pair}_{i+1} = \mathsf{Pair}_i.$$

   The new trees use $O(N)$ adders per layer, and propagation uses $O(N)W^{O(1)}$ additional cost per layer, including its checks. The two applications of Lemma D.6 cost $N \log^{O(1)} N\, W^{O(1)}$ per layer, including their proof auxiliaries. All sums have scale f' and depth $O(L)$, hence width $O(W)$. Finally, chain these scalar equalities by transitivity (55), from $i = L - 1$ down to 0, to obtain the endpoint equality. This last chain costs $O(LW)$; it reuses the recorded equalities without copying their derivations. The total cost of this step over the L layers is $NL \log^{O(1)} N\, W^{O(1)}$.

For the conjugated identity, repeat these steps on the supplied $\overline{\mathsf{FFT}}(\boldsymbol{u})$ and $\overline{\mathsf{FFT}}^T(\boldsymbol{v})$, replacing the local coefficients by their conjugates. Construct its intermediate pairings and local checks once; its network wires are already supplied. The same algebra applies and all widths remain $O(W)$. Including both identities, all new pairing and regrouping circuits, and all local proof blocks, the total additional cost for this group is $NL \log^{O(1)} N\, W^{O(1)} = N \log^{O(1)} N$.

Identification of the inverse network. We next prove

$$\mathsf{IFFT}(\boldsymbol{v}) = \frac{1}{N}\overline{\mathsf{FFT}}^T(\boldsymbol{v}). \tag{117}$$

Both complete networks are already supplied. The unnormalized inverse $\mathsf{IFFT}_N^{\text{raw}}(\boldsymbol{v})$ is the subcircuit of $\mathsf{IFFT}(\boldsymbol{v})$ preceding its final division by N. We first compare it with the supplied conjugated transpose, block by block.

1. *Construct and prove the shared hardwired certificates.* For each $m \geq 2$ and fixed $j < m/2$, construct the closed checks

$$q_m(j + m/2) = -q_m(j), \qquad \overline{q_m(j + m/2)} = -\overline{q_m(j)},$$

using the existing hardwired table entries. The negation, conjugation, and equality gates are new unless already present. These checks are true by the precomputation in Appendix A.4: the two addresses differ only in their highest bit, changing the last selected factor from $\mathsf{Prmt}_p(1)$ to $z_1 = \mathsf{Prmt}_p(-1)$. All products and promotions in that recipe are exact. This observation determines the outputs of the closed checks. To derive their output-one assertions in EF, evaluate their Boolean gates in order, using the gate-evaluation construction in the proof of Lemma E.2. No variable-address twiddle computation is constructed. Each pair has $O(W)$-bit working numerators and costs $W^{O(1)}$ for its new gates and their evaluation. There are $\sum_{s=1}^{L} 2^{s-1} = N - 1$ pairs across the tables, so the total is $NW^{O(1)}$. Later butterfly comparisons share these certificates.

2. *Compare the two local blocks.* Fix a size-m recursive FFT call, $m \geq 2$, and $0 \leq j < m/2$. Consider the backward blocks corresponding to the forward butterfly that produces entries $j, j + m/2$ of this call's output array. Write

$$q_0 := q_m(j), \qquad q_1 := q_m(j + m/2)$$

for that butterfly's two hardwired coefficients; $\bar{q}_0, \bar{q}_1$ denote their complex conjugates. First compare both backward blocks on the same input wires a, b at scale g', corresponding to those two forward output positions. The conjugated transpose returns

$$\mathsf{Prmt}_{g'+t}(a +_{g'} b), \qquad \bar{q}_0 a +_{g'+t} \bar{q}_1 b,$$

whereas the unnormalized inverse returns

$$\mathsf{Prmt}_{g'+t}(a +_{g'} b), \qquad \bar{q}_0(a -_{g'} b).$$

Reuse any of these outputs already present in the supplied networks; construct the missing local comparison formulas and their checks. The first outputs agree. For the second outputs, construct the missing intermediate arithmetic and derive

$$\bar{q}_0 a +_{g'+t} \bar{q}_1 b = \bar{q}_0 a -_{g'+t} \bar{q}_0 b = \bar{q}_0(a -_{g'} b).$$

The first equality substitutes the shared certificate $\bar{q}_1 = -\bar{q}_0$ through the multiplier and adder by Lemma B.4; subtraction distributivity (67) with a zero first term and multiplication by zero give $(-\bar{q}_0)b = -(\bar{q}_0 b)$. The second equality is (65) in reverse. Use the padding identities of Lemma C.1 to compare any differing intermediate widths. These are constant-size arithmetic expressions on $O(W)$-bit inputs, so all missing local gates, checks, and lemma applications cost $W^{O(1)}$ per block.

Proceed through the two supplied networks in their common order of evaluation. Their reversed wiring and recursive interleaving agree. At each block, apply Lemma B.4 to its already proved input equalities, and then use the same-input local identity just established. Any mixed-input local copy needed for this comparison is auxiliary, has a constant number of short-word operations, and is charged once at that block. Record the resulting output equalities for use by the next blocks. Lemmas B.2 and B.3 extract incoming bit equalities and combine the final list into the array assertion

$$\mathsf{IFFT}_N^{\mathrm{raw}}(\boldsymbol{v}) = \overline{\mathsf{FFT}}^{T}(\boldsymbol{v}).$$

There are $O(NL)$ blocks. All comparison copies, local arithmetic identities, padding checks, and small-step inferences therefore cost $O(NL)W^{O(1)}$; the final folding costs $O(NW)$ within this bound.

3. *Apply the shared normalization.* Both normalized outputs in the target equality are already supplied. Division by $N = 2^L$ leaves their numerator bits unchanged and raises both scales from $g + Lt$ to $g+Lt+L$. Thus the equality just proved also identifies the normalized outputs, using the width-padding identities of Lemma C.1 if their encodings differ. Unfolding and folding with Lemmas B.2 and B.3, together with any missing padding checks, costs at most $NW^{O(1)}$. No division or rounding circuit is introduced.

Including the shared certificates and all network-comparison auxiliaries, the total additional cost for this group is $O(NL)W^{O(1)}$.

Moving IFFT through a pairing. Let $\boldsymbol{T}$ name the existing output $\overline{\mathsf{FFT}}^T(\boldsymbol{v})$; this introduces no gates. Reuse the supplied array $\boldsymbol{T}/N$ and the pairings $\langle \boldsymbol{u}, \boldsymbol{T}\rangle_{[f']}$ and $\langle \overline{\mathsf{FFT}}(\boldsymbol{u}), \boldsymbol{v}\rangle_{[f']}$. Construct the intermediate pairing $\langle \boldsymbol{u}, \boldsymbol{T}/N\rangle_{[f'+L]}$, the normalized scalar $N^{-1}\langle \boldsymbol{u}, \boldsymbol{T}\rangle_{[f']}$, and any missing equality checks. The two endpoint expressions below are already supplied. Using the inverse and transpose identities just proved, we derive

$$\begin{aligned}
\langle \boldsymbol{u}, \mathsf{IFFT}(\boldsymbol{v})\rangle_{[f'+L]} &= \langle \boldsymbol{u}, \boldsymbol{T}/N\rangle_{[f'+L]} && \text{(117); Lemma B.4 (50)} \\
&= \frac{1}{N}\langle \boldsymbol{u}, \boldsymbol{T}\rangle_{[f']} && \text{same numerator circuits; Lemma B.4 (50)} \\
&= \frac{1}{N}\langle \overline{\mathsf{FFT}}(\boldsymbol{u}), \boldsymbol{v}\rangle_{[f']} && \text{conjugated (116); Lemma B.4 (50).}
\end{aligned}$$

For the middle transition, division by $N = 2^L$ leaves numerator bits and widths unchanged and increases the scale by L. Construct the new pairing with the same product widths and sum-tree shape as $\langle \boldsymbol{u}, \boldsymbol{T}\rangle_{[f']}$. Their integer numerator circuits then have identical gates and inputs, so equality propagation identifies their outputs. Both sides of this transition have scale $f' + L$. The last transition uses the conjugated transpose equality in reverse and retains its numerator equality after normalization.

The new circuitry is dominated by the intermediate pairing, with N complex products and $N-1$ additions on existing operands, of total size $NW^{O(1)}$. Normalizing the existing scalar output uses only wiring. All working widths remain $O(W)$ by the pairing width bound above, and all three equality checks have scale $f' + L$. The first two equality-propagation applications, including their missing checks and proof auxiliaries, cost $NW^{O(1)}$ each; the last application and joining the three scalar equalities cost $O(W)$. Network outputs and the two previously proved identities are reused. Thus the total additional cost is $NW^{O(1)}$.

Total cost. The two transpose derivations cost $NL\log^{O(1)} N\, W^{O(1)}$, the inverse-network comparison costs $O(NL)W^{O(1)}$, and the final pairing derivation costs $NW^{O(1)}$. These counts include all newly constructed pairings, regrouping and comparison circuits, hardwired checks, and small-step inferences. The supplied network circuits, shared certificates, and previously recorded equalities are reused throughout. Since $L = \log N$ and $W = \log^{O(1)} N$, the shared additional budget is $N\log^{O(1)} N$ for all the conclusions. □

F.3 Network Bounds and Cancellation Identities

For an L-bit index k, define the computed twiddle vector $\boldsymbol{\chi}_k \in \mathcal{C}_t^N$ by

$$(\boldsymbol{\chi}_k)_a := \bar{\mathsf{q}}_N(ak) \qquad (a < N).$$

Lemma F.4 (Network bounds inside EF)**.** *Let $\boldsymbol{u} \in \mathcal{C}_f^N$ and $\boldsymbol{v} \in \mathcal{C}_g^N$ be complex arrays, let $U \in \mathcal{D}_{f_U}$ and $V \in \mathcal{D}_{f_V}$ be real bound words, and let k be an L-bit index. Take the input component widths and the fixed scales f, g, f_U, f_V to be $O(W)$. Then*

$$\Gamma:\ \|\boldsymbol{u}\|_\infty \le U, \quad \|\boldsymbol{v}\|_\infty \le V, \quad 0 \le U, V \le N^{32} \ \rhd_{\mathsf{EF}}^{N\log^{O(1)} N}$$

$$\|\mathsf{FFT}(\boldsymbol{v})\|_\infty, \quad \|\overline{\mathsf{FFT}}(\boldsymbol{v})\|_\infty, \quad \|\mathsf{IFFT}(\boldsymbol{v})\|_\infty \le N^2 V, \tag{118a}$$

$$\mathsf{FFT}(\mathsf{IFFT}(\boldsymbol{v})) \overset{\varepsilon_{\mathrm{tw}} N^8 V}{\approx} \boldsymbol{v}, \qquad \mathsf{IFFT}(\mathsf{FFT}(\boldsymbol{v})) \overset{\varepsilon_{\mathrm{tw}} N^8 V}{\approx} \boldsymbol{v}, \tag{118b}$$

$$\langle \mathsf{FFT}(\boldsymbol{u}), \overline{\mathsf{FFT}}(\boldsymbol{v}) \rangle \overset{\varepsilon_{\mathrm{tw}} N^6 UV}{\approx} N \langle \boldsymbol{u}, \boldsymbol{v} \rangle, \tag{118c}$$

$$\overline{\mathsf{FFT}}(\boldsymbol{e}_k) \overset{\varepsilon_{\mathrm{tw}} N^3}{\approx} \boldsymbol{\chi}_k, \tag{118d}$$

$$\overline{\mathsf{FFT}}(\mathsf{CShift}_k \mathsf{Reflect} \boldsymbol{v}) \overset{\varepsilon_{\mathrm{tw}} N^8 V}{\approx} \boldsymbol{\chi}_k \odot \mathsf{FFT}(\boldsymbol{v}). \tag{118e}$$

Proof. We prove the magnitude bounds, cancel the two networks one layer at a time, and then establish the pairing, impulse, and shifted-reflection estimates. Recall $L = \log N$ and $W = O(L^4)$ from Appendix A.5.4. The networks, routings, pairings, and final checks in the conclusions are supplied. The intermediate circuits described below are auxiliary; repeated expressions reuse their output wires and already derived assertions.

Shared preparation.

1. *Certify the hardwired coefficients.* For every $m \ge 2$ and fixed $j < m/2$, derive

$$|q_m(j)|_1 \le 2, \qquad q_m(j)\bar{q}_m(j) \overset{\varepsilon_{\mathrm{tw}}}{\approx} 1, \qquad q_m(j + m/2) = -q_m(j). \tag{119}$$

   Construct the closed twiddle computations at these addresses and identify their outputs with the supplied constants by Lemma E.1. Apply Lemma E.3, (105a), (105c), and (105e), then transfer the checks to the constants by Lemma B.4 (52). Evaluate the fixed address comparisons. The norm bounds for the second coefficient and both conjugates follow from Lemma C.1, (78b) and (78a). Identify any separately hardwired conjugate with the conjugation circuit by evaluating their closed equality check.

   There are $O(N)$ coefficients across all sizes. Their missing computations, checks, and proofs cost $N \log^{O(1)} N$, including one shared application of Lemma E.2 if its certificates are absent. All working widths are $O(W)$ by Appendix E.1. Every occurrence of a coefficient shares its certificates.

2. *Extract the input bounds.* Apply the AND-tree unfolding assertion of Lemma B.3 (48) to obtain $[|u_j|_1 \le U]_{j<N}$ and $[|v_j|_1 \le V]_{j<N}$. Their checks are existing leaves of the premises; unfolding costs $O(N)$. All recursive calls reuse the appropriate coordinate assertions.

The shared preparation costs $N \log^{O(1)} N$. The following totals count additional work beyond it.

Network magnitude bounds: $\|\mathsf{FFT}(\boldsymbol{v})\|_\infty, \|\overline{\mathsf{FFT}}(\boldsymbol{v})\|_\infty, \|\mathsf{IFFT}(\boldsymbol{v})\|_\infty \le N^2 V$**.**

1. *Bound a single butterfly.* Let x, y be its existing inputs at scale f', with $|x|_1, |y|_1 \le B$, and let q be its hardwired coefficient. Construct the missing norms, bound expressions, and comparison checks. The forward bound is

$$\begin{aligned} |\mathsf{Prmt}_{f'+t}(x) +_{f'+t} qy|_1 &\le |x|_1 + |q|_1 |y|_1 && \text{Lemma C.1 (77), (78c); (88) for } x \approx 0 \\ &\le B + 2B = 3B && \text{(119); Lemma C.1 (73), (74), (66).} \end{aligned} \tag{120}$$

   Apply it to each of the two forward coefficients and to their conjugates. For the two unnormalized inverse outputs, derive

$$\begin{aligned} |\mathsf{Prmt}_{f'+t}(x +_{f'} y)|_1 &\le |x|_1 + |y|_1 \le 2B && \text{Lemma C.1 (77), (88), (73),} \\ |\bar{q}(x -_{f'} y)|_1 &\le |\bar{q}|_1 \big(|x|_1 + |y|_1\big) && \text{Lemma C.1 (78c), (77), (78b)} \\ &\le 4B && \text{(119); Lemma C.1 (78a), (73), (74).} \end{aligned} \tag{121}$$

The arithmetic identities of Lemma C.1 identify the fixed multiples of B; (68) joins each inequality chain. For the shifts computing $2B$ and $4B$, apply (60a) and Lemma B.4 to transfer the product and order identities. Every local construction and derivation costs $W^{O(1)}$. The expressions use a fixed number of operations on $O(W)$-bit words.

2. *Propagate the bounds through the layers.* Construct $3^iV, 4^iV$ once for each $i \le L$ and the norm checks on existing intermediate outputs. Starting from the input bounds, the local calculation gives 3^iV after i forward levels and 4^iV after i unnormalized inverse levels. The final comparisons are

$$\begin{aligned} \|\mathsf{FFT}(\boldsymbol{v})\|_\infty,\ \|\overline{\mathsf{FFT}}(\boldsymbol{v})\|_\infty \le 3^L V \le N^2 V && \text{(120), applied through } L \text{ layers; } 3^L \le N^2\text{; Lemma C.1 (74)}, \\ \|\mathsf{IFFT}(\boldsymbol{v})\|_\infty \le (4^L/N)V \le N^2 V && \text{(121), applied through } L \text{ layers; final division by } N; \ 4^L/N \le N^2\text{; Lemma C.1 (74)}. \end{aligned}$$

Evaluate the two closed coefficient comparisons. Use Lemma C.1 (60) and Lemma B.4 for the fixed power-of-two factors and exact division in these bounds. The transfers cost $W^{O(1)}$ per bound, within the per-layer accounting. Normalization leaves the numerator unchanged and raises its scale by L; its bound check is the undivided check with the same numerator alignment. The fixed-shift comparison argument of Lemma C.1 (87) certifies this identification. Fold the output-coordinate checks using Lemma B.2 (47).

The $O(NL)$ local instances cost $O(NL)W^{O(1)}$ in total, including their new bound circuits and checks. The bound words add only $O(L)$ bits. For each recursive size-m call, let $\boldsymbol{u}', \boldsymbol{v}'$ denote its input subarrays of $\boldsymbol{u}, \boldsymbol{v}$. The same derivation gives

$$\|\mathsf{FFT}_m(\boldsymbol{u}')\|_\infty \le m^2 U, \tag{122a}$$

$$\|\mathsf{FFT}_m(\boldsymbol{v}')\|_\infty, \quad \|\overline{\mathsf{FFT}}_m(\boldsymbol{v}')\|_\infty \le m^2 V. \tag{122b}$$

Construct and retain these checks on the existing recursive outputs, including those of the supplied $\mathsf{FFT}(\boldsymbol{u})$ used in the pairing estimate, at the same total cost $O(NL)W^{O(1)}$.

Thus all magnitude bounds needed here cost $N \log^{O(1)} N$.

The two inverse comparisons: $\mathsf{FFT}(\mathsf{IFFT}(\boldsymbol{v})) \overset{\varepsilon_{\mathrm{tw}} N^8 V}{\approx} \boldsymbol{v}$ and $\mathsf{IFFT}(\mathsf{FFT}(\boldsymbol{v})) \overset{\varepsilon_{\mathrm{tw}} N^8 V}{\approx} \boldsymbol{v}$.

1. *Use paired forward blocks.* For each butterfly pair at positions $j, j + m/2$ of a size-m call, put $q := q_m(j)$. Construct

$$B_q(x, y) := \big(\mathsf{Prmt}_{f'+t}(x) +_{f'+t} qy, \mathsf{Prmt}_{f'+t}(x) -_{f'+t} qy\big)$$

on the existing child outputs. Its first coordinate is already supplied; for the second, derive

$$\begin{aligned} \mathsf{Prmt}_{f'+t}(x) +_{f'+t} q_m(j + m/2)y &= \mathsf{Prmt}_{f'+t}(x) +_{f'+t} (-q)y && \text{(119); Lemma B.4 (50)} \\ &= \mathsf{Prmt}_{f'+t}(x) -_{f'+t} qy && \text{Lemma C.1 (67), (63f), (62).} \end{aligned}$$

The last equality applies subtraction distributivity to $(0 - q)y$. Compare each paired block with its supplied forward block and propagate the input equalities through the next block using Lemma B.4 (50). Keep the original routing wires and sign-extend both block outputs to their original boundary widths. The conjugated version uses the conjugated certificates in the same way.

Each new block, check, and local proof costs $W^{O(1)}$. Constructing the constant number of paired networks for these compositions costs $O(NL)W^{O(1)}$. Their widths obey the same boundary bound $w + \Delta w$ as the supplied networks.

2. *Normalize each inverse layer by two.* Construct an auxiliary inverse using the original routing and the blocks

$$J_q(x,y) := \tfrac{1}{2}\big(\mathsf{Prmt}_{f'+t}(x +_{f'} y),\ \bar{q}(x -_{f'} y)\big).$$

It has no final division by N. Let R_i name the supplied unnormalized inverse output after i levels and $\widetilde{R}_i$ the new output there. For input scale f_0, their scales are $f_0 + it$ and $f_0 + i(t+1)$. The numerator circuits are identical: at each block both input scales have increased by the same i, while division by two changes no bits. Hence

$$\widetilde{R}_i = R_i/2^i \qquad (0 \le i \le L) \qquad \text{by Lemma B.4 (50).}$$

Construct the missing equality checks and compare the gates once in order of evaluation. At $i = L$, the right side is the supplied IFFT output. The new networks and comparisons cost $O(NL)W^{O(1)}$. Their scales increase by at most L extra bits; their numerator widths are unchanged and remain $O(W)$.

3. *Bound the error of cancelling two matched blocks.* For $|x|_1, |y|_1 \le B$, we will prove

$$J_q B_q(x,y) \overset{\varepsilon_{\mathrm{tw}} B}{\approx} (x,y), \qquad B_q J_q(x,y) \overset{\varepsilon_{\mathrm{tw}} B}{\approx} (x,y). \tag{123}$$

Construct the two local compositions, the defect $d_q := q\bar{q} - 1$, the half-difference $r_{x,y} := (x-y)/2$, and the missing arithmetic and checks in the following identities. Reuse $q, \bar{q}$ and their certificates. With the omitted-scale convention, derive

$$\begin{aligned}
J_q B_q(x,y) &= \tfrac{1}{2}\big(2x,\ \bar{q}(2qy)\big) && \text{definitions of } J_q, B_q;\ \text{Lemma C.1 (62), (64), (65)}\\
&= \big(x,\ q\bar{q}\,y\big) && \text{Lemma C.1 (63); (60a), (61a), (61c),}\\
B_q J_q(x,y) &= \tfrac{1}{2}\big(x+y+q\bar{q}(x-y),\ x+y-q\bar{q}(x-y)\big) && \text{definitions of } B_q, J_q;\ \text{Lemma C.1 (63b)}\\
&= \big(x + d_q r_{x,y},\ y - d_q r_{x,y}\big) && \text{Lemma C.1 (62), (66), (67).}
\end{aligned}$$

The last transition substitutes $q\bar{q} = 1 + d_q$ and cancels opposite terms. The actual composition outputs have scale $f' + 2t + 1$; d_q has scale $2t$ and $r_{x,y}$ has scale $f' + 1$. Promote the other terms to that scale using Lemma C.1, (83)–(86). Equality propagation (50) of Lemma B.4 applies through each enclosing expression. The factors of two use (60a), followed by Lemma B.4, when applying multiplication identities. Moving exact division through sums and products and cancelling the factor two use (61). For the norm inequalities below, (60b) transfers the multiplication and order bounds to exact division by two. Each such local transfer, including any missing auxiliary gates, costs $W^{O(1)}$.

The error calculations are

$$\begin{aligned}
|r_{x,y}|_1 &\le \tfrac{1}{2}\big(|x|_1 + |y|_1\big) \le B && \text{Lemma C.1 (77), (78b), (73),}\\
|d_q y|_1 &\le |d_q|_1 |y|_1 \le \varepsilon_{\mathrm{tw}} B && \text{(119); Lemma C.1 (78c), (74),}\\
|d_q r_{x,y}|_1 &\le |d_q|_1 |r_{x,y}|_1 \le \varepsilon_{\mathrm{tw}} B && \text{Lemma C.1 (78c), (74).}
\end{aligned}$$

Subtraction distributivity (67) identifies $q\bar{q}\,y - y = d_q y$. Substitute the exact identities into the difference checks and use Lemma C.1, (62) and (78b), to obtain (123). The comparison scale also accommodates $\varepsilon_{\mathrm{tw}} B$. Under the same input bounds $|x|_1, |y|_1 \le B$, the single-butterfly calculations give

$$\|B_q(x,y)\|_\infty \le 3B \qquad \text{(120)}, \tag{124a}$$

$$\|J_q(x,y)\|_\infty \le 2B \qquad \text{(121), division by two.} \tag{124b}$$

For the second bound, exact division by two changes the component bounds $2B, 4B$ to $B, 2B$; Lemma C.1 (87) identifies the corresponding checks after the scale change. Fold the two component bounds using Lemma B.2 (47).

All these local circuits and derivations cost $W^{O(1)}$ per instance, with $O(W)$-bit working numerators. The next step specifies the instances.

4. *Cancel one matched layer at a time.* Let B_i denote the i-th paired forward layer and J_i its matched reversed layer normalized by $1/2$, including the routing wires. Construct the missing shortened compositions

$$\begin{aligned} \boldsymbol{H}_r &:= J_1 \cdots J_{L-r} B_{L-r} \cdots B_1(\boldsymbol{v}), \\ \boldsymbol{K}_r &:= B_L \cdots B_{r+1} J_{r+1} \cdots J_L(\boldsymbol{v}) \qquad (0 \le r \le L). \end{aligned}$$

Composition is read from right to left. The existing $\boldsymbol{H}_0, \boldsymbol{K}_0$ have been identified with $\mathsf{IFFT}(\mathsf{FFT}(\boldsymbol{v})), \mathsf{FFT}(\mathsf{IFFT}(\boldsymbol{v}))$, and $\boldsymbol{H}_L = \boldsymbol{K}_L = \boldsymbol{v}$. Propagate the network equalities from the first two steps through the outer network by Lemma B.4 (50); this adds $O(NL)W^{O(1)}$. Reuse identical prefixes; even constructing every shortened composition separately uses only $O(NL^2)$ blocks. Put $g_1 := g + 2Lt + L$ and $g_2 := \max\{g_1, p + f_V\}$. Promote the outputs to g_1, naming them $\widehat{\boldsymbol{H}}_r, \widehat{\boldsymbol{K}}_r$, and construct their consecutive error checks at scale g_2.

For $i = L - r$, the removed pair in $\boldsymbol{H}_r$ receives inputs bounded by $3^{i-1}V$, and its error passes through $i - 1$ remaining inverse layers. For $\boldsymbol{K}_r$ the corresponding counts are $L - r - 1$ inverse layers before cancellation and as many forward layers afterwards. Thus

$$\begin{aligned} \|\widehat{\boldsymbol{H}}_r - \widehat{\boldsymbol{H}}_{r+1}\|_\infty &\le 2^{i-1}\big(\varepsilon_{\mathrm{tw}} 3^{i-1} V\big) \le 6^L \varepsilon_{\mathrm{tw}} V && \text{(123); (124a), (124b); Lemma C.1 (74)}, \\ \|\widehat{\boldsymbol{K}}_r - \widehat{\boldsymbol{K}}_{r+1}\|_\infty &\le 3^{L-r-1}\big(\varepsilon_{\mathrm{tw}} 2^{L-r-1} V\big) \le 6^L \varepsilon_{\mathrm{tw}} V && \text{(123); (124b), (124a); Lemma C.1 (74)}. \end{aligned}$$

To propagate an error through a remaining block, construct that block on the difference of its two inputs. Distributivity and addition identities of Lemma C.1, (65) and (62), identify it with the difference of the block outputs; then apply the appropriate bound in (124). Use Lemma B.4 (50) once per block, retaining its output assertion for the next layer. Promotion identities (83)–(86) and error-check equivalence (89) of Lemma C.1 align these comparisons at g_2.

The shortened and difference networks, local cancellations, bound checks, and propagation proofs cost $O(NL^2)W^{O(1)}$. Each path has at most $2L$ butterfly layers; the promotions and the hardwired powers $2^i, 3^i, 6^L$ keep all widths and scales $O(W)$.

5. *Combine the consecutive errors.* For either chain, the final calculation is

$$\begin{aligned} \|\widehat{\boldsymbol{H}}_0 - \mathsf{Prmt}_{g_1}(\boldsymbol{v})\|_\infty &\le L 6^L \varepsilon_{\mathrm{tw}} V && \text{Lemma C.1 (80), } L \text{ times} \\ &\le \varepsilon_{\mathrm{tw}} N^8 V && L6^L \le N^8; \text{ Lemma C.1 (74), (68)}. \end{aligned}$$

All checks use scale g_2; the same calculation holds for $\widehat{\boldsymbol{K}}_0$. Evaluate the closed coefficient inequality, construct the accumulated tolerance words, and substitute the already proved identities for $\boldsymbol{H}_0, \boldsymbol{K}_0$ using Lemma B.4 (52). Lemma C.1 (89) removes the explicit comparison scale and endpoint promotions. The $O(L)$ vector combinations and final conversions cost $O(NL)W^{O(1)}$; the tolerance sums add at most L bits.

Both inverse comparisons, including all new networks and checks, therefore cost $O(NL^2)W^{O(1)} = N \log^{O(1)} N$.

The pairing estimate: $\langle \mathsf{FFT}(\boldsymbol{u}), \overline{\mathsf{FFT}}(\boldsymbol{v})\rangle \overset{\varepsilon_{\mathrm{tw}} N^6 UV}{\approx} N\langle \boldsymbol{u}, \boldsymbol{v}\rangle$**.**

1. *Construct the recursive pairings.* At a size-$m = 2^s$ call, let $\boldsymbol{u}', \boldsymbol{v}'$ be its fixed input subarrays and put
$$Q_m := \langle \mathsf{FFT}_m(\boldsymbol{u}'), \overline{\mathsf{FFT}}_m(\boldsymbol{v}')\rangle, \qquad S_m := \langle \boldsymbol{u}', \boldsymbol{v}'\rangle, \qquad \varepsilon_m := \varepsilon_{\mathrm{tw}} m^6 UV.$$
Starting with size one, we prove
$$Q_m \overset{\varepsilon_m}{\approx} mS_m. \tag{125}$$
The base $m = 1$ has $Q_1 = S_1$ and zero error. The FFT arrays are existing subcircuits; construct the missing pairings and error checks, reusing the supplied root operands Q_N, NS_N. Construct $\varepsilon_{\mathrm{tw}} UV$ once and ε_m once per size. Since $m^6 = 2^{6s}$, each tolerance ε_m is obtained by shifting the numerator of $\varepsilon_{\mathrm{tw}} UV$ by $6s \le 6L$ bits. Its width thus grows by only $O(L)$, and its scale is unchanged. Write $f_s := f + st$ and $g_s := g + st$, so the data scale is $f_s + g_s$. The tolerance scale is $p + f_U + f_V$, and we compare at $f'_s := \max\{f_s + g_s,\ p + f_U + f_V\}$.

   The calls at any size contain N entries in total. These new products, sums, bound words, and checks cost $O(NL)W^{O(1)}$. Products add operand widths, balanced sums add at most L bits, and promotions add $O(W)$ bits, so all widths are $O(W)$.

2. *Identify each butterfly pair's contribution.* Apply Step 1 of the inverse-comparison argument to the forward and conjugated networks on $\boldsymbol{u}, \boldsymbol{v}$, constructing the missing paired blocks and their comparisons. Let x, y and x', y' be the existing child outputs for one pair, at scales f_{s-1} and g_{s-1}. Construct the missing local products and sums in
$$\begin{aligned}
&(x + qy)(x' + \bar{q}y') + (x - qy)(x' - \bar{q}y') \\
&\quad = 2xx' + 2q\bar{q}\, yy' && \text{Lemma C.1 (64)–(67); (62), (63)} \\
&\quad = 2xx' + 2yy' + 2(q\bar{q} - 1)yy' && \text{Lemma C.1 (66), (62), (63d).}
\end{aligned}$$
   The first transition cancels the two cross terms; the second writes $q\bar{q} = 1 + (q\bar{q} - 1)$. The factors of two are shifts; Lemma C.1 (60a) and Lemma B.4 transfer the multiplication identities to them, at $W^{O(1)}$ cost per pair. The input promotions are those of B_q, and all terms are compared at $f_{s-1} + g_{s-1} + 2t = f_s + g_s$, using the promotion identities (83)–(86) of Lemma C.1. Use Lemma B.4 (50) through the local products and sums to identify the paired-block products with the supplied pairing leaves. There are $O(NL)$ such constant-size arithmetic calculations. Their new circuits and proofs cost $O(NL)W^{O(1)}$, with $O(W)$ widths.

3. *Combine the child estimates.* Let $m \ge 2$, and let $Q_{\mathrm{e}}, Q_{\mathrm{o}}, S_{\mathrm{e}}, S_{\mathrm{o}}$ name the already constructed child pairings. The instances of (125) already derived for the even and odd size-$m/2$ calls are
$$Q_{\mathrm{e}} \overset{\varepsilon_{m/2}}{\approx}_{f'_{s-1}} (m/2)S_{\mathrm{e}}, \qquad Q_{\mathrm{o}} \overset{\varepsilon_{m/2}}{\approx}_{f'_{s-1}} (m/2)S_{\mathrm{o}}. \tag{126}$$
   Construct the sum
$$\Delta_m := \sum_{j < m/2} 2(q_m(j)\bar{q}_m(j) - 1)y_j y'_j$$
   from the defect leaves in Step 2; $y := \mathsf{FFT}_{m/2}(\boldsymbol{u}'^{\mathrm{o}})$ and $y' := \overline{\mathsf{FFT}}_{m/2}(\boldsymbol{v}'^{\mathrm{o}})$ name the existing outputs of the calls on the odd-position inputs. The recursive calculation is
$$\begin{aligned}
Q_m &= 2Q_{\mathrm{e}} + 2Q_{\mathrm{o}} + \Delta_m && \text{Step 2; Lemmas D.5 and D.6; Lemma C.1 (64), (62)} \\
&\overset{4\varepsilon_{m/2}}{\approx}_{f'_s} m(S_{\mathrm{e}} + S_{\mathrm{o}}) + \Delta_m && \overset{(126)}{\text{Lemma C.1 (77), (78c), (74)}} \\
&\overset{2(m/2)^5\varepsilon_{\mathrm{tw}}UV}{\approx}_{f'_s} m(S_{\mathrm{e}} + S_{\mathrm{o}}) && \text{the defect bound below; Lemma C.1 (62)} \\
&= mS_m && \text{Lemma D.6.}
\end{aligned}$$

The first transition regroups pairing leaves by butterfly, applies the local identities, and separates the three sums node by node. The last uses the fixed even/odd permutation of the input products to identify $S_m = S_{\mathrm{e}} + S_{\mathrm{o}}$. For each tree, Lemma B.4 (50) uses the two recorded child equalities once at their parent. The second transition doubles each error in (126) and adds the two bounds using Lemma C.1 (73); (88) aligns their checks. Here and in the defect calculation below, multiplication by 2, $m/2$, or a fixed power of $m/2$ uses shifts. Apply Lemma C.1 (60a) and Lemma B.4 whenever using distributivity, associativity, or a product or order bound on these operations. The transfers cost $W^{O(1)}$ per short-word calculation.

Apply (122a) and (122b) to these size-$m/2$ calls on $\boldsymbol{u}'^{\mathrm{o}}, \boldsymbol{v}'^{\mathrm{o}}$. Unfold their vector-bound checks by Lemma B.3 (48) to obtain

$$|y_j|_1 \le (m/2)^2 U, \qquad |y'_j|_1 \le (m/2)^2 V \qquad (j < m/2). \tag{127}$$

This reuses the existing coordinate checks and adds $O(m)$ small-step inferences. Hence

$$\begin{aligned} |\Delta_m|_1 &\le \sum_{j<m/2} 2|q_m(j)\bar{q}_m(j) - 1|_1 |y_j|_1 |y'_j|_1 && \text{Lemma C.1 (77), (78c)} \\ &\le \sum_{j<m/2} 2\varepsilon_{\mathrm{tw}} (m/2)^4 UV && \text{(119), (127); Lemma C.1 (74), (73)} \\ &= 2(m/2)^5 \varepsilon_{\mathrm{tw}} UV && \text{Lemma C.1 (66).} \end{aligned}$$

Combine the two errors by Lemma C.1 (79), then weaken the tolerance using

$$4\varepsilon_{m/2} + 2(m/2)^5 \varepsilon_{\mathrm{tw}} UV = \big(4(m/2)^6 + 2(m/2)^5\big)\varepsilon_{\mathrm{tw}} UV \le m^6 \varepsilon_{\mathrm{tw}} UV = \varepsilon_m.$$

The equality uses distributivity; evaluate the closed coefficient inequality and apply (74) and (68) of Lemma C.1.

The new circuitry consists of the regrouped sum trees, Δ_m, intermediate sums in the chain, tolerance words, and their checks. It costs $mW^{O(1)}$ apart from the regrouping auxiliaries. Including Lemma D.6, all new gates and derivations cost $m \log^{O(1)} m\, W^{O(1)}$ per call. The balanced sums add $O(\log m)$ bits and all comparisons use f'_s, so their widths remain $O(W)$.

4. *Return the root estimate.* The root operands are the supplied pairing outputs. Apply Lemma C.1 (88) to return from f'_L to the default comparison scale, at cost $W^{O(1)}$ including any missing scalar-check gates.

The total additional cost is

$$O(NL)W^{O(1)} + \sum_{s=0}^{L} \frac{N}{2^s} 2^s \log^{O(1)}(2^s+1) W^{O(1)} = N \log^{O(1)} N.$$

Each child pairing and its derived estimate is shared with its parent.

The impulse estimate: $\overline{\mathsf{FFT}}(\boldsymbol{e}_k) \overset{\varepsilon_{\mathrm{tw}} N^3}{\approx} \boldsymbol{\chi}_k$.

1. *Compare one selected row with a direct sum.* Let a be an arbitrary L-bit row index (Section 3.2). Construct the selector on the supplied $\overline{\mathsf{FFT}}(\boldsymbol{e}_k)$, the conjugated direct row $\bar{D}_a(\boldsymbol{e}_k)$, and its check. The

decoder bits give $\|\boldsymbol{e}_k\|_\infty \le 1$: evaluate their two possible absolute values, use the complex-embedding assertion (81) of Lemma C.1, and fold with Lemma B.2 (47). Then

$$\mathsf{Sel}(\overline{\mathsf{FFT}}(\boldsymbol{e}_k), a) \overset{\varepsilon_{\mathrm{tw}} N^3}{\approx} \bar{D}_a(\boldsymbol{e}_k) \qquad \text{by Lemma F.1, conjugated version.}$$

The decoder proof, direct row, selector, and row-lemma instance cost $N \log^{O(1)} N$. All new a-dependent gates and proof steps are included; the network and shared root certificates are reused. The working widths are $O(W)$ by the row construction.

2. *Reduce the direct sum to one twiddle.* Let $w_j := \bar{\mathsf{q}}_N(aj)$ name the existing direct-row twiddles. Construct $\mathsf{Sel}(\boldsymbol{w}, k)$, the single twiddle $\bar{\mathsf{q}}_N(ak)$ and its modular address, and the missing pairing and equality checks. Derive

$$\begin{aligned} \bar{D}_a(\boldsymbol{e}_k) &= \langle \boldsymbol{e}_k, \boldsymbol{w} \rangle && \text{Lemma C.1 (63a); Lemma B.4 (50)} \\ &= \mathsf{Sel}(\boldsymbol{w}, k) && \text{Lemma D.11} \\ &= \bar{\mathsf{q}}_N(ak) && \text{Lemma B.6; Lemma B.5 (57)} \\ &= \mathsf{Sel}(\boldsymbol{\chi}_k, a) && \text{Lemma B.6; Lemma B.5 (57).} \end{aligned}$$

For the third transition, under each fixed case $k = j$, the selector returns w_j, and Lemma B.4 (51) identifies the short address and twiddle computation with that same w_j. For the last, use the cases $a = j$ to identify both outputs with the supplied $(\boldsymbol{\chi}_k)_j$. These are two separate lists of N cases.

The new selectors, pairing, twiddle, and checks cost $N \log^{O(1)} N$. Each fixed-address proof costs $O(W \log(N+1))$ and each short-circuit substitution costs $\log^{O(1)} N$. The two enumerations and the decoder pairing proof keep the total cost of this step at $N \log^{O(1)} N$. All working widths are $O(W)$; neither full array is rebuilt per case.

3. *Apply index-generalization to the row comparison.* Use Lemma B.4 (52) through the row-error check to obtain

$$\mathsf{Sel}(\overline{\mathsf{FFT}}(\boldsymbol{e}_k), a) \overset{\varepsilon_{\mathrm{tw}} N^3}{\approx} \mathsf{Sel}(\boldsymbol{\chi}_k, a).$$

The predicate after its two selectors has size $W^{O(1)}$. The derivation above works at an arbitrary row index a, as index-generalization (Lemma B.8) requires. The direct row, $\mathsf{Sel}(\boldsymbol{w}, k)$, twiddle, address cases, and all their proofs belong to the indexed budget $S = N \log^{O(1)} N$. That lemma applies this indexed derivation once and gives all coordinate comparisons at cost $S + NW^{O(1)} + O(NL)$. Fold by Lemma B.2 (47); any comparison-scale conversion uses Lemma C.1 (89) at cost $NW^{O(1)}$.

The impulse estimate therefore costs $N \log^{O(1)} N$, including the single application of the indexed derivation and all its auxiliary circuits.

The shifted-reflection estimate: $\overline{\mathsf{FFT}}(\mathsf{CShift}_k \mathsf{Reflect}\, \boldsymbol{v}) \overset{\varepsilon_{\mathrm{tw}} N^8 V}{\approx} \boldsymbol{\chi}_k \odot \mathsf{FFT}(\boldsymbol{v})$.

1. *Construct the two selected-row comparisons.* Let a be an arbitrary L-bit row index. Let $\boldsymbol{z}$ name the supplied routing output $\mathsf{CShift}_k \mathsf{Reflect}\, \boldsymbol{v}$. Construct $\bar{D}_a(\boldsymbol{z}), D_a(\boldsymbol{v})$, the selectors on the two supplied transform outputs, and the missing checks. Lemma D.4 (101) gives $\|\boldsymbol{z}\|_\infty \le V$, so

$$\mathsf{Sel}(\overline{\mathsf{FFT}}(\boldsymbol{z}), a) \overset{\varepsilon_{\mathrm{tw}} N^3 V}{\approx} \bar{D}_a(\boldsymbol{z}) \qquad \text{Lemma F.1, conjugated version,} \tag{128a}$$

$$D_a(\boldsymbol{v}) \overset{\varepsilon_{\mathrm{tw}} N^3 V}{\approx} \mathsf{Sel}(\mathsf{FFT}(\boldsymbol{v}), a) \qquad \text{Lemma F.1; Lemma C.1 (78b).} \tag{128b}$$

The second comparison reverses the row lemma's error using $D_a(\boldsymbol{v}) - \mathsf{Sel}(\mathsf{FFT}(\boldsymbol{v}), a) = -(\mathsf{Sel}(\mathsf{FFT}(\boldsymbol{v}), a) - D_a(\boldsymbol{v}))$, from Lemma C.1 (62). The routing proof, two direct rows, selectors, checks, and row-lemma instances cost $N \log^{O(1)} N$, with $O(W)$ widths. The transforms and root certificates are reused; all other a-dependent circuitry is charged to this indexed proof.

2. *Route the first direct sum back to the original data.* Let $w_j := \bar{\mathsf{q}}_N(aj)$ name the existing twiddles in $\bar{D}_a(\boldsymbol{z})$. Construct the two routed arrays below, their pairings, and the missing equality checks. We derive

$$\begin{aligned} \bar{D}_a(\boldsymbol{z}) &= \langle \mathsf{CShift}_k \mathsf{Reflect}\, \boldsymbol{w}, \mathsf{CShift}_k \mathsf{Reflect}\, \boldsymbol{z} \rangle && \text{Lemma D.10 (102b)} \\ &= \langle \mathsf{CShift}_k \mathsf{Reflect}\, \boldsymbol{w}, \boldsymbol{v} \rangle && \text{Lemma D.3; Lemma B.4 (50)} \\ &= \underbrace{\sum_{j<N}^{[g+t]} \bar{\mathsf{q}}_N(a(k-j)) v_j}_{T_a} && \text{the coordinate identification below.} \end{aligned} \tag{129}$$

Here $\bar{D}_a(\boldsymbol{z}) = \langle \boldsymbol{w}, \boldsymbol{z} \rangle$ by definition. The routing and pairing constructions and their first two comparisons cost $N \log^{O(1)} N$.

For the final transition, at an arbitrary L-bit index i, derive

$$\begin{aligned} \mathsf{Sel}(\mathsf{CShift}_k \mathsf{Reflect}\, \boldsymbol{w}, i) &= \mathsf{Sel}(\mathsf{Reflect}\, \boldsymbol{w}, i-k) && \text{Lemma D.2 (98)} \\ &= \mathsf{Sel}(\boldsymbol{w}, k-i) && \text{Lemma D.2 (99); Lemma C.1 (62), (90)} \\ &= \bar{\mathsf{q}}_N(a(k-i)) && \text{Lemma B.6; Lemma B.5 (57).} \end{aligned}$$

All addresses are modulo N. For the last equality use the N cases $k - i = j$, just as in Step 2 of the impulse argument: conditional equality propagation (51) of Lemma B.4 identifies the computed twiddle with w_j. Construct the missing selectors, short address and twiddle circuits, and checks. This indexed derivation costs $N \log^{O(1)} N$; its local predicate, including the twiddle on the right, has size $\log^{O(1)} N$.

Apply index-generalization (Lemma B.8) once to obtain the coordinate equalities. Its base context contains $a, k, \boldsymbol{w}$ and the routed arrays; every i-dependent circuit and proof is charged to its budget. Construct the array $(\bar{\mathsf{q}}_N(a(k-j)))_{j<N}$ and T_a if absent, at cost $N \log^{O(1)} N$. Fold the coordinate checks by Lemma B.2 (47) and propagate their equality through the pairing by Lemma B.4 (50). This completes the final transition.

The whole step, including the single inner application of index-generalization, costs $N \log^{O(1)} N$. Products have scale $g + t$, sums add at most L bits, and routing preserves entry types, so all widths are $O(W)$. This entire step remains part of the outer a-dependent proof.

3. *Factor out the twiddle for the selected row.* Construct the single factor $c := \bar{\mathsf{q}}_N(ak)$ and its address if missing; reuse it for every coordinate. By Lemma E.3 (105a) and Lemma C.1 (78a), $|c|_1 \le 2$. For each j, let $b := aj \bmod N$ and $h := ak \bmod N$ name the existing addresses in $D_a(\boldsymbol{v})$ and c. Derive

$$\begin{aligned} \bar{\mathsf{q}}_N(a(k-j)) &= \bar{\mathsf{q}}_N(h-b) && \text{Lemma C.1 (65), (90a), (91); Lemma B.4 (50)} \\ &\overset{\varepsilon_{\mathrm{tw}}}{\approx} c\bar{\mathsf{q}}_N(-b) && \text{Lemma E.3 (105b); Lemma C.1 (78a)} \\ &\overset{2\varepsilon_{\mathrm{tw}}}{\approx} c\mathsf{q}_N(b) && \text{Lemma E.3 (105d); } |c|_1 \le 2\text{; Lemma C.1 (78c), (74).} \end{aligned}$$

The first equality uses $a(k-j) = ak - aj$ and removes multiples of N from the two operands. The first approximate transition conjugates the product law at $(h, -b)$ and reverses it. To identify the

conjugated product, expand the complex components and use Lemma C.1, (62), (63), and (64), giving $\overline{xy} = \bar{x}\bar{y}$. For the last transition use the conjugation law at $-b$; the addition and remainder identities (62), (90a), and (91) identify $-(-b)$ with b modulo N. Subtraction distributivity (65) factors out c from the difference, and its norm bound multiplies the error by at most two.

Construct all missing twiddles, products, conjugations, and checks in these N instances. Each costs $\log^{O(1)} N$, including its proof; the root certificates and the single c computation are shared. Combine the two errors using Lemma C.1 (79) to obtain tolerance $3\varepsilon_{\mathrm{tw}}$. Now construct the factored sum below from the existing leaves of $D_a(\boldsymbol{v})$. The resulting calculation is

$$\begin{aligned} T_a &\overset{3\varepsilon_{\mathrm{tw}}NV}{\approx} \sum_{j<N} c\big(\mathsf{q}_N(aj)v_j\big) && \text{the coordinate comparisons; } |v_j|_1 \le V; \text{ Lemma C.1 (78c), (77), (73)} \\ &= cD_a(\boldsymbol{v}) && \text{Lemma C.1 (63b), (64); Lemma B.4 (50)}. \end{aligned} \tag{130}$$

For the approximate transition, subtraction distributivity (67) turns each summand difference into its twiddle difference times v_j, of norm at most $3\varepsilon_{\mathrm{tw}}V$. Promote the checks to $\max\{g+2t, p+f_V\}$ using Lemma C.1 (88), then add the errors once per sum-tree node. For the exact transition construct c times each existing node output of $D_a(\boldsymbol{v})$ and apply distributivity once per node, sharing the two child equalities.

These new sum and tolerance circuits, checks, and node proofs add $NW^{O(1)}$. Each product contains one data entry and at most two twiddles, and summation adds L bits, so all widths remain $O(W)$. The total cost of this step is $N\log^{O(1)} N$.

4. *Combine the rows and return the array comparison.* Put $g_1 := \max\{g+t, p+f_V\} + (L+1)t$ and construct the missing scalar checks at this common scale. We have

$$\begin{aligned} \mathsf{Sel}(\overline{\mathsf{FFT}}(z), a) &\overset{\varepsilon_{\mathrm{tw}}N^3V}{\approx}_{g_1} \bar{D}_a(z) && \text{(128a); Lemma C.1 (88)} \\ &= T_a && \text{(129)} \\ &\overset{3\varepsilon_{\mathrm{tw}}NV}{\approx}_{g_1} cD_a(\boldsymbol{v}) && \text{(130); Lemma C.1 (88)} \\ &\overset{2\varepsilon_{\mathrm{tw}}N^3V}{\approx}_{g_1} c\mathsf{Sel}(\mathsf{FFT}(\boldsymbol{v}), a) && \text{(128b); } |c|_1 \le 2; \text{ Lemma C.1 (78c), (74)} \\ &= \mathsf{Sel}(\boldsymbol{\chi}_k \odot \mathsf{FFT}(\boldsymbol{v}), a) && \text{Lemma B.6; Lemma B.5 (57)}. \end{aligned}$$

Apply Lemma C.1 (88) to (128a) and (130), respectively, to obtain the first and third comparisons at g_1 with the same error bounds. For the fourth, first promote (128b) to $g_1 - t$ using the same lemma; multiplication by the scale-t value c then has scale g_1. Use subtraction distributivity (65) to factor the difference before applying the norm bound. These scalar checks and derivations cost $W^{O(1)}$ each.

For the last equality, construct the missing selectors on $\boldsymbol{\chi}_k$ and the supplied pointwise-product array. Under each fixed case $a = j$, Lemma B.6 returns their coordinates, while Lemma B.4 (51) identifies c with $(\boldsymbol{\chi}_k)_j$ and propagates this through the multiplier. Enumeration combines the N cases. The selectors, checks, and proofs cost $N\log^{O(1)} N$; the pointwise products are reused.

Lemma C.1 (79) combines the three errors. Evaluate $3N^3 + 3N \le N^8$ and use (74) and (68) to obtain the selected comparison with tolerance $\varepsilon_{\mathrm{tw}}N^8V$. Its predicate after the two selectors has size $W^{O(1)}$. Apply index-generalization (Lemma B.8) once with indexed budget $S = N\log^{O(1)} N$, including every a-dependent construction and proof above, especially the entire inner application in Step 2. The derivation above works at an arbitrary row index a, as that lemma requires.

This single outer instantiation costs $S + NW^{O(1)} + O(NL)$. Fold the coordinate assertions by Lemma B.2 (47) and return to the default scale by Lemma C.1 (89), at additional cost $NW^{O(1)}$. All comparison scales and widths are $O(W)$.

The shifted-reflection estimate costs $N \log^{O(1)} N$, including both applications of index-generalization. Only the two direct rows at the arbitrary index a are constructed.

Total cost. The shared preparation costs $N \log^{O(1)} N$. The additional costs are $O(NL)W^{O(1)}$ for the magnitude bounds, $O(NL^2)W^{O(1)}$ for both inverse comparisons, and $N \log^{O(1)} N$ for each of the pairing, impulse, and shifted-reflection estimates. These counts include all new circuits, checks, local derivations, and indexed-proof copies. With $L = \log N$ and $W = \log^{O(1)} N$, the shared total is $N \log^{O(1)} N$. □

G The Rounded Coefficient Theorem

For an L-bit index k, define one exact integer convolution coefficient by

$$\mathsf{Conv}_k(\boldsymbol{a}, \boldsymbol{b}) := \langle \boldsymbol{a}, \mathsf{CShift}_k \mathsf{Reflect} \boldsymbol{b} \rangle_{[0]} = \sum_{j<N}^{[0]} a_j b_{k-j \bmod N}.$$

This circuit uses barrel routing, N short multipliers, and a balanced sum tree, so its size is $N \log^{O(1)} N$. Let

$$M := N^8, \quad \varepsilon_{\mathrm{out}} := \varepsilon_{\mathrm{tw}} N^{32}.$$

Evaluating the closed parameter checks proves $6N^{14}\varepsilon_{\mathrm{out}} < 1/4$. The next lemma compares one selected FFT output with this integer coefficient; rounding then gives exact equality.

Lemma G.1 (A selected coefficient, including the rounding gate)**.** *Let $\boldsymbol{a}, \boldsymbol{b} \in \mathcal{D}_0^N$ have $O(W)$-bit components, and let k be an L-bit index. With $\mathcal{R} := \{\|\boldsymbol{a}\|_\infty \le M,\ \|\boldsymbol{b}\|_\infty \le M\}$, we have*

$$\Gamma : \mathcal{R} \ \triangleright_{\mathsf{EF}}^{N \log^{O(1)} N}$$

$$\mathsf{Sel}(\mathsf{FFTConv}^{\mathrm{raw}}(\boldsymbol{a}, \boldsymbol{b}), k) \overset{\varepsilon_{\mathrm{out}}}{\approx} \mathsf{Conv}_k(\boldsymbol{a}, \boldsymbol{b}), \tag{131a}$$

$$\mathsf{Sel}(\mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{a}, \boldsymbol{b}), k) = \mathsf{Conv}_k(\boldsymbol{a}, \boldsymbol{b}). \tag{131b}$$

Also,

$$\Gamma : \mathcal{R} \ \triangleright_{\mathsf{EF}}^{N \log^{O(1)} N} \ \mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{a}, \boldsymbol{b}) \overset{\varepsilon_{\mathrm{out}}}{\approx} \mathsf{FFTConv}^{\mathrm{raw}}(\boldsymbol{a}, \boldsymbol{b}), \tag{132}$$

where the rounded integer vector is embedded into the complex representation.

Proof. We first move the selected raw output into a pairing, then use the network estimates to compare it with $\mathsf{Conv}_k(\boldsymbol{a}, \boldsymbol{b})$. The supplied raw circuit already contains $\mathsf{FFT}(\boldsymbol{a})$, $\mathsf{FFT}(\boldsymbol{b})$, their pointwise product, and its inverse transform. We reuse these outputs throughout. Recall $W = O(L^4)$ and $L = \log N$ from Appendix A.5.4.

1. *Prepare the bounds and the selected pairing.* We will prove

$$\mathsf{Sel}(\mathsf{FFTConv}^{\mathrm{raw}}(\boldsymbol{a}, \boldsymbol{b}), k) = \frac{1}{N} \langle \overline{\mathsf{FFT}}(\boldsymbol{e}_k), \mathsf{FFT}(\boldsymbol{a}) \odot \mathsf{FFT}(\boldsymbol{b}) \rangle. \tag{133}$$

Construct the decoder $\boldsymbol{e}_k$, $\overline{\mathsf{FFT}}(\boldsymbol{e}_k)$, and the two auxiliary pairings needed to apply Lemmas D.11 and F.3:

$$\begin{aligned}\mathsf{Sel}(\mathsf{FFTConv}^{\mathrm{raw}}(\boldsymbol{a},\boldsymbol{b}),k) &= \langle \boldsymbol{e}_k, \mathsf{IFFT}(\mathsf{FFT}(\boldsymbol{a}) \odot \mathsf{FFT}(\boldsymbol{b}))\rangle && \text{Lemma D.11}\\ &= \frac{1}{N}\langle \overline{\mathsf{FFT}}(\boldsymbol{e}_k), \mathsf{FFT}(\boldsymbol{a}) \odot \mathsf{FFT}(\boldsymbol{b})\rangle && \text{Lemma F.3 (115b).}\end{aligned}$$

The pairing on the last line has scale $3Lt$, and exact division by N gives scale $f_* := 3Lt + L$, the raw-output scale. The real-to-complex bound conversion in Lemma C.1 (81) supplies the input bounds for the FFTs. Applying Lemma F.4 (118a) gives $\|\mathsf{FFT}(\boldsymbol{a})\|_\infty, \|\mathsf{FFT}(\boldsymbol{b})\|_\infty \le N^2M$. Construct their missing bound checks here. The new decoder, network, pairings, and checks have size $N \log^{O(1)} N$; the cited derivations add $N \log^{O(1)} N$.

2. *Compare the pairing with the integer coefficient.* Construct $\boldsymbol{\chi}_k$, the routed array $\mathsf{CShift}_k\mathsf{Reflect}\boldsymbol{b}$, its conjugated FFT, and the intermediate products and pairings in the following chain. The routing already occurs in $\mathsf{Conv}_k(\boldsymbol{a},\boldsymbol{b})$ and is reused. We prove

$$\begin{aligned}\mathsf{Sel}(\mathsf{FFTConv}^{\mathrm{raw}}(\boldsymbol{a},\boldsymbol{b}),k) &= \frac{1}{N}\langle \overline{\mathsf{FFT}}(\boldsymbol{e}_k), \mathsf{FFT}(\boldsymbol{a}) \odot \mathsf{FFT}(\boldsymbol{b})\rangle && (133)\\ &\overset{\varepsilon_{\mathrm{tw}}N^7M^2}{\approx} \frac{1}{N}\langle \boldsymbol{\chi}_k, \mathsf{FFT}(\boldsymbol{a}) \odot \mathsf{FFT}(\boldsymbol{b})\rangle && \text{Lemma F.4 (118d)}\\ &= \frac{1}{N}\langle \mathsf{FFT}(\boldsymbol{a}), \boldsymbol{\chi}_k \odot \mathsf{FFT}(\boldsymbol{b})\rangle && \text{Lemma C.1 (63)}\\ &\overset{\varepsilon_{\mathrm{tw}}N^{10}M^2}{\approx} \frac{1}{N}\langle \mathsf{FFT}(\boldsymbol{a}), \overline{\mathsf{FFT}}(\mathsf{CShift}_k\mathsf{Reflect}\boldsymbol{b})\rangle && \text{Lemma F.4 (118e)}\\ &\overset{\varepsilon_{\mathrm{tw}}N^5M^2}{\approx} \langle \boldsymbol{a}, \mathsf{CShift}_k\mathsf{Reflect}\boldsymbol{b}\rangle && \text{Lemma F.4 (118c)}\\ &= \mathsf{Conv}_k(\boldsymbol{a},\boldsymbol{b}) && \text{definition of } \mathsf{Conv}_k.\end{aligned}$$

Here is how the displayed errors are derived in EF. For the first replacement, the error in each product leaf is at most $(\varepsilon_{\mathrm{tw}}N^3)(N^2M)(N^2M)$. Summing N such errors and dividing by N gives $\varepsilon_{\mathrm{tw}}N^7M^2$. For the second, each leaf error is at most $(N^2M)(\varepsilon_{\mathrm{tw}}N^8M)$, giving $\varepsilon_{\mathrm{tw}}N^{10}M^2$ after the same sum and division. These leaf estimates use subtraction distributivity and the complex-norm product bound in Lemma C.1 (65), (67), and (78c). Its triangle inequality (77) combines the leaf errors along each existing balanced tree, once per adder. For the last replacement, Lemma D.4 (101) gives the bound M on $\mathsf{CShift}_k\mathsf{Reflect}\boldsymbol{b}$; divide the pairing error $\varepsilon_{\mathrm{tw}}N^6M^2$ by N. The exact middle transition applies commutativity and associativity at each leaf, then Lemma B.4 through the sum and division circuits.

Multiplication by the powers of two N^j and exact division by N use wiring. Lemma C.1 (60) transfers the product and order estimates to these implementations, and (61a) cancels N. The intermediate pairings have scales $3Lt$, $(2L+1)t$, and $2Lt$ before division; all their divided outputs and error words have scale at most f_*. Use (88) to compare them at f_*. Their full-result widths, including the L extra bits for summation, are $O(W)$.

The new circuits consist of a constant number of length-N networks, product arrays, and pairings, together with their checks: size $N \log^{O(1)} N$. The three network estimates, the routing bound, and $O(N)$ short-word and sum-tree proof blocks add $N \log^{O(1)} N$. Finally, Lemma C.1 (79) combines the three errors. Evaluate

$$\varepsilon_{\mathrm{tw}}(N^7 + N^{10} + N^5)M^2 \le \varepsilon_{\mathrm{tw}}N^{12}M^2 \le \varepsilon_{\mathrm{out}}$$

and use (68) to obtain (131a). These last parameter checks and small-step inferences cost $W^{O(1)}$. The total cost through the selected raw estimate is $N \log^{O(1)} N$.

3. *Use the actual rounding gates.* We will identify the selected rounded output with $\mathsf{Conv}_k(\boldsymbol{a}, \boldsymbol{b})$. Construct the short circuit $\mathsf{Round}(\mathrm{Re}\,\mathsf{Sel}(\mathsf{FFTConv}^{\mathrm{raw}}(\boldsymbol{a}, \boldsymbol{b}), k))$ and its equality check against $\mathsf{Sel}(\mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{a}, \boldsymbol{b}), k)$ Under each address case $k = j$, Lemma B.6 opens the two selectors. Both sides then use the same coordinate's rounding circuit; Lemma B.4 identifies their outputs. Lemma B.5 combines the N cases. The missing selector and decoder gates and these proofs cost $N \log^{O(1)} N$.

   To supply the premise of the rounding assertion, write $z := \mathsf{Sel}(\mathsf{FFTConv}^{\mathrm{raw}}(\boldsymbol{a}, \boldsymbol{b}), k)$ and $c := \mathsf{Conv}_k(\boldsymbol{a}, \boldsymbol{b})$ in this paragraph. These names reuse existing output wires and introduce no gates. Reuse the difference and norm circuits in (131a), and construct any missing scalar arithmetic and comparison gates in the derivation

$$\begin{aligned} |\mathrm{Re}\, z - c| &\le |\mathrm{Re}\, z - c| + |\mathrm{Im}\, z| && \text{Lemma C.1 (76), (71), (62)} \\ &= |z - c|_1 && \text{component subtraction and the norm definition} \\ &\le \varepsilon_{\mathrm{out}} && \text{(131a)} \\ &< \tfrac{1}{4} < \tfrac{1}{2} && \text{closed parameter checks.} \end{aligned}$$

   For the first inequality, instantiate (76) with $x = \mathrm{Im}\, z$ to obtain $0 \le |\mathrm{Im}\, z|$. Adding $|\mathrm{Re}\, z - c|$ to both sides by (71) gives $0 + |\mathrm{Re}\, z - c| \le |\mathrm{Im}\, z| + |\mathrm{Re}\, z - c|$. The addition identities (62) remove zero and commute the right-hand summands; Lemma B.4 transfers these equalities through the comparison circuit. For the equality, c is embedded with imaginary component zero. The addition identities identify the imaginary component of $z - c$ with $\mathrm{Im}\, z$, and Lemma B.4 propagates this identity through the absolute-value and norm circuits. All differences and comparisons use the common scale f_*. The missing arithmetic, component-identification checks, parameter checks, and these short-word derivations together cost $W^{O(1)}$.

   Order transitivity and its mixed strict version in Lemma C.1 (68), (70) now derive $|\mathrm{Re}\, z - c| < 1/2$. Apply (96) to obtain $\mathsf{Round}(\mathrm{Re}\, z) = c$, and use Lemma B.4 with the selector–rounding equality proved above to obtain (131b). These final checks and proof blocks add $W^{O(1)}$. Both selected conclusions therefore have total additional cost $N \log^{O(1)} N$.

4. *Obtain the array error bound.* At an arbitrary L-bit index k (Section 3.2), the two selected conclusions give

$$\mathsf{Sel}(\mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{a}, \boldsymbol{b}), k) \overset{\varepsilon_{\mathrm{out}}}{\approx} \mathsf{Sel}(\mathsf{FFTConv}^{\mathrm{raw}}(\boldsymbol{a}, \boldsymbol{b}), k).$$

   This conclusion compares two short words. Its indexed derivation costs $N \log^{O(1)} N$, including construction of $\mathsf{Conv}_k(\boldsymbol{a}, \boldsymbol{b})$ and every other index-dependent auxiliary circuit used above. Apply index-generalization (Lemma B.8) to this small predicate, then Lemma B.2 to the returned coordinate assertions. The generalization, including its single application of the indexed derivation, costs $N \log^{O(1)} N$; folding adds $O(N)$. Thus (132) has the stated total cost. Only one selected coefficient sum is constructed.

□

Lemma G.2 (Coefficient bounds and zero entries)**.** *Let $\boldsymbol{a}, \boldsymbol{b}$ satisfy the input bounds of Lemma G.1, with those assertions available in Γ. All working words have width $O(W)$. For nonnegative bound words A, B, we have*

$$\begin{aligned} \Gamma :\ & 0 \le a_j \le A, \quad 0 \le b_j \le B\ (j < N) \ \rhd_{\mathsf{EF}}^{N \log^{O(1)} N} \\ & \big[0 \le \mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{a}, \boldsymbol{b})_j \le NAB\big]_{j<N}. \end{aligned} \tag{134}$$

For signed inputs, the corresponding bound is

$$\Gamma:\ \|\boldsymbol{a}\|_\infty \le A, \quad \|\boldsymbol{b}\|_\infty \le B \ \rhd_{\mathsf{EF}}^{N\log^{O(1)} N} \\ \|\mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{a},\boldsymbol{b})\|_\infty \le NAB. \tag{135}$$

If $r, s \ge 1$ are fixed, $r + s - 1 < N$, and the inputs vanish at indices at least r, s, respectively, then

$$\Gamma:\ [a_j = 0]_{r\le j<N}, \quad [b_j = 0]_{s\le j<N} \ \rhd_{\mathsf{EF}}^{N\log^{O(1)} N} \\ [\mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{a},\boldsymbol{b})_j = 0]_{r+s-1\le j<N}. \tag{136}$$

Proof. 1. *Bound the selected sum.* At an arbitrary L-bit index k, reuse the routing and sum in $\mathsf{Conv}_k(\boldsymbol{a},\boldsymbol{b})$. Lemma D.4 carries the input bounds on $\boldsymbol{b}$ to the routed array $\mathsf{CShift}_k\mathsf{Reflect}\boldsymbol{b}$: for nonnegative inputs, apply (100) with the check $0 \le x \le B$, and for signed inputs, apply (101). Lemma C.1 (74), (73), and (78c) bound the products and then their sum by NAB; nonnegative inputs also give a lower bound zero. At each adder, (73) combines its two child bounds. Use (131b) to transfer these checks to the selected rounded output. The routed bound checks, N product checks, and partial-sum checks are new, but the coefficient circuit is shared. They and their proofs cost $N\log^{O(1)} N$.

2. *Prove that high-index entries are zero.* Apply Lemma D.4 (100) to $\boldsymbol{b}$ with the check $P(x,p) := [p \ge s] \Rightarrow [x = 0]$. Its premises hold for $j \ge s$ by $b_j = 0$, and for $j < s$ because the closed check $[j \ge s]$ evaluates to false. It gives $[(k - j \bmod N) \ge s] \Rightarrow [(\mathsf{CShift}_k\mathsf{Reflect}\boldsymbol{b})_j = 0]$ for every $j < N$. Under $k \ge r + s - 1$, each summand $a_j(\mathsf{CShift}_k\mathsf{Reflect}\boldsymbol{b})_j$ is zero. If $j \ge r$, this follows from $a_j = 0$. If $j < r$, short integer arithmetic gives $k \ge j$ and $k - j \ge s$; there is no modular borrow, so the routed entry is zero. Lemma C.1 (63e), (63f), and the addition identities then give a zero sum. The application of Lemma D.4 and the N short-word derivations of this step cost $N\log^{O(1)} N$. An input array whose entries are all zero is handled directly by multiplication by zero.

3. *Pass to all coordinates.* The selected range predicate and the conditional zero check $[k \ge r + s - 1] \Rightarrow [\mathsf{Sel}(\mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{a},\boldsymbol{b}), k) = 0]$ each have size $W^{O(1)}$ once their selected words are supplied. Apply index-generalization (Lemma B.8), charging the selected coefficient circuit and routing proofs to its indexed derivation, and use Lemmas B.2 and B.3 as needed for the displayed lists or norm check. The total cost is $N\log^{O(1)} N$ for each coefficient-bound or zero-entry assertion.

□

H Vector Distributivity, Shift Compatibility, and Associativity

The raw convolution is exactly bilinear. Its rounded outputs satisfy the same identities because the accumulated errors are smaller than the gap between distinct integers. Associativity needs a further step: compare the FFT of each rounded inner product with the pointwise product that produced it.

Recall that $M = N^8$ is a bound on the coordinates of the input array.

Lemma H.1 (Exact identities for rounded coefficient circuits)**.** *Let $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \in \mathcal{D}_0^N$ have $O(W)$-bit components. Let $\mathcal{R}$ consist of $\|\boldsymbol{h}\|_\infty \le M$, for every array $\boldsymbol{h} \in \{\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}\}$ used as an input to a rounded convolution*

$\mathsf{FFTConv}^{\mathrm{round}}$ *in the following conclusions, including inner outputs used by outer convolutions. Then*

$$\Gamma : \mathcal{R} \ \triangleright_{\mathsf{EF}}^{N \log^{O(1)} N}$$

$$\mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{a}+\boldsymbol{b},\boldsymbol{c}) = \mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{a},\boldsymbol{c}) + \mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{b},\boldsymbol{c}), \tag{137a}$$

$$\mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{a},\boldsymbol{b}) = \mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{b},\boldsymbol{a}), \tag{137b}$$

$$\mathsf{FFTConv}^{\mathrm{round}}(2\boldsymbol{a},\boldsymbol{b}) = 2\mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{a},\boldsymbol{b}), \tag{137c}$$

$$\mathsf{FFTConv}^{\mathrm{round}}(-\boldsymbol{a},\boldsymbol{b}) = -\mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{a},\boldsymbol{b}), \tag{137d}$$

$$\mathsf{FFTConv}^{\mathrm{round}}(\mathsf{CShift}_1\boldsymbol{a},\boldsymbol{b}) = \mathsf{CShift}_1\mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{a},\boldsymbol{b}), \tag{137e}$$

$$\mathsf{FFTConv}^{\mathrm{round}}(\mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{a},\boldsymbol{b}),\boldsymbol{c}) = \mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{a},\mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{b},\boldsymbol{c})). \tag{137f}$$

For any one conclusion, only the bounds on inputs to the convolution instances in that conclusion are required. The additions and subtractions act on scale-zero coefficient words, and doubling uses left-shift wiring. The binary-string adder ADD *is a different circuit.*

Proof. All raw convolutions occurring inside the supplied rounded circuits are reused. The auxiliary circuits below contain a constant number of further network copies and $O(N)$ short-word operations per step. Their full-result widths are $O(W)$.

Distributivity, symmetry, doubling, and negation.

1. *Prove the exact raw identities.* We first prove

$$\mathsf{FFTConv}^{\mathrm{raw}}(\boldsymbol{a}+\boldsymbol{b},\boldsymbol{c}) = \mathsf{FFTConv}^{\mathrm{raw}}(\boldsymbol{a},\boldsymbol{c}) + \mathsf{FFTConv}^{\mathrm{raw}}(\boldsymbol{b},\boldsymbol{c}). \tag{138}$$

   Construct the intermediate array sums, pointwise products, and inverse transforms in this chain, reusing every operand already in the supplied raw convolutions:

$$\begin{aligned}
\mathsf{FFTConv}^{\mathrm{raw}}(\boldsymbol{a}+\boldsymbol{b},\boldsymbol{c}) &= \mathsf{IFFT}\big((\mathsf{FFT}(\boldsymbol{a}) + \mathsf{FFT}(\boldsymbol{b})) \odot \mathsf{FFT}(\boldsymbol{c})\big) && \text{Lemma F.2 (110)}\\
&= \mathsf{IFFT}\big(\mathsf{FFT}(\boldsymbol{a}) \odot \mathsf{FFT}(\boldsymbol{c}) + \mathsf{FFT}(\boldsymbol{b}) \odot \mathsf{FFT}(\boldsymbol{c})\big) && \text{Lemma C.1 (66)}\\
&= \mathsf{FFTConv}^{\mathrm{raw}}(\boldsymbol{a},\boldsymbol{c}) + \mathsf{FFTConv}^{\mathrm{raw}}(\boldsymbol{b},\boldsymbol{c}) && \text{Lemma F.2 (111).}
\end{aligned}$$

   The additions have scales Lt, $2Lt$, and $3Lt+L$, respectively. The middle transition uses one distributivity derivation per coordinate. Lemma B.4 carries each array equality through its surrounding network, once; differences in operand widths are handled by the local padding proofs used in Lemma F.2.

   For symmetry, apply Lemma C.1 (63a) at each pointwise product and propagate the result through IFFT. For doubling and negation, apply scalar linearity (110), (111) and multiplication associativity (63b) at each coordinate. For the scalar 2, (60a) identifies multiplication with the specified shift; for -1, (67) applied to $0-1$, with (63f) and (63d), identifies multiplication with negation. The new intermediate arrays, a constant number of networks, and their checks cost $N \log^{O(1)} N$. The constant number of linearity and propagation proofs, together with $O(N)$ short-word blocks, add $N \log^{O(1)} N$ for these four raw identities.

2. *Transfer the identities to the rounded outputs.* The three instances of Lemma G.1 (132) give

$$\begin{aligned}
\mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{a}+\boldsymbol{b},\boldsymbol{c}) &\overset{\varepsilon_{\mathrm{out}}}{\approx} \mathsf{FFTConv}^{\mathrm{raw}}(\boldsymbol{a}+\boldsymbol{b},\boldsymbol{c}) && (132)\\
&= \mathsf{FFTConv}^{\mathrm{raw}}(\boldsymbol{a},\boldsymbol{c}) + \mathsf{FFTConv}^{\mathrm{raw}}(\boldsymbol{b},\boldsymbol{c}) && (138)\\
&\overset{2\varepsilon_{\mathrm{out}}}{\approx} \mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{a},\boldsymbol{c}) + \mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{b},\boldsymbol{c}) && (132); (77).
\end{aligned}$$

Construct the missing difference, norm, and comparison circuits in these bounds. Lemma C.1 (80) gives total error $3\varepsilon_{\mathrm{out}} < 1$. Unfold this vector bound by Lemma B.3, apply the integer-gap assertion (97) at each coordinate, and fold the equalities by Lemma B.2. This proves (137a).

The same comparison for doubling has errors $\varepsilon_{\mathrm{out}}$ and $2\varepsilon_{\mathrm{out}}$; for negation they are $\varepsilon_{\mathrm{out}}$ and $\varepsilon_{\mathrm{out}}$. The scalar error transfers use (78c) and (78b), with (60a) for doubling. Symmetry follows by propagating the exact raw equality through the rounding gates. All comparisons may be made at scale $3Lt + L$ using (89); the integer outputs remain at scale zero. There are a constant number of coefficient-lemma instances and $O(N)$ new short checks and integer-gap proofs. Their additional cost is $N \log^{O(1)} N$. Together with the previous step, this is the total cost of (137a)–(137d).

Compatibility with a cyclic shift. We prove (137e) using one selected coefficient on each side.

1. *Reindex the selected coefficient.* At an arbitrary L-bit index k (Section 3.2), construct the two sums $\mathsf{Conv}_k(\mathsf{CShift}_1 \boldsymbol{a}, \boldsymbol{b})$ and $\mathsf{Conv}_{k-1}(\boldsymbol{a}, \boldsymbol{b})$. All address operations are modulo N. Using the fixed shift by -1 on both factors of the first pairing gives

$$\begin{aligned} \mathsf{Conv}_k(\mathsf{CShift}_1 \boldsymbol{a}, \boldsymbol{b}) &= \langle \boldsymbol{a}, \mathsf{CShift}_{-1}\mathsf{CShift}_k\mathsf{Reflect}\boldsymbol{b} \rangle && \text{Lemma D.10 (102a)} \\ &= \langle \boldsymbol{a}, \mathsf{CShift}_{k-1}\mathsf{Reflect}\boldsymbol{b} \rangle && \text{Lemma D.2 (98)} \\ &= \mathsf{Conv}_{k-1}(\boldsymbol{a}, \boldsymbol{b}) && \text{definition of Conv.} \end{aligned}$$

   For the routing equality in the second line, use an arbitrary coordinate index i. The two selected addresses are both $i-(k-1)$ modulo N. Lemma C.1 (62), (92) prove this address identity; Lemma B.4 transfers it through the selector. Apply index-generalization (Lemma B.8) with k retained in the base context, then fold the coordinate equalities. The missing routing and pairing circuits have size $N \log^{O(1)} N$. The fixed-pairing permutation proof, routing proof, and equality substitutions add $N \log^{O(1)} N$.

2. *Remove the coefficient sums from the conclusion.* Apply Lemma G.1 (131b) to both sums, and use Lemma D.2 (98) on the rounded array. The result is

$$\mathsf{Sel}(\mathsf{FFTConv}^{\mathrm{round}}(\mathsf{CShift}_1 \boldsymbol{a}, \boldsymbol{b}), k) = \mathsf{Sel}(\mathsf{CShift}_1 \mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{a}, \boldsymbol{b}), k).$$

   This indexed derivation works at the arbitrary index k, and all auxiliary sums, routings, and proofs from the preceding step are charged to it. Its cost is $N \log^{O(1)} N$. The conclusion is a short-word equality, so index-generalization (Lemma B.8) followed by folding (Lemma B.2) gives (137e) with the same total cost.

Associativity. Name outputs already in the supplied nested convolutions by

$$\begin{aligned} \boldsymbol{A} &:= \mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{a}, \boldsymbol{b}), & \boldsymbol{B} &:= \mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{b}, \boldsymbol{c}), \\ \boldsymbol{P} &:= \mathsf{FFT}(\boldsymbol{a}) \odot \mathsf{FFT}(\boldsymbol{b}), & \boldsymbol{Q} &:= \mathsf{FFT}(\boldsymbol{b}) \odot \mathsf{FFT}(\boldsymbol{c}). \end{aligned}$$

Naming these outputs introduces no gates. The two outer raw circuits also already contain $\mathsf{FFT}(\boldsymbol{A})$ and $\mathsf{FFT}(\boldsymbol{B})$.

1. *Compare the transformed inner outputs with their pointwise products.* We will prove, with $\beta := 2N^2\varepsilon_{\mathrm{out}}$ and $f := 4Lt + L$,

$$\mathsf{FFT}(\boldsymbol{A}) \overset{\beta}{\approx}_f \boldsymbol{P}, \qquad \mathsf{FFT}(\boldsymbol{B}) \overset{\beta}{\approx}_f \boldsymbol{Q}. \tag{139}$$

Construct the missing $\mathsf{FFT}(\mathsf{IFFT}(\boldsymbol{P}))$ and the network on the difference $\boldsymbol{A} - \mathsf{IFFT}(\boldsymbol{P})$; $\mathsf{IFFT}(\boldsymbol{P})$ is already the raw inner output. Construct the corresponding circuits for $\boldsymbol{Q}$ once as well. For $\boldsymbol{P}$, the derivation is

$$\begin{aligned} \boldsymbol{A} &\overset{\varepsilon_{\text{out}}}{\approx} \mathsf{IFFT}(\boldsymbol{P}) && \text{Lemma G.1 (132)} \\ \mathsf{FFT}(\boldsymbol{A}) &\overset{N^2\varepsilon_{\text{out}}}{\approx}_f \mathsf{FFT}(\mathsf{IFFT}(\boldsymbol{P})) && \text{Lemma F.2 (110); Lemma F.4 (118a)} \\ &\overset{\varepsilon_{\text{tw}}N^{28}}{\approx}_f \boldsymbol{P} && \text{Lemma F.4 (118b).} \end{aligned}$$

For the last line, the FFT bounds and Lemma C.1 (78c) give $\|\boldsymbol{P}\|_\infty \le N^4M^2 = N^{20}$. The same estimates apply to $\boldsymbol{Q}$. The difference-network input is bounded by ε_{out}; both this bound and N^{20} satisfy the N^{32} restriction of Lemma F.4. Evaluate these parameter checks and $N^2\varepsilon_{\text{out}} + \varepsilon_{\text{tw}}N^{28} \le \beta$. Then Lemma C.1 (80) and (68) prove (139).

For the second line, promote the difference operands to scale $3Lt + L$. The local promotion identities (84), (86), and (85), applied once per butterfly, identify the compared outputs with the promotions of the existing $\mathsf{FFT}(\boldsymbol{A})$ and $\mathsf{FFT}(\mathsf{IFFT}(\boldsymbol{P}))$. All new scales are at most f, and widths remain $O(W)$. A constant number of new networks, the product-bound and error checks, and the cited coefficient, linearity, and network derivations cost $N \log^{O(1)} N$.

2. *Compare the two outer raw convolutions.* We will prove, at comparison scale $f' := 6Lt + 2L$,

$$\mathsf{FFTConv}^{\text{raw}}(\boldsymbol{A}, \boldsymbol{c}) \overset{2\beta N^4 M}{\approx}_{f'} \mathsf{FFTConv}^{\text{raw}}(\boldsymbol{a}, \boldsymbol{B}). \tag{140}$$

Construct the two auxiliary triple-product arrays and the differences in

$$\begin{aligned} \mathsf{FFT}(\boldsymbol{A}) \odot \mathsf{FFT}(\boldsymbol{c}) &\overset{\beta N^2 M}{\approx} \boldsymbol{P} \odot \mathsf{FFT}(\boldsymbol{c}) && \text{(139); Lemma C.1 (78c)} \\ &= \mathsf{FFT}(\boldsymbol{a}) \odot \boldsymbol{Q} && \text{Lemma C.1 (63b)} \\ &\overset{\beta N^2 M}{\approx} \mathsf{FFT}(\boldsymbol{a}) \odot \mathsf{FFT}(\boldsymbol{B}) && \text{(139); Lemma C.1 (78c).} \end{aligned}$$

The factor bounds are $\|\mathsf{FFT}(\boldsymbol{a})\|_\infty, \|\mathsf{FFT}(\boldsymbol{c})\|_\infty \le N^2M$. Subtraction distributivity (65), (67) identifies the leaf differences to which the product bound applies. Use (80) to obtain error $2\beta N^2 M$. Construct one inverse network on the difference of the two endpoint product arrays. Its input bound is at most N^{32}, by a closed parameter check. Lemma F.2 (111) and Lemma F.4 (118a) multiply that error by at most N^2, giving (140). Promote the product comparisons to $f + Lt$; the inverse output then has scale f'. The same local promotion proofs as in the preceding step identify the endpoint outputs with the existing raw convolutions. The new product arrays, difference network, checks, and all cited derivations cost $N \log^{O(1)} N$; their widths are $O(W)$.

3. *Use the integer gap on the outer outputs.* Reuse the outer rounding circuits and apply Lemma G.1 (132) to each. The resulting chain is

$$\begin{aligned} \mathsf{FFTConv}^{\text{round}}(\boldsymbol{A}, \boldsymbol{c}) &\overset{\varepsilon_{\text{out}}}{\approx}_{f'} \mathsf{FFTConv}^{\text{raw}}(\boldsymbol{A}, \boldsymbol{c}) && \text{(132); Lemma C.1 (89)} \\ &\overset{2\beta N^4 M}{\approx}_{f'} \mathsf{FFTConv}^{\text{raw}}(\boldsymbol{a}, \boldsymbol{B}) && \text{(140)} \\ &\overset{\varepsilon_{\text{out}}}{\approx}_{f'} \mathsf{FFTConv}^{\text{round}}(\boldsymbol{a}, \boldsymbol{B}) && \text{(132); Lemma C.1 (89).} \end{aligned}$$

The triangle and order assertions (80), (68), together with closed parameter evaluation, give

$$2\varepsilon_{\text{out}} + 2\beta N^4 M = 2\varepsilon_{\text{out}} + 4N^6 M \varepsilon_{\text{out}} \le 6N^{14}\varepsilon_{\text{out}} < 1.$$

Unfold the bound, apply Lemma C.1 (97) to the integer outputs, and fold their equalities. The two coefficient derivations cost $N \log^{O(1)} N$; the missing comparison circuits and coordinate proofs add $NW^{O(1)}$. Thus the total additional cost of associativity is $N \log^{O(1)} N$. The proof uses four rounded convolution instances without expanding a double convolution sum.

Total cost. Each of the three groups uses a constant number of network or coefficient lemmas and $N \log^{O(1)} N$ short-word work, including missing circuitry, premise checks, and the indicated applications of index-generalization. Shared FFT outputs and previously proved assertions are reused. Since there are only a constant number of groups, all six conclusions together have additional cost $N \log^{O(1)} N$. □

I Carry Resolution

The vector identities concern integer coefficients, while FFTMul returns binary digits. We connect them by a local certificate: replacing coefficients $\boldsymbol{a}$ by their carry-resolved digits changes $\boldsymbol{a}$ by a vector of the form $2\boldsymbol{u} - \mathsf{Shift}\,\boldsymbol{u}$. Here Shift is the ordinary left shift, with a zero at index zero and the last entry discarded. It equals CShift_1 when the last entry is zero.

All coefficient and carry words below have scale zero and width $O(W)$. Their additions and subtractions retain full results; multiplication by two uses left-shift wiring. The quotient $\lfloor x/2 \rfloor$ is the integer quotient circuit of Appendix A.1.

Lemma I.1 (Carry identities from local gates)**.** *For length-N coefficient arrays $\boldsymbol{a}, \boldsymbol{u}$ with $O(W)$-bit working words,*

$$\Gamma:\ u_{N-1} = 0 \ \rhd_{\mathsf{EF}}^{N \log^{O(1)} N}\ \mathsf{Carry}(\boldsymbol{a} + 2\boldsymbol{u} - \mathsf{Shift}\,\boldsymbol{u}) = \mathsf{Carry}(\boldsymbol{a}). \tag{141}$$

For coefficient arrays $\boldsymbol{a}, \boldsymbol{a}'$, let $\mathcal{R}$ assert that $\boldsymbol{a}, \boldsymbol{a}'$ are nonnegative, the terminal carries for $\boldsymbol{a}, \boldsymbol{a}', \boldsymbol{a} + \boldsymbol{a}'$ vanish, and the final carry of the output adder vanishes. Then

$$\Gamma : \mathcal{R}\ \rhd_{\mathsf{EF}}^{N \log^{O(1)} N}\ \mathsf{Carry}(\boldsymbol{a} + \boldsymbol{a}') = \mathsf{ADD}(\mathsf{Carry}(\boldsymbol{a}), \mathsf{Carry}(\boldsymbol{a}')). \tag{142}$$

Finally, let $\boldsymbol{b} := \mathsf{Carry}(\boldsymbol{a})$, let κ_j be its carry words, and put $u_j := \kappa_{j+1}$. Then

$$\Gamma\ \rhd_{\mathsf{EF}}^{N \log^{O(1)} N}\ \boldsymbol{a} - \boldsymbol{b} = 2\boldsymbol{u} - \mathsf{Shift}\,\boldsymbol{u}, \tag{143}$$

$$\Gamma:\ \kappa_N = 0\ \rhd_{\mathsf{EF}}^{W^{O(1)}}\ u_{N-1} = 0. \tag{144}$$

Proof. We compare the actual carry gates, recording one invariant at each column. The supplied circuits already contain their coefficient sums, output bits, and carries. Only the arithmetic and checks expressing the invariants below are auxiliary.

1. *Record the local quotient-and-remainder identity.* For every carry column, Lemma C.1 (90a) gives

$$h_j = a_j + \kappa_j = b_j + 2\kappa_{j+1}, \qquad b_j \in \{0, 1\}. \tag{145}$$

We will also use the following consequence: if $h = b + 2q$ and b is a bit, then $h \bmod 2 = b$ and $\lfloor h/2 \rfloor = q$. To derive it, inspect the adder for $b + 2q$: its low column returns b without a carry, and its higher columns return the bits of q, with sign extension. Lemma B.4 transfers the equality $h = b + 2q$ through the low-bit and quotient wiring. This is a $W^{O(1)}$-cost local derivation, including missing checks, on arbitrary signed q; it does not assume that h is nonnegative. For algebra involving the shifts $2q$, use Lemma C.1 (60a) before its multiplication identities. All such transfers are included in the per-column costs below.

2. *Compare carry chains after a certificate perturbation.* For (141), let κ_j, b_j be the carries and bits for $\boldsymbol{a}$, and let $\widetilde{\kappa}_j, \widetilde{b}_j$ be those for $\boldsymbol{a} + 2\boldsymbol{u} - \mathsf{Shift}\,\boldsymbol{u}$. Set $u_{-1} := 0$ for the first-column wiring. Construct the short invariant checks

$$\widetilde{\kappa}_j = \kappa_j + u_{j-1}.$$

At $j = 0$ both carries and u_{-1} are zero. Assuming the invariant at column j, derive

$$\begin{aligned}(a_j + 2u_j - u_{j-1}) + \widetilde{\kappa}_j &= a_j + \kappa_j + 2u_j && \text{Lemma C.1 (62); the invariant}\\ &= b_j + 2(\kappa_{j+1} + u_j) && \text{(145); Lemma C.1 (64).}\end{aligned}$$

The local consequence in Step 1 gives $\widetilde{b}_j = b_j$ and $\widetilde{\kappa}_{j+1} = \kappa_{j+1} + u_j$. Thus the output bits agree at all columns; $u_{N-1} = 0$ also identifies the terminal carries. Fold the bit equalities by Lemma B.2. Each column introduces a fixed number of $O(W)$-bit sums and checks and uses $W^{O(1)}$ additional proof steps. The total new-circuit and proof cost is $NW^{O(1)} = N \log^{O(1)} N$.

3. *Compare coefficient addition with binary addition.* For (142), write $\boldsymbol{b}, \boldsymbol{b}'$ for the two carry outputs, $\kappa_j, \kappa'_j, \kappa''_j$ for the three word-carry chains, and γ_j, d_j for the carries and output bits of $\mathsf{ADD}(\boldsymbol{b}, \boldsymbol{b}')$. All these wires already occur in the conclusion circuits. Construct the invariant

$$\kappa''_j = \kappa_j + \kappa'_j + \gamma_j,$$

which holds at column zero. The word identities and the full-adder equation give

$$\begin{aligned}a_j + a'_j + \kappa''_j &= (b_j + b'_j + \gamma_j) + 2(\kappa_{j+1} + \kappa'_{j+1}) && \text{(145); Lemma C.1 (62)}\\ &= d_j + 2(\kappa_{j+1} + \kappa'_{j+1} + \gamma_{j+1}) && \text{the output-adder cell; (64).}\end{aligned}$$

Step 1 identifies the next output bit with d_j and proves the next invariant. After all N columns, the supplied zero final carry identifies the extra high adder bit with the zero padding on the other side. The missing invariant circuits and proofs cost $W^{O(1)}$ per column; folding adds $O(N)$. The total cost is $N \log^{O(1)} N$.

4. *Extract the certificate and its boundary.* The definition $u_j := \kappa_{j+1}$ only names existing carry wires. Rearrange (145) using Lemma C.1 (62) to obtain

$$a_j - b_j = 2\kappa_{j+1} - \kappa_j = 2u_j - (\mathsf{Shift}\,\boldsymbol{u})_j.$$

The second equality is wiring, including $\kappa_0 = 0$. The new coordinate arithmetic, equality checks, and their proofs cost $NW^{O(1)}$; folding proves (143) within that bound. Since u_{N-1} names κ_N, the boundary assertion (144) needs only a short equality check and its derivation, of cost $W^{O(1)}$.

□

Lemma I.2 (Carry resolution may be deferred across multiplication)**.** *Let $\boldsymbol{a}, \boldsymbol{z} \in \mathcal{D}_0^N$ and let $\boldsymbol{b} := \mathsf{Carry}(\boldsymbol{a})$. Let $\boldsymbol{u} := (\kappa_1, \ldots, \kappa_N)$ be the array of outgoing carries in this computation, so that $2\boldsymbol{u} - \mathsf{Shift}\,\boldsymbol{u}$ is its carry certificate in (143). All working words have width $O(W)$. Let $\mathcal{R}$ contain nonnegativity of $\boldsymbol{a}$, $\kappa_N = 0$, the infinity-norm bound $M = N^8$ on each of $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{z}, \boldsymbol{u}, 2\boldsymbol{u}, \mathsf{Shift}\,\boldsymbol{u}, \boldsymbol{a} - \boldsymbol{b}$, and the assertion $\mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{u}, \boldsymbol{z})_{N-1} = 0$. Then*

$$\Gamma : \mathcal{R} \ \triangleright_{\mathsf{EF}}^{N \log^{O(1)} N}$$

$$\mathsf{Carry}(\mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{a}, \boldsymbol{z})) = \mathsf{Carry}(\mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{b}, \boldsymbol{z})), \tag{146a}$$

$$\mathsf{Carry}(\mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{z}, \boldsymbol{a})) = \mathsf{Carry}(\mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{z}, \boldsymbol{b})). \tag{146b}$$

The second conclusion is the first one with the arguments of both convolutions exchanged, so that the resolved array occupies the second argument.

Proof. We first show that the two coefficient outputs differ by another carry certificate, then apply Lemma I.1.

1. *Transfer the certificate through rounded convolution.* Construct $\boldsymbol{V} := \mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{u}, \boldsymbol{z})$ once, together with the intermediate convolutions in the following chain. Reuse $\mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{a}, \boldsymbol{z})$ and $\mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{b}, \boldsymbol{z})$ from the supplied carry circuits. We prove

$$\begin{aligned}
&\mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{a}, \boldsymbol{z}) - \mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{b}, \boldsymbol{z}) \\
&= \mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{a} - \boldsymbol{b}, \boldsymbol{z}) && \text{Lemma H.1 (137a)} \\
&= \mathsf{FFTConv}^{\mathrm{round}}(2\boldsymbol{u} - \mathsf{Shift}\,\boldsymbol{u}, \boldsymbol{z}) && \text{Lemma I.1 (143)} \\
&= \mathsf{FFTConv}^{\mathrm{round}}(2\boldsymbol{u}, \boldsymbol{z}) - \mathsf{FFTConv}^{\mathrm{round}}(\mathsf{Shift}\,\boldsymbol{u}, \boldsymbol{z}) && \text{Lemma H.1 (137a)} \\
&= 2\boldsymbol{V} - \mathsf{Shift}\,\boldsymbol{V} && \text{Lemma H.1 (137c), (137e).}
\end{aligned} \tag{147}$$

   The subtraction forms in the first and third lines follow by applying (137a) to a difference and the subtracted summand, then cancelling that summand using Lemma C.1 (62). For the last line, (144) gives $u_{N-1} = 0$, and the last-coordinate premise gives $V_{N-1} = 0$. Consequently $\mathsf{CShift}_1\boldsymbol{u} = \mathsf{Shift}\,\boldsymbol{u}$ and $\mathsf{CShift}_1\boldsymbol{V} = \mathsf{Shift}\,\boldsymbol{V}$ by their wiring. Lemma B.4 transfers these equalities and the certificate through the displayed convolution and arithmetic circuits.

   The stated bounds supply the vector-law premises. The additional input $2\boldsymbol{u} - \mathsf{Shift}\,\boldsymbol{u}$ equals $\boldsymbol{a} - \boldsymbol{b}$ by the certificate, so its bound is derived from the supplied one; the other sums used in the subtraction argument equal $\boldsymbol{a}$ or $2\boldsymbol{u}$. Construct these bound checks and derive them by equality substitution. There are a constant number of new convolutions, coordinate arithmetic arrays, and checks, of total size $N \log^{O(1)} N$. The same bound covers all vector-law, carry-certificate, and substitution proofs.

2. *Remove the new certificate by carry resolution.* Rearrange (147) coordinatewise to identify $\mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{a}, \boldsymbol{z})$ with $\mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{b}, \boldsymbol{z}) + 2\boldsymbol{V} - \mathsf{Shift}\,\boldsymbol{V}$. Construct the carry circuit on this last expression if it is missing. Lemma B.4 and Lemma I.1 (141), with $V_{N-1} = 0$, identify its output with $\mathsf{Carry}(\mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{b}, \boldsymbol{z}))$. The new carry circuit, coordinate checks, and these proofs add $N \log^{O(1)} N$. This proves (146a). For (146b), apply symmetry (137b) before and after the derivation; the same symmetry transfers the last-coordinate assertion to the exchanged order. Thus either deferral, including all its auxiliary convolutions and premise checks, has total additional cost $N \log^{O(1)} N$.

□

Lemma I.3 (Bounds and termination of carry resolution)**.** *Let $\boldsymbol{a} = (a_0, \ldots, a_{N-1})$ be the input coefficient array of $\boldsymbol{b} := \mathsf{Carry}(\boldsymbol{a})$, and let κ_j be the carry entering position j. Thus $\kappa_0 = 0$, and $\kappa_{j+1} = \lfloor (a_j + \kappa_j)/2 \rfloor$ is the carry leaving that position. All working words have width $O(W)$. Fix a nonnegative integer A of width $O(W)$. Then*

$$\Gamma : \ [0 \le a_j \le A]_{j<N} \ \rhd_{\mathsf{EF}}^{NW^{O(1)}} \ [0 \le \kappa_j \le A]_{j \le N}. \tag{148}$$

Put $q := \lceil \log(A+1) \rceil$. For a fixed $r \ge 0$ with $r + q \le N$, we also have

$$\begin{aligned}
\Gamma : \ &[0 \le a_j \le A]_{j<N}, \quad [a_j = 0]_{r \le j < N} \ \rhd_{\mathsf{EF}}^{NW^{O(1)}} \\
&[\kappa_j = 0]_{r+q \le j \le N}, \quad [b_j = 0]_{r+q \le j < N}.
\end{aligned} \tag{149}$$

Proof. The first assertion says that a carry never exceeds the coefficient bound. The initial carry $\kappa_0 = 0$ satisfies the bound. At position j, assume the bound $0 \le \kappa_j \le A$ has already been derived. Together with $0 \le a_j \le A$, apply the rule for adding two inequalities of Lemma C.1 (73) to the lower bounds and then to the upper bounds, obtaining

$$0 \le a_j + \kappa_j \le 2A.$$

The quotient wiring gives a nonnegative outgoing carry. If it exceeded A, the column identity (145) would give $a_j + \kappa_j = b_j + 2\kappa_{j+1} > 2A$, a contradiction. This uses (74) and (60a) for multiplication by two. Unrolling this argument over the N positions proves (148).

Once the input coefficients become zero, each further position halves the remaining carry, discarding its low bit:

$$\kappa_{j+1} = \lfloor \kappa_j / 2 \rfloor \qquad (j \geq r).$$

The bound $0 \leq \kappa_r \leq A < 2^q$ implies that only its lowest q bits can be nonzero. The borrow-comparator gates certify this bit consequence in $W^{O(1)}$ size. After q successive right shifts all these bits have been discarded, so $\kappa_{r+q} = 0$. Every later carry and output bit is zero by the same column gates. For $A = 0$, we have $q = 0$ and the carry is already zero.

The input, output, and carry wires are reused. Only the bound and zero checks and the short arithmetic used to justify them are auxiliary. Each position costs $W^{O(1)}$ for these gates and derivations; the total additional cost is $NW^{O(1)}$ for either assertion. □

For a binary word v and a target length k at least its length, let $\mathsf{Pad}_k(v)$ append high zero bits to reach length k. This operation uses only wiring.

Lemma I.4 (Zero-padding the multiplier output)**.** *Fix positive input lengths r, s with $\ell := r + s + L + 2 < N$. Let $x \in \{0,1\}^r$ and $y \in \{0,1\}^s$, and let $\boldsymbol{a}, \boldsymbol{b}$ be their length-N bit embeddings from Appendix A.5. Then*

$$\Gamma \;\rhd^{N \log^{O(1)} N}_{\mathsf{EF}}\; \mathsf{Pad}_N\big(\mathsf{FFTMul}_{r,s}(x,y)\big) = \mathsf{Carry}\big(\mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{a},\boldsymbol{b})\big).$$

Proof. Reuse the rounded convolution and carry circuits inside $\mathsf{FFTMul}_{r,s}(x,y)$. Write $\boldsymbol{c}$ for the full length-N carry output; this name introduces no gates. We prove that every bit omitted from the multiplier's returned word is zero.

The input embedding gives $0 \leq a_j, b_j \leq 1$, with $a_j = 0$ for $j \geq r$ and $b_j = 0$ for $j \geq s$. Derive these coordinate checks from the input bits and hardwired zeros. Use the closed comparison $1 \leq M$ and order transitivity from Lemma C.1 (68), then Lemma B.2, to supply the input bounds required by Lemma G.2. Its assertions (134) and (136) give

$$0 \leq \mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{a},\boldsymbol{b})_j \leq N \quad (j < N), \qquad \mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{a},\boldsymbol{b})_j = 0 \quad (j \geq r + s - 1).$$

The missing checks and these derivations cost $N \log^{O(1)} N$; their arithmetic words have width $O(W)$.

Apply Lemma I.3 (149) with coefficient bound $A = N$, zero-entry cutoff $r + s - 1$, and $q = \lceil \log(N + 1) \rceil = L + 1$. The length condition holds because $r + s + L < \ell < N$. It proves $c_j = 0$ for $j \geq r + s + L$ and that the final carry is zero, at additional cost $NW^{O(1)}$, including missing checks. In particular, $c_j = 0$ for every $j \geq \ell$.

For $j < \ell$, the multiplier's returned bit is c_j by its definition. For $j \geq \ell$, the padding bit is zero and agrees with c_j by the preceding bound. Fold these bit equalities using Lemma B.2, at additional cost $O(N)$. Together, the new checks and all cited derivations have additional cost $N \log^{O(1)} N$. □

J Proof of Main Theorem

J.1 Bounds and Padding

When we prove Theorem A.2 in Appendices J.2 and J.3, we will invoke Lemma H.1 for the coefficient identities, Lemma I.1 to pass between coefficient arrays and binary words, and Lemma I.2 to move carry resolution past a convolution. Every one of these rules is conditional, and between them they ask for three kinds of premise:

- *coefficient bounds*: the magnitude bound $\|\cdot\|_\infty \le M = N^8$ on each array entering a rounded convolution, and nonnegativity of each array being carry-resolved;
- *zero entries*: $\kappa_N = 0$ for an array being carry-resolved, and a vanishing last coordinate for the auxiliary convolution of each deferral;
- *vanishing final carries*: the terminal carries of two coefficient arrays and of their sum, together with the final carry of the adder that combines their carry outputs.

In this subsection we give the EF derivations of these premises, for all the applications in Appendices J.2 and J.3.

Throughout, $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$ denote the length-N bit embeddings of the three n-bit inputs of Theorem A.2: their entries lie in $\{0, 1\}$ and vanish for $j \ge n$.

Coefficient bounds. The arrays $\boldsymbol{h}$ that enter a rounded convolution in the proof of Theorem A.2 include the inner outputs consumed by the outer convolutions, so there are several of them, and each needs its own bound before $\|\boldsymbol{h}\|_\infty \le M$ can be cited. We bound them in two stages: first the coefficient arrays, in the table below, and then the carry arrays that appear inside the proof of Lemma I.2.

Each entry of the table bounds the integer at a single array position, and every array listed is nonnegative. The last column records the rule used and the earlier rows supplying its premises.

Row	Array	Upper bound	Derivation and dependencies
1	Input bit arrays, including carry-resolved outputs	1	Each entry is a bit by the input encoding or carry-output wiring. No earlier row is needed.
2	The coefficient sum $\boldsymbol{y} + \boldsymbol{z}$	2	Row 1 bounds both summands by 1. Lemma C.1 (73) adds these bounds, giving $y_j + z_j \le 1 + 1 = 2$.
3	Rounded convolution of two bit arrays	N	Lemma G.2 (134) gives NAB; use row 1 for both inputs, with $A = B = 1$.
4	A further convolution with a bit array	N^2	Apply Lemma G.2 (134) with $A = N$ from row 3 and $B = 1$ from row 1.
5	Sum of two convolutions of bit arrays	$2N$	Row 3 bounds both summands by N. Lemma C.1 (73) adds these bounds, giving the bound $N + N = 2N$.

For row 1, the checks $0 \le x_j \le 1$ follow from the two possible bit values, and the high zero entries are hardwired; Lemma B.2 combines the N coordinate checks at cost $O(N)$. Nonnegativity comes with each bound: in rows 2 and 5 the same addition rule applied to the two lower bounds gives it, and (134) asserts it for a convolution. One case not listed in the table is the convolution of a bit array with $\boldsymbol{y} + \boldsymbol{z}$; rows 1 and 2 in (134) bound it by $2N \le N^2$.

To supply the remaining magnitude premises of Lemma I.2, we also bound the additional convolution inputs in its proof: $\boldsymbol{u}$, $2\boldsymbol{u}$, $\mathsf{Shift}\,\boldsymbol{u}$, $\boldsymbol{a} - \boldsymbol{b}$, and $2\boldsymbol{u} - \mathsf{Shift}\,\boldsymbol{u}$, where $\boldsymbol{b} := \mathsf{Carry}(\boldsymbol{a})$ and $u_j := \kappa_{j+1}$. In each application, $0 \le a_j \le A$ with $A \le N^2$. Lemma I.3 (148) gives $0 \le u_j \le A$, and $b_j \in \{0, 1\}$ by the carry-output wiring. The addition and norm rules of Lemma C.1 therefore bound the absolute values of all these entries by $4N^2$; shifted and negated inputs obey the same bound. The sums appearing in the subtraction argument equal $\boldsymbol{a}$ or $2\boldsymbol{u}$, as shown in the proof of Lemma I.2. Thus every input to a rounded convolution below has infinity norm at most $4N^2 \le M = N^8$.

For each convolution, first derive the bound on its inputs and compare it with M, then apply Lemma G.2 to bound its output. This order avoids using an output bound before its input premises are proved. The

comparisons with M use closed parameter checks and order transitivity (68); folding collects the coordinate bounds. The convolution and carry circuits are shared with their later uses. Their construction, when missing, and the bound checks and derivations cost $N \log^{O(1)} N$ for the constant number of instances.

Zero entries needed to defer carry resolution. The distributivity and multiplication-associativity arguments in the proof of Theorem A.2 use Lemma I.2 to move carry resolution past a convolution. Each application requires both $\kappa_N = 0$ for the array being carry-resolved and $\mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{u}, \boldsymbol{v})_{N-1} = 0$, where $\boldsymbol{u}$ is its array of outgoing carries and $\boldsymbol{v}$ is the other convolution input. Both premises say that an array vanishes from some index onward, and we obtain them by tracking that index through the construction. Two rules do the tracking: Lemma I.3 (149) moves the index from r to $r + q$ across carry resolution of coefficients bounded by A, where $q := \lceil \log(A + 1) \rceil$; and Lemma G.2 (136) sends indices r and s to $r + s - 1$ across a convolution, provided $r + s - 1 < N$. There are two kinds of application, and we treat each in turn.

For distributivity, the array being carry-resolved is $\boldsymbol{a} := \boldsymbol{y} + \boldsymbol{z}$. Its coefficients are at most 2 and vanish for $j \geq n$. The carry rule with $A = 2$, hence $q = 2$, gives $\kappa_j = 0$ for $j \geq n + 2$. Since $u_j = \kappa_{j+1}$, the two successive conclusions are

$$u_j = 0 \quad (j \geq n + 1), \qquad \mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{u}, \boldsymbol{x})_j = 0 \quad (j \geq 2n).$$

The first only renames carry wires; the second applies the convolution rule with $r = n + 1$ and $s = n$, using $x_j = 0$ for $j \geq n$.

For associativity, take $\boldsymbol{a} := \mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{x}, \boldsymbol{y})$ and let $\boldsymbol{u}$ be its outgoing carries. Here $A = N$ and $q = L + 1$. The corresponding chain is

$$\begin{aligned} a_j &= 0 \quad (j \geq 2n - 1), \\ \kappa_j &= 0 \quad (j \geq 2n + L), \\ u_j &= 0 \quad (j \geq 2n + L - 1), \\ \mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{u}, \boldsymbol{z})_j &= 0 \quad (j \geq 3n + L - 2). \end{aligned}$$

The first and last lines use (136), the second uses (149), and the third only renames carry wires. The other associativity deferral exchanges the input names and has the same bounds.

The parameter choice in Appendix A.5.4 gives $3n + 2L + 5 < N - 1$. Closed comparisons therefore place both cutoffs $2n$ and $3n + L - 2$ below $N - 1$, and justify the length conditions $r + s - 1 < N$ used above. Both premises of Lemma I.2 follow: the last coordinate of the auxiliary convolution is zero, and $\kappa_N = u_{N-1} = 0$ in each deferral. The missing zero checks, the constant number of carry and convolution derivations, and their folding cost $N \log^{O(1)} N$.

Vanishing final carries. The distributivity argument in the proof of Theorem A.2 uses Lemma I.1 (142) to turn carry resolution of a coefficient sum into addition of the two carry outputs. Besides the nonnegativity already proved, this rule requires the terminal carries of both coefficient arrays and of their sum to vanish, and the final carry of the output adder to vanish as well. The first three follow from the zero-suffix rule used above; the fourth is a statement about the adder, and we derive it from its full-adder gates. We give the argument for $\boldsymbol{U} := \mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{x}, \boldsymbol{y})$ and $\boldsymbol{V} := \mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{x}, \boldsymbol{z})$. Their entries are nonnegative, at most N, and zero for $j \geq 2n - 1$. Their sum has the same zero entries and entry bound $2N$. By Lemma I.3, the carries of $\boldsymbol{U}, \boldsymbol{V}$ vanish from index $2n + L$, and those of $\boldsymbol{U} + \boldsymbol{V}$ vanish from index $2n + L + 1$. These cutoffs are below N, so all three terminal carries are zero.

The output bits of $\mathsf{Carry}(\boldsymbol{U})$ and $\mathsf{Carry}(\boldsymbol{V})$ also vanish from index $2n + L$. In their binary adder, that column has two zero input bits; its incoming carry is a bit, so its outgoing carry is zero by the full-adder gates. All later carries are consequently zero as well. This establishes the remaining premise of (142). The same

argument applies to the other instance of that rule, on the sum $\boldsymbol{y} + \boldsymbol{z}$ of the original bit arrays, using their zero entries from index n onward.

The auxiliary triple convolutions in the associativity proof have entry bound N^2 and zero entries for $j \geq 3n - 2$ by (136). With $q = 2L + 1$, (149) makes their carries and output bits zero from index $3n + 2L - 1 < N$ onward as well.

These proofs reuse the carry and adder gates and add bound and zero checks of total circuit and proof cost $N \log^{O(1)} N$.

Returned bits and zero padding. To identify the array expressions below with the multiplier circuits in Theorem A.2, we apply Lemma I.4 first to the inner multipliers and then to the outer ones. Here we check its length condition for each application. With $r = s = n$ an inner product returns $\ell = 2n + L + 2$ bits; feeding that back as one input, a nested product returns $3n + 2L + 4$ bits. A product with an $(n + 1)$-bit sum, and a sum of two products, each return $2n + L + 3$ bits. All these lengths are below N by the same parameter comparison. If an input is extended by high zero bits, its length-N embedding is unchanged. With the global parameters fixed, Lemma B.4 identifies the resulting FFT, rounding, and carry outputs. The length checks, applications of Lemma I.4, and comparisons through the shared circuits cost $N \log^{O(1)} N$ for the constant number of instances. Bounds already derived above are reused.

Total cost of the bounds and padding. Each argument above introduces only a constant number of length-N arithmetic circuits and $O(N)$ checks on $O(W)$-bit words. Circuits and assertions already constructed are reused in later arguments. Including every missing circuit and every application of the cited rules, the total additional cost is $N \log^{O(1)} N$. The proofs in Appendices J.2 and J.3 cite the premises established here whenever they apply Lemmas H.1, I.1 and I.2.

J.2 Proof for Unsigned Inputs

The length-N bit embeddings of x, y, z are $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$. The first three items below use the coefficient bounds and zero entries from Appendix J.1; in particular, every convolution premise of Lemmas H.1 and I.2 invoked there was discharged above. The first two items also use Lemma I.4, with its length conditions checked in Appendix J.1. The FFT, rounding, and carry circuits belonging to the theorem's two sides are already supplied and are reused. The last two items, the two identities of the adder alone, use none of this material: they argue directly on the adder columns of the conclusion, and their cost is linear in n.

1. *Distributivity.* Name the existing outputs $\boldsymbol{u} := \mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{x}, \boldsymbol{y})$ and $\boldsymbol{v} := \mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{x}, \boldsymbol{z})$. First identify the second input to the left-hand multiplier:

$$\mathsf{Pad}_N\big(\mathsf{ADD}(y, z)\big) = \mathsf{Carry}(\boldsymbol{y} + \boldsymbol{z}). \tag{150}$$

   The full-adder cells prove this comparison column by column. The missing coefficient-sum and carry circuits, checks, and derivation cost $NW^{O(1)}$.

   Construct the intermediate convolutions, carry circuits, and full-length output adder in the following chain, reusing the theorem's multiplier and adder circuits. Padding adds only wiring. We compare the

two theorem outputs at length $N+1$, because adding two length-N carry outputs returns $N+1$ bits:

$$
\begin{aligned}
&\mathsf{Pad}_{N+1}\big(\mathsf{FFTMul}(x, \mathsf{ADD}(y,z))\big) \\
&= \mathsf{Pad}_{N+1}\big(\mathsf{Carry}(\mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{x}, \mathsf{Carry}(\boldsymbol{y}+\boldsymbol{z})))\big) && \text{(a) Lemma I.4; (150)} \\
&= \mathsf{Pad}_{N+1}\big(\mathsf{Carry}(\mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{x}, \boldsymbol{y}+\boldsymbol{z}))\big) && \text{(b) Lemma I.2 (146b)} \\
&= \mathsf{Pad}_{N+1}\big(\mathsf{Carry}(\boldsymbol{u}+\boldsymbol{v})\big) && \text{(c) Lemma H.1 (137a)} \\
&= \mathsf{ADD}(\mathsf{Carry}(\boldsymbol{u}), \mathsf{Carry}(\boldsymbol{v})) && \text{(d) Lemma I.1 (142)} \\
&= \mathsf{ADD}\big(\mathsf{Pad}_N(\mathsf{FFTMul}(x,y)), \mathsf{Pad}_N(\mathsf{FFTMul}(x,z))\big) && \text{(e) Lemma I.4} \\
&= \mathsf{Pad}_{N+1}\big(\mathsf{ADD}(\mathsf{FFTMul}(x,y), \mathsf{FFTMul}(x,z))\big) && \text{(f) full-adder gates.}
\end{aligned}
$$

In (a), Lemma I.4 identifies the padded multiplier output with its full carry array, and (150) identifies the bit embedding of its second input. In (e), the same lemma replaces the two full carry arrays by the padded product outputs. Lemma B.4 transfers these equalities, and those in (b)–(c), through their surrounding circuits. The required bounds, vanishing final carries in (d), and length conditions are supplied by Appendix J.1. The constant number of auxiliary convolution and carry circuits, their checks, and the derivations in (a)–(e) cost $N \log^{O(1)} N$ altogether.

For (f), put $\ell_0 := 2n + L + 2$, the width of each returned product. The adder on those two ℓ_0-bit words and the adder on their length-N zero-padded versions have matching first ℓ_0 columns. Compare their carry bits and output bits by the full-adder gates. At column ℓ_0, the longer adder receives two zero input bits and the shorter adder's final carry; it returns that carry as its output bit and sets its next carry to zero. All subsequent output bits and the final carry are zero. This proves (f), including its bit-equality checks, at cost $O(N)$.

Both theorem outputs have declared length $\ell_0+1 = 2n+L+3$. Apply Lemma B.3 to the padded-output equality, retain the equalities at these output positions, and apply Lemma B.2 to obtain the theorem's output-equality check. This removes the padding at additional cost $O(N)$. Including the input-adder comparison and the premise derivations in Appendix J.1, the total additional cost is $N \log^{O(1)} N$.

2. *Associativity of multiplication.* Put $\boldsymbol{A} := \mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{x}, \boldsymbol{y})$ and $\boldsymbol{B} := \mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{y}, \boldsymbol{z})$; $\boldsymbol{A}$ is the output already named $\boldsymbol{u}$, and both circuits are supplied. Reuse the four multiplier circuits in the theorem's two sides, and construct the missing intermediate outer convolutions, carry circuits, and equality checks in the following chain. Padding adds only wiring. We compare the two nested multiplier outputs after padding to length N:

$$
\begin{aligned}
&\mathsf{Pad}_N\big(\mathsf{FFTMul}(\mathsf{FFTMul}(x,y), z)\big) \\
&= \mathsf{Carry}\big(\mathsf{FFTConv}^{\mathrm{round}}(\mathsf{Pad}_N(\mathsf{FFTMul}(x,y)), \boldsymbol{z})\big) && \text{(a) Lemma I.4} \\
&= \mathsf{Carry}(\mathsf{FFTConv}^{\mathrm{round}}(\mathsf{Carry}(\boldsymbol{A}), \boldsymbol{z})) && \text{(b) Lemma I.4} \\
&= \mathsf{Carry}(\mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{A}, \boldsymbol{z})) && \text{(c) Lemma I.2 (146a)} \\
&= \mathsf{Carry}(\mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{x}, \boldsymbol{B})) && \text{(d) Lemma H.1 (137f)} \\
&= \mathsf{Carry}(\mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{x}, \mathsf{Carry}(\boldsymbol{B}))) && \text{(e) Lemma I.2 (146b)} \\
&= \mathsf{Carry}\big(\mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{x}, \mathsf{Pad}_N(\mathsf{FFTMul}(y,z)))\big) && \text{(f) Lemma I.4} \\
&= \mathsf{Pad}_N\big(\mathsf{FFTMul}(x, \mathsf{FFTMul}(y,z))\big) && \text{(g) Lemma I.4.}
\end{aligned}
$$

In (a) and (g), Lemma I.4 identifies each outer multiplier's padded output with its full carry array; its inner input is the length-N bit embedding of the returned inner product. In (b) and (f), the same rule identifies that padded inner product with $\mathsf{Carry}(\boldsymbol{A})$ or $\mathsf{Carry}(\boldsymbol{B})$. Lemma B.4 (50) transfers the

inner-product equalities through the outer convolution and carry circuits. It also transfers the coefficient equality in (d) through the carry circuit. The premises of (c)–(e) and the length conditions for all four padding-rule applications are supplied by Appendix J.1. The constant number of new convolution and carry circuits, their checks, and the cited derivations cost $N \log^{O(1)} N$ altogether.

Both theorem outputs have declared length $\ell := 3n + 2L + 4 < N$. Compose the chain using Lemma B.4 (55), then apply Lemma B.3 to its padded-output equality. Retain the equalities at positions $j < \ell$ and apply Lemma B.2 to obtain the theorem's output-equality check. These equality conversions and removal of padding cost $O(N)$. Including the premise derivations in Appendix J.1, the total additional cost remains $N \log^{O(1)} N$.

3. *Commutativity of multiplication.* By the definition of the multiplier in Appendix A.5, the two sides of this conclusion are the first ℓ bits of $\mathsf{Carry}(\mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{x}, \boldsymbol{y}))$ and of $\mathsf{Carry}(\mathsf{FFTConv}^{\mathrm{round}}(\boldsymbol{y}, \boldsymbol{x}))$. The symmetry identity Lemma H.1 (137b) equates the two coefficient arrays; its premise here is the pair of input bounds on $\boldsymbol{x}, \boldsymbol{y}$ supplied by the bounds step above. Lemma B.4 then carries the array equality through the carry gates. Fold the resulting output-bit equalities by Lemma B.2. No convolution or carry circuit beyond the conclusion's own is introduced, and the cited derivation costs $N \log^{O(1)} N$.

4. *Associativity of addition.* Name the incoming carries of the two adders on the left by α_j, β_j, and those on the right by α'_j, β'_j. Their sum invariant is

$$\alpha_j + \beta_j = \alpha'_j + \beta'_j.$$

It holds at column zero. Adding the two full-adder equations on either side expresses $x_j + y_j + z_j$ plus that side's incoming carry sum as its output bit plus twice its outgoing carry sum. The local quotient-and-remainder argument in the proof of Lemma I.1 therefore identifies the output bits and proves the next invariant. These carry sums have only two bits, so each column uses constant-size checks and proofs. Process the input and final carry columns, then fold the output bit equalities by Lemma B.2. The additional circuit and proof cost is $O(n)$.

5. *Commutativity of addition.* This identity was already proved in the roadmap (Section 4.1): with $w = n$, the column-by-column derivation there carries the invariant $\gamma_j = \gamma'_j$ between the carries of $\mathsf{ADD}(x, y)$ and those of $\mathsf{ADD}(y, x)$, identifies the output bits column by column, and folds the equality check by Lemma B.2. Nothing beyond the two adders of the conclusion is constructed, and the cost is $O(n)$.

The three conclusions involving the multiplier share total additional cost $N \log^{O(1)} N$: this includes the auxiliary constant certificates needed by the invoked FFT lemmas, every missing convolution or carry circuit, and the coefficient bounds and zero-entry proofs supplying their premises. Since $N = O(n + 1)$, this is $n \log^{O(1)}(n + 2)$. Their conclusion-circuit definitions have the same size, giving the same bound for the complete EF proofs. Items 4 and 5 bound each of the two conclusions of the adder by $O(n)$, and the circuits of those conclusions, two adders and one equality check, also have size $O(n)$, so both bounds are linear in n for these two identities.

J.3 Proof for Signed Integers

For signed n-bit two's-complement inputs, choose the global parameters using $n + 2$. Extend each input by one sign bit, then compute its nonnegative magnitude by conditional complement and addition of one; a zero test clears the sign of zero. Signed multiplication applies the unsigned FFT multiplier to the magnitudes, XORs their signs, and conditionally negates the product. Signed addition is the full-result sign-extended ripple adder.

An r-bit signed input produces an r-bit unsigned magnitude, using the extra sign bit during negation. A product of declared widths r, s returns $r+s+L+3$ signed bits: the unsigned multiplier's $r+s+L+2$ bits and one spare sign bit, followed by conditional negation. Signed addition returns $\max(r, s) + 1$ bits, and output comparisons sign-extend to a common width. The largest nested-product width is $3n + 2L + 6 < N - 1$, certified by the closed parameter check. Apply Lemma I.4 to each unsigned magnitude multiplier at its declared input lengths. High-zero padding of its inputs preserves the embeddings, as in Appendix J.1.

The extra circuits act on long binary words, so we prove their properties column by column, rather than applying short-word multiplication lemmas to $O(n)$-bit operands.

1. *Certify the sign and magnitude operations.* For a nonnegative word u and a bit b, let $E_b(u)$ denote conditional negation in a width with a spare sign bit. These circuits already occur in the signed implementation. Construct their round-trip checks and prove

$$\mathrm{mag}(E_b(u)) = u, \qquad \mathrm{sign}(E_b(u)) = b \wedge [u \neq 0]. \tag{151}$$

   The bit identity $u_j + (\neg u_j) = 1$ and the full-adder cells prove additive inversion and double negation through the columns; the spare sign bit and zero test give the two conclusions. The same cells show that equal-sign inputs add as $E_b(\mathsf{ADD}(u, v))$. For opposite signs, the borrow chain under $u \geq v$ constructs the nonnegative difference d and proves $\mathsf{ADD}(d, v) = u$; the signed sum is $E_b(d)$. The other comparison branch exchanges u, v, and $d = 0$ clears the sign.

   Sign extension commutes with these adder comparisons by a check of the high columns: equal input signs force the outgoing carry to that sign, while unequal signs preserve the incoming carry; in either case extra columns repeat the output sign bit. Every claim uses a constant-size Boolean derivation per column and a constant number of sign cases. The new difference, comparison, and round-trip circuits and their proofs have total cost $O(n)$ for the finitely many instances needed.

2. *Reduce signed distributivity to unsigned sums and differences.* For nonnegative $u \geq v$, reuse Step 1's difference d and its assertion $d + v = u$. Unsigned distributivity gives

$$\mathsf{FFTMul}(d, w) + \mathsf{FFTMul}(v, w) = \mathsf{FFTMul}(u, w).$$

   Here addition is the full binary adder. The adder and borrow-chain comparison from Step 1 identifies $\mathsf{FFTMul}(d, w)$ with the nonnegative difference of the two products; the displayed equality also supplies their order. Construct the missing unsigned products and their comparison circuits once.

   Split on the three input sign bits. For equal-sign summands, use unsigned distributivity and the additive-inverse column proof from Step 1. For opposite signs, split on their magnitude comparison and use the difference identity just proved. (151) identifies the resulting sign and magnitude with the specified signed output gates, including zero. Lemma B.5 combines this constant number of cases. A constant number of unsigned theorem instances and auxiliary multipliers costs $N \log^{O(1)} N$; the additional sign, borrow, and equality proofs cost $O(n)$. This proves signed distributivity within $N \log^{O(1)} N$ total cost.

3. *Prove both associativity identities.* For multiplication, use (151) for the sign and magnitude conversions and Lemma I.4 for padding the unsigned magnitude products. Lemma B.4 then identifies the endpoint magnitudes with the padded unsigned associativity circuits. Apply the unsigned result once. We also need that an unsigned multiplier returns zero if either input is zero. Under that condition, the zero-product and addition identities of Lemma C.1, (63e), (63f), and (62), propagate zero through the forward transform of that input, the pointwise products, and the inverse transform. The rounding and carry gates then return zero. The multiplier gates are reused; the missing zero checks and local derivations cost $N \log^{O(1)} N$ for the constant number of instances here.

If the common magnitude is zero, both signed outputs are zero. If it is nonzero, this zero-input property implies that all input magnitudes and both intermediate magnitude products are nonzero. Their zero tests therefore leave the signs unchanged, and both final signs are the XOR of the same three input sign bits. The fixed three-bit XOR proof, followed by Lemma B.5 on the zero check, gives signed output equality. The unsigned derivation and any missing magnitude circuits cost $N \log^{O(1)} N$; the new sign checks and proofs cost $O(n)$.

For addition, use the same two-bit carry-sum invariant as in the unsigned proof, on the sign-extended inputs. Step 1's high-column comparison handles differing intermediate widths. This adds $O(n)$ gates and proof steps.

4. *Prove commutativity of multiplication.* The two signed products apply the unsigned multiplier to the same pair of magnitudes in the two orders, so one instance of the unsigned identity just proved equates their magnitude products, and their zero tests then agree too. Each final sign is the XOR of the same two input sign bits, in the two orders, so a fixed two-bit check identifies them, and (151) identifies the two signed outputs. The unsigned derivation costs $N \log^{O(1)} N$, and the sign and zero-test checks cost $O(n)$.

5. *Prove commutativity of addition.* The signed adder is sign-extended before the ripple chain, so it is symmetric in its two inputs, and the roadmap derivation (Section 4.1) applies unchanged at the extended width, again at cost $O(n)$.

There are only a constant number of unsigned proof instances and $O(n)$ additional sign, carry, and comparison work. The three signed identities involving the multiplier consequently have the same $n \log^{O(1)}(n+2)$ additional and complete proof bounds. The two signed identities of the adder invoke no instance of the unsigned multiplier: the addition part of Step 3 and Step 5 derive them from the columns of the sign-extended adder, using only Step 1's high-column comparison, so their bounds remain $O(n)$.

Total proof size. The preceding constructions use $NL^{O(1)}$ short-word blocks, each of additional cost $W^{O(1)}$, including its auxiliary circuits and proof steps. The width calculations at their points of construction give $W = O(L^4)$. With $L = \log N$ and $N = \Theta(n+1)$, the total additional proof size is $NL^{O(1)}W^{O(1)} = N \log^{O(1)} N = n \log^{O(1)}(n+2)$. The conclusion circuits have the same asymptotic size by Appendix A.5.4, so including their definitions gives the same total bound for the complete EF proofs of Theorem A.2. This accounts for the three identities involving the multiplier. The two identities of the adder alone use no short-word block at all, and the bound for them is $O(n)$ in both the unsigned and the signed case, as recorded above.

Acknowledgments

Jiatu Li is supported by MIT Schwarzman College of Computing PhD Fellowship. Part of work is undertaken during the program Logical Foundations of Complexity Theory (LFC) organized by the Isaac Newton Institute for Mathematical Sciences, Cambridge; the LFC program was supported by EPSRC grant EP/Z000580/1. Andrew Krapivin and Elaine Shi are in part supported by a Packard Fellowship, a JP Morgan research award, and an Amazon research award. Andrew Krapivin was also supported by the Jeanne B. and Richard F. Berdik ARCS Pittsburgh Endowed Scholar Award.

AI Methodology

The paper's results stem from a continuous and long-running collaboration between the human authors and AI. The human authors are solely responsible for the conceptual contributions, including devising a language to describe EF proofs in a modular fashion, allowing for fine-grained proof size accounting. We had continuous interactions with ChatGPT 5.6, Claude Fable 5, Claude Fable 5.1 and ChatGPT 6 since spring 2026. AI assisted with devising the proofs; the index generalization idea (Section 3) was discovered after months of interactions with AI. AI also assisted with writing, exposition, polishing, and auditing throughout the paper. The authors verified the correctness and originality of all content, including references.

References

[BB93] Maria Luisa Bonet and Samuel R. Buss. The deduction rule and linear and near-linear proof simulations. *J. Symb. Log.*, 58(2):688–709, 1993.

[BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In Shafi Goldwasser, editor, *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, pages 326–349. ACM, 2012.

[BL19] Paul Beame and Vincent Liew. Toward verifying nonlinear integer arithmetic. *J. ACM*, 66(3):22:1–22:30, 2019.

[BN21] Sam Buss and Jakob Nordström. Proof complexity and SAT solving. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 233–350. IOS Press, 2021.

[BPR00] Maria Luisa Bonet, Toniann Pitassi, and Ran Raz. On interpolation and automatization for Frege systems. *SIAM J. Comput.*, 29(6):1939–1967, 2000.

[BSBHR19] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable zero knowledge with no trusted setup. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part III*, volume 11694 of *Lecture Notes in Computer Science*, pages 701–732. Springer, 2019.

[Bus86] Samuel R. Buss. *Bounded Arithmetic*, volume 3 of *Studies in Proof Theory, Lecture Notes*. Bibliopolis, Naples, 1986. Revision of the author's 1985 Ph.D. thesis, Princeton University.

[Bus95] Samuel R. Buss. Some remarks on lengths of propositional proofs. *Arch. Math. Log.*, 34(6):377–394, 1995.

[Bus97] Samuel R. Buss. Bounded arithmetic and propositional proof complexity. In Helmut Schwichtenberg, editor, *Logic of Computation*, pages 67–121. Springer, Berlin, Heidelberg, 1997.

[Bus21] Sam Buss. Substitution and propositional proof complexity. In Ivo Düntsch and Edwin Mares, editors, *Alasdair Urquhart on Nonclassical and Algebraic Logic and Complexity of Proofs*, Outstanding Contributions to Logic, pages 477–496. Springer, 2021.

[CLO24] Lijie Chen, Jiatu Li, and Igor C. Oliveira. Reverse mathematics of complexity lower bounds. In *65th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2024, Chicago, IL, USA, October 27-30, 2024*, pages 505–527. IEEE, 2024.

[CN10] Stephen Cook and Phuong Nguyen. *Logical Foundations of Proof Complexity*. Perspectives in Logic. Cambridge University Press and Association for Symbolic Logic, 2010.

[Coo75] Stephen A. Cook. Feasibly constructive proofs and the propositional calculus (preliminary version). In William C. Rounds, Nancy Martin, Jack W. Carlyle, and Michael A. Harrison, editors, *Proceedings of the 7th Annual ACM Symposium on Theory of Computing, May 5-7, 1975, Albuquerque, New Mexico, USA*, pages 83–97. ACM, 1975.

[CR79] Stephen A. Cook and Robert A. Reckhow. The relative efficiency of propositional proof systems. *J. Symb. Log.*, 44(1):36–50, 1979.

[CT65] James W. Cooley and John W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, 19(90):297–301, 1965.

[DHJQ26] Jesko Dujmovic, Yao-Ching Hsieh, Abhishek Jain, and Willy Quach. How to use polynomially-hard iO: Turing machine obfuscation and more. In Nadia Heninger and Mike Rosulek, editors, *Advances in Cryptology - CRYPTO 2026 - 46th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-20, 2026, Proceedings, Part I*, volume 16800 of *Lecture Notes in Computer Science*, pages 317–349. Springer, 2026.

[Für09] Martin Fürer. Faster integer multiplication. *SIAM Journal on Computing*, 39(3):979–1005, 2009.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, volume 7881 of *Lecture Notes in Computer Science*, pages 626–645. Springer, 2013.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 305–326. Springer, 2016.

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge. *IACR Cryptol. ePrint Arch.*, page 953, 2019.

[HJLM26] Yao-Ching Hsieh, Abhishek Jain, Jiatu Li, and Surya Mathialagan. SNARGs for NP from unprovability of mathematical theorems (or: How to use the simplicity of cryptographic reasoning). In Aditya Bhaskara and Artur Czumaj, editors, *Proceedings of the 58th Annual ACM Symposium on Theory of Computing, STOC 2026, Salt Lake City, UT, USA, June 22-26, 2026*, pages 31–41. ACM, 2026.

[HvdH21] David Harvey and Joris van der Hoeven. Integer multiplication in time $O(n \log n)$. *Annals of Mathematics*, 193(2):563–617, 2021.

[Jeř07] Emil Jeřábek. Approximate counting in bounded arithmetic. *J. Symb. Log.*, 72(3):959–993, 2007.

[JJ22] Abhishek Jain and Zhengzhong Jin. Indistinguishability obfuscation via mathematical proofs of equivalence. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 1023–1034. IEEE, 2022.

[JJMP25] Abhishek Jain, Zhengzhong Jin, Surya Mathialagan, and Omer Paneth. On succinct obfuscation via propositional proofs. In *66th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2025, Sydney, Australia, December 14-17, 2025*, pages 1703–1740. IEEE, 2025.

[JKLM25] Zhengzhong Jin, Yael Tauman Kalai, Alex Lombardi, and Surya Mathialagan. Universal SNARGs for NP from proofs of correctness. In Michal Koucký and Nikhil Bansal, editors, *Proceedings of the 57th Annual ACM Symposium on Theory of Computing, STOC 2025, Prague, Czechia, June 23-27, 2025*, pages 933–943. ACM, 2025.

[JKLV24] Zhengzhong Jin, Yael Kalai, Alex Lombardi, and Vinod Vaikuntanathan. SNARGs under LWE via propositional proofs. In Bojan Mohar, Igor Shinkar, and Ryan O'Donnell, editors, *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024, Vancouver, BC, Canada, June 24-28, 2024*, pages 1750–1757. ACM, 2024.

[JLP26] Zhengzhong Jin, Mingqi Lu, and Bo Peng. SNARKs from LWE via non-black-box reductions. In Aditya Bhaskara and Artur Czumaj, editors, *Proceedings of the 58th Annual ACM Symposium on Theory of Computing, STOC 2026, Salt Lake City, UT, USA, June 22-26, 2026*, pages 1812–1823. ACM, 2026.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In S. Rao Kosaraju, Mike Fellows, Avi Wigderson, and John A. Ellis, editors, *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada*, pages 723–732. ACM, 1992.

[KO17] Jan Krajíček and Igor C. Oliveira. Unprovability of circuit upper bounds in Cook's theory PV. *Log. Methods Comput. Sci.*, 13(1), 2017.

[KP90] Jan Krajíček and Pavel Pudlák. Quantified propositional calculi and fragments of bounded arithmetic. *Math. Log. Q.*, 36(1):29–46, 1990.

[KP98] Jan Krajíček and Pavel Pudlák. Some consequences of cryptographical conjectures for S_2^1 and EF. *Inf. Comput.*, 140(1):82–94, 1998.

[Kra95] Jan Krajíček. *Bounded arithmetic, propositional logic, and complexity theory*, volume 60 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, 1995.

[Kra19] Jan Krajíček. *Proof complexity*, volume 170. Cambridge University Press, 2019.

[LAH+22] Ning Luo, Timos Antonopoulos, William R. Harris, Ruzica Piskac, Eran Tromer, and Xiao Wang. Proving UNSAT in zero knowledge. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 2203–2217. ACM, 2022.

[Li25] Jiatu Li. An introduction to feasible mathematics and bounded arithmetic for computer scientists. Technical Report TR25-086, Electronic Colloquium on Computational Complexity, 2025. https://eccc.weizmann.ac.il/report/2025/086/.

[LKA+24] Daniel Luick, John C. Kolesar, Timos Antonopoulos, William R. Harris, James Parker, Ruzica Piskac, Eran Tromer, Xiao Wang, and Ning Luo. ZKSMT: a VM for proving SMT theorems in zero knowledge. In *33rd USENIX Security Symposium, USENIX Security 2024, Philadelphia, PA, USA, August 14-16, 2024*. USENIX Association, 2024.

[LO23] Jiatu Li and Igor C. Oliveira. Unprovability of strong complexity lower bounds in bounded arithmetic. In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 1051–1057. ACM, 2023.

[LOB24] Evan Laufer, Alex Ozdemir, and Dan Boneh. zkPi: proving lean theorems in zero-knowledge. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024*, pages 4301–4315. ACM, 2024.

[MDS25] Yaohua Ma, Chenxin Dai, and Elaine Shi. Quasi-Linear Indistinguishability Obfuscation via Mathematical Proofs of Equivalence and Applications. Cryptology ePrint Archive, Report 2025/307, 2025. Section 2.2. https://eprint.iacr.org/2025/307.pdf.

[Mic00] Silvio Micali. Computationally sound proofs. *SIAM J. Comput.*, 30(4):1253–1298, 2000.

[MP20] Moritz Müller and Ján Pich. Feasibly constructive proofs of succinct weak circuit lower bounds. *Ann. Pure Appl. Log.*, 171(2), 2020.

[PHGR13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pages 238–252. IEEE Computer Society, 2013.

[Pic15] Ján Pich. Logical strength of complexity theory and a formalization of the PCP theorem in bounded arithmetic. *Log. Methods Comput. Sci.*, 11(2), 2015.

[PS21] Ján Pich and Rahul Santhanam. Strong co-nondeterministic lower bounds for NP cannot be proved feasibly. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 223–233. ACM, 2021.

[PSW02] Matthew R. Pillmeier, Michael J. Schulte, and Eugene George Walters, III. Design alternatives for barrel shifters. In *Advanced Signal Processing Algorithms, Architectures, and Implementations XII*, volume 4791 of *Proceedings of SPIE*, pages 436–447. SPIE, 2002.

[Raz95a] Alexander A. Razborov. Bounded arithmetic and lower bounds in Boolean complexity. In Peter Clote and Jeffrey B. Remmel, editors, *Feasible Mathematics II*, pages 344–386. Birkhäuser, Boston, 1995.

[Raz95b] Alexander A. Razborov. Unprovability of lower bounds on circuit size in certain fragments of bounded arithmetic. *Izvestiya: Mathematics*, 59(1):205–227, 1995.

[SC04] Michael Soltys and Stephen A. Cook. The proof complexity of linear algebra. *Ann. Pure Appl. Log.*, 130(1-3):277–323, 2004.

[Set20] Srinath T. V. Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part III*, volume 12172 of *Lecture Notes in Computer Science*, pages 704–737. Springer, 2020.

[SS71] Arnold Schönhage and Volker Strassen. Schnelle Multiplikation großer Zahlen. *Computing*, 7(3–4):281–292, 1971.

[Tha22] Justin Thaler. Proofs, arguments, and zero-knowledge. *Found. Trends Priv. Secur.*, 4(2-4):117–660, 2022.

[Tse70] G. S. Tseitin. On the complexity of derivation in propositional calculus. In A. O. Slisenko, editor, *Studies in Constructive Mathematics and Mathematical Logic, Part II*, pages 115–125. Springer US, 1970. English translation of the 1968 Russian original (Zapiski Nauchnykh Seminarov LOMI, vol. 8).