Spain: Succinct proofs for numerical computations

Zachary DeStefano, Noah Golub, Zile Huang, Julius Zhang, Sam Frank, Michael Walfish

Courant Institute, NYU

Abstract. In a succinct proof protocol, a *verifier* gets assurance that an untrusted *prover* executed an agreed computation, without requiring the verifier to re-execute the computation itself. In little more than a decade, this area has undergone a remarkable transformation from theory to implemented systems. This activity is extremely exciting. But there is a catch. To apply succinct proofs, one needs to translate one's computation to a set of equations, or *constraints*. The required translation has so far completely blocked systematic support for *numerical* computations, namely those for which the bulk of the computation uses approximations of real numbers. This paper fills that void with the design, implementation, and evaluation of a system called Spain. The starting insight of Spain is that since numerical computations inherently have approximation error, the constraint formalism should likewise allow for *approximate* satisfiability. Based on this insight, Spain introduces a new proof protocol and new ways to translate computations to constraints. Spain's implementation improves over natural baselines by multiple orders of magnitude.

1 Introduction

A *succinct proof protocol* provides execution integrity: one party (the *verifier*) gets assurance that another party (the *prover*) executed an agreed computation, without requiring the verifier to redo the computation or trust the prover. A variant of this setup is zero-knowledge, which additionally hides sensitive inputs from the verifier. Note that execution integrity is orthogonal to program verification. Program verification is about ensuring that a given program meets a specification; here, the question is whether the alleged outputs of a program, verified or otherwise, truly came from running that program.

Succinct proofs have undergone a remarkable transformation. These constructs were known to theorists, based on foundational results in interactive proofs, zero-knowledge proofs, arguments, and probabilistically checkable proofs (PCPs) in the 1980s and 1990s [13, 14, 16, 17, 34, 65, 80, 89, 91, 120]. At the time, they were considered to be wildly impractical. However, over the past 15 years, these constructs have been refined and implemented (see [130, 140] for surveys). They have even been deployed [69, 126], mostly in cryptocurrency [54, 55, 72, 73, 84, 88, 103]. For example, in a traditional blockchain, every node re-executes all transactions to check the validity of state transitions; with succinct proofs, by contrast, a prover generates a proof once and records it on the blockchain, and other nodes need only verify the proof [133].

Despite all of the progress, there is an inescapable awkwardness in applying succinct proofs. One must *arithmetize* one's computation, which means translating it to equations, or *constraints*; roughly speaking, a solution to the constraints corresponds to a valid program trace. Unfortunately, this process does not match how software developers write programs. Control flow, for example, is unnatural to represent as constraints. Making matters worse, the constraints are expressed over a finite field (such as the integers mod a given prime q), yet finite fields have no notion of negative numbers or even order relationships. Consequently, a simple program operation like conditionally branching based on a comparison between two numbers has a very verbose expression in constraints (§2).

Still, researchers and practitioners manage to get around the semantic gap for certain small-scale computations (like validating blockchain transactions). The verifier is truly fast, and the prover's overhead is "only" $10^6\times$ relative to executing the computation natively (§7, §8).

However, the semantic gap becomes a yawning chasm for *numerical* computations: those for which the bulk of the computation uses fixed-point or floating-point approximations of real numbers. The core issue is that numerical computations don't map naturally to equations over finite fields, not even with the kinds of gymnastics used for non-numerical computations. Not to mention, researchers have no hope of competing with the decades of investment in optimizing FPUs and GPUs.

Yet, numerical computations were one of the primary motivations for the invention of computers, and continue to be of intense interest. In the modern era, LLM training and inference, physics simulations, cyberphysical systems, and far more all compute over approximations of real numbers. Succinct proofs applied to numerical computations would thus allow such computations to run on untrusted infrastructure. For example, a user of an LLM could get a proof that the produced tokens truly came from running inference on a given model. Or someone could run a fluid simulation and then prove to others that the simulation ran as specified, without anyone having to redo the computation.

The purpose of this paper is to produce a system that applies succinct proofs to numerical computations, with three goals:

1. *The system should be general-purpose.* This means that the core proving machinery should not be specialized to the computation itself; in particular, if the computation changes, the core protocol should not have to change, nor should the pencil-and-paper mathematics that undergird the correctness of the succinct proof protocol.
2. *The verifier should be less expensive computationally than native execution.* Otherwise, the verifier could simply run the computation itself. Note that in zero-knowledge setups, this requirement does not arise, as the verifier

is not expected to be able to run the computation itself. However, zero-knowledge is a non-goal for us.

3. *The prover's overhead is no more than three orders of magnitude versus natively executing the computation.* While in ordinary computing contexts, overhead of 1000× would be preposterous, in this research area, the state of the art, including in deployed systems, is $10^5\times$ or $10^6\times$.

Prior works encoding numerical computations in proof systems do one of two things (§8). Some works give standalone encodings of specific numerical operations, embedded in general-purpose proof systems; these arguably meet goal 1 but at the cost of goal 3 [9, 50, 51, 59, 82, 115, 124]. Other works build end-to-end special-purpose protocols for specific numerical computations, sacrificing goal 1 – and in most cases goal 3, too [19, 63, 77, 78, 86, 127, 128, 151]. In fact, only one work that we are aware of has achieved goal 3 [105], but at severe compromise of goals 1 and 2.

This paper describes a system, *Spain*, that achieves all three goals in certain regimes. Spain makes several contributions:

A succinct proof framework for approximate rational arithmetic (§3). Spain's idea here is to reflect existing numerical notions of accuracy by using constraints but having the proving machinery establish that each constraint holds *approximately*.

A new proving protocol (§4). To actually prove that the constraints hold approximately, Spain introduces a new proof protocol. It is built on and inspired by prior work, specifically Spartan [116], Zaratan [42], and DARK [40].

Highly compact arithmetizations (§5). Rather than paying a number of constraints proportional to the number of bits in numerical operations, Spain translates individual numerical operations to *single constraints*; for example, division and square root become one constraint each. This in turn enables highly efficient comparisons, range checks, and piecewise functions. Spain also uses division to create efficient translations of transcendental operations, such as e^x. These encodings could be of independent interest.

Rigorous proof (Apps. B, E, F). We prove correctness of Spain's proof protocol and its arithmetizations.

Implementation of Spain (§6, Appx. G). We have implemented Spain. It includes several *front-ends* that translate numerical computations to constraints and a common *back-end* that implements the Spain prover and verifier.

Experimental evaluation (§7). We evaluate Spain and baseline systems on various applications, including linear programming problems; machine learning primitives and an end-to-end application (GPT-2); fluid simulations; and geospatial calculations. We find that Spain has the fastest (general-purpose) numerical prover in the literature. Additionally, Spain's verifier is the first that we are aware of to beat native execution for reasonably-sized numerical computations.

Like all succinct proof systems, Spain must pay high overheads. However, in exhibiting a new kind of arithmetization, Spain has substantially expanded what is possible.

2 Background

2.1 Numerical computations

People want to perform computations that involve real-numbered arithmetic. However, doing so exactly is often intractable. Instead, numerical programs are implemented with *approximate arithmetic*, which comes in two flavors: fixed-point and floating-point. Both have some rounding, or *error*. *Numerical analysis* studies how to ensure that the final result is meaningful.

Fixed-point arithmetic encodes a subset of the rational numbers (denoted $\mathbb{Q}$), specifically those expressible as an integer divided by a fixed denominator (normally a power of two). Real numbers outside of fixed-point are mapped to nearby fixed-point numbers via rounding, prior to entering fixed-point operations. Operations obey absolute error bounds. For any two fixed-point numbers x and y and primitive operation op (for example, $+$, $-$, $\times$, or $\div$), the fixed-point operation op_δ is defined (ignoring overflow) as:

$$(x \ \text{op}_\delta \ y) = (x \ \text{op} \ y) + \widehat{\delta_\pm},$$

where $|\widehat{\delta_\pm}| \leq \delta$ for some fixed δ that depends on the operation and the fixed-point format.

With floating-point representations and arithmetic, operations obey relative error bounds. Then with op as above, x, y as floating-point numbers, and op_δ now denoting the floating-point operation:

$$(x \ \text{op}_\delta \ y) = (x \ \text{op} \ y) \cdot (1 + \widehat{\delta_\pm}),$$

where $|\widehat{\delta_\pm}| \leq \delta$. Even simple operations introduce errors. For example, when using single-precision IEEE 754 [2] floating point, $(2 \div_\delta 5) \approx 0.400000006$ rather than exactly 0.4.

2.2 Succinct and probabilistic proofs

A *succinct proof* (or *probabilistic proof*) is a cryptographic protocol between a *prover* and a *verifier* (typically thought of as probabilistic algorithms) about a *statement* that the prover wants to persuade the verifier of [64, 130]. We will in this work consider statements of the form: "(in, out) is a valid input-output pair for some procedure F." That is, the succinct proof is aimed at establishing that the alleged *out* is truly the output when F is run on *in*.

Astonishingly, the data flowing from prover to verifier, and the work required by the verifier, is (at least in principle) much smaller than the work to execute F. Yet, a verifier is unlikely to be fooled by a false claim from the prover.

Succinct proof implementations typically have a *front-end* and a *back-end*. The front-end compiles F into a set of *constraints*. The back-end is the proving and verifying algorithms. We elaborate on both below.

Front-end: Arithmetization and R1CS. Various proof back-ends require statements to be compiled, by the front-end, to a *rank-one constraint system* (R1CS), which we often

refer to as *constraints*. An R1CS *structure* is a system of m equations and n variables over a finite field (typically $\mathbb{F}_q$, the integers mod a prime q) with a subset of the variables designated as *in* and a subset designated as *out* [118]. These equations are represented by three matrices A, B, C, each of dimension $m \times n$. An R1CS *instance* for a given structure is (A, B, C, in, out); we say that an instance is *satisfiable* if there exists some *witness* w such that, for $z := (in, out, 1, w)$, we have: $(A \cdot z) \circ (B \cdot z) - (C \cdot z) = \vec{0}$, with $\circ$ denoting entry-wise multiplication. Unpacking the algebra, each constraint $i \in \{1, \ldots, m\}$ restricts any *assignment*, that is any valuation of the variables $z = (z_1, \ldots, z_n)$, as follows: $(A_{i,1}z_1 + \cdots + A_{i,n}z_n) \cdot (B_{i,1}z_1 + \cdots + B_{i,n}z_n) - (C_{i,1}z_1 + \cdots + C_{i,n}z_n) = 0$.

As a simple example, take the computation

function $F(z_1, z_2, z_3)$
  $z_4 \leftarrow (z_1^2 - 4) \cdot (z_2 + 3z_3) - 2$
  return z_4

Here *in* is (z_1, z_2, z_3) and *out* is z_4. The first step in translation to R1CS is to unroll the computation so that each line of code is written as a product of linear combinations of variables, minus another linear combination of variables:

function $F(z_1, z_2, z_3)$
  $z_6 \leftarrow z_1 \cdot z_1 - 4$ // z_6 is a new var, part of the witness
  $z_4 \leftarrow z_6 \cdot (z_2 + 3z_3) - 2$
  return z_4

Next, these lines are translated to constraints, each of which becomes a row in the A, B, C matrices. These constraints are:

$$(1 \cdot z_1) \cdot (1 \cdot z_1) - (1 \cdot z_6 + 4 \cdot 1) = 0$$
$$(1 \cdot z_6) \cdot (1 \cdot z_2 + 3 \cdot z_3) - (1 \cdot z_4 + 2 \cdot 1) = 0.$$

In matrix form, the constraints are:

$$A = \begin{array}{c} \begin{matrix} z_1 & z_2 & z_3 & z_4 & z_5 & z_6 \end{matrix} \\ \left[\begin{array}{ccc|c|c} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right] \end{array}$$

$$B = \left[\begin{array}{ccc|c|c} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 3 & 0 & 0 & 0 \end{array}\right]$$

$$C = \begin{array}{c} \left[\begin{array}{ccc|c|c} 0 & 0 & 0 & 0 & 4 & 1 \\ 0 & 0 & 0 & 1 & 2 & 0 \end{array}\right] \\ \begin{matrix} in & & & out & 1 & w \end{matrix} \end{array}$$

Now, consider this valuation of the *in* variables: z_1=3, z_2=2, z_3=−1 (these quantities are in $\mathbb{F}_q$, so −1 is shorthand for $q - 1$), and the corresponding *out* valuation: $z_4 = -7$ (shorthand for $q - 7$). In this case, the full assignment is $z = [3, 2, -1, -7, 1, 5]^\intercal$. When subsequently giving example constraints, we drop the explicit "1·", add the "C piece" to both sides, and use semantically appropriate variables to refer to elements of z.

One can translate an entire unrolled computation to a set of constraints, in the sense that some z satisfies the constraints only if z respects the original semantics of the computation. This translation is called *arithmetization* [7, 17, 18, 24, 35, 36, 49, 75, 81, 96, 97, 115, 118, 138, 146, 147]. It is typically accompanied by a *witness generator*, which either executes the computation and thereby obtains the assignment to z [15, 24, 26, 76, 81, 102], or else solves the constraints using annotations provided by the translation process [35, 75, 96, 100, 114, 115].

Modern proof systems also translate computations into *lookup tables* [31, 33, 41, 57, 71, 119, 119], often in combination with constraints. However, lookup tables outperform constraints only when the table is small (for example, 2^8 elements) or has algebraic structure [119]. Numerical operations do not generally fit into either of these categories.

Back-end: Proving R1CS satisfiability. The job of the back-end is, given an R1CS structure and instance (A, B, C, in, out), to prove that a satisfying assignment exists. The back-end provides the following properties, which we state informally.

- *Back-end soundness:* If there does not exist a w such that $z := (in, out, 1, w)$ makes $(A \cdot z) \circ (B \cdot z) - (C \cdot z) = 0$, then the probability (over the verifier's random choices) that the verifier accepts is negligible.
- *Back-end completeness:* If a prover has access to w such that $z := (in, out, 1, w)$ makes $(A \cdot z) \circ (B \cdot z) - (C \cdot z) = 0$, then the prover can make the verifier accept with probability 1, over the prover's and verifier's random choices.

For our work, the relevant strand of back-end protocols is based on the *sum-check* primitive [89]. We defer justifying this choice until we have established the necessary context (§4.1, §4.2.2). The sum-check primitive is an interactive proof [16, 65] by which the prover persuades the verifier that the sum of a given multi-variate polynomial's evaluations over all Boolean combinations that its variables can take is a given value. (Appendix A reviews this primitive.) Succinct proofs that use the sum-check primitive typically do so multiple times within the same protocol; for example, GKR [66, 67] invokes the sum-check primitive once per layer in a layered circuit while Spartan [116] invokes the primitive twice (loosely speaking, one invocation is over the rows of the R1CS instance and one is over the columns). We call proofs built on the sum-check primitive *sum-check protocols* [30, 42, 44–46, 48, 60, 66–68, 116, 117, 119, 123, 129, 135–137, 139, 143, 147, 148].

Modern works in this strand [131] rely on *polynomial commitment*. The idea is that w is encoded as a polynomial by the prover; a small *commitment* (far smaller than w) is sent to the verifier; this commitment binds the prover to that polynomial; and one or more invocations of the sum-check primitive prove that an additional polynomial (formed from the prover's committed polynomial, from *in* and *out*, and from the R1CS structure itself) has a certain property that is equivalent to $z = (in, out, 1, w)$ satisfying the instance.

The costs of the back-end are driven by the *number of constraints*, m and the *number of witness elements*, $|w|$. Specifically, the prover's costs are roughly $O(m + |w|)$. The verifier's costs vary based on the protocol. Our work follows Spartan [92, 116] (without its SPARK module), where the verifier

has an $O(m)$ *fixed cost* that is specific to the structure (the constraints); this cost amortizes over synchronized instances, with each instance adding cost that is $O(\log(|w|))$. There is also an $O(\log(|w|))$ *setup cost*, reusable across all future invocations, including different R1CS structures. See elsewhere [38, 116, 117, 119] for techniques that asymptotically lower the verifier's work at the prover's expense.

To quantify, running the prover on a single machine is generally considered unreasonable when m and $|w|$ are above 2^{29} (unless one possesses terabytes of RAM and hundreds of cores) [108], owing to memory bottlenecks. There are techniques for scaling across a fleet [87, 108, 142, 144], and reducing memory requirements at the cost of increased prover time [20, 93, 99], but neither approach mitigates the sheer amount of work required to prove large computations.

Arithmetization poses a semantic challenge. A salient cost is translating from ordinary computations to R1CS; this appears in the values of m and $|w|$. Indeed, there is a massive semantic gap in this research area: ordinary computations do not map to equations over finite fields in a natural way. To highlight the challenge, consider the division of two positive k-bit integers. Checking this computation, $z \leftarrow x/y$, in R1CS involves two steps. First, the prover supplies the purported quotient z and remainder r and the following constraint is added: $y \cdot z = x - r$, where r is a witness variable. Second, constraints are required to assert that $x \geq (z+1)$ and $y \geq (r+1)$.

But finite fields have no native notion of order, so a typical arithmetization of assert$(x \geq (z+1))$, where x and z are k-bit numbers, introduces witness variables $b_0, \ldots, b_{k-1}$ representing the bits of $x - (z+1)$. Then, for all i, constraints are added to enforce that each variable b_i is a bit: $b_i \cdot (1 - b_i) = 0$. Finally, a constraint is added to enforce the relationship among x, z, and the bits: $\sum_{i=0}^{k-1} b_i 2^i = x - z - 1$. This set of constraints is satisfiable only if $x - z - 1$ is itself a k-bit number. This assert alone requires k additional variables and $k + 1$ constraints.

This blowup is not isolated to division. Researchers in succinct proofs often perform contortions to represent computations. While there are, for example, more succinct encodings of certain kinds of conditional operations [115], arbitrary low-level operations of the kind that would ordinarily be accelerated in hardware are not well-handled. In particular, representing a single numerical operation in R1CS typically involves explicitly expressing the digital logic of the operation. This results in a number of auxiliary variables and constraints that is a multiple of the *number of bits of the numbers being represented* in each operation [9, 50, 51, 82, 124] (§7, §8), rather than the *number of machine instructions that would be required in an ordinary computing context*.

3 Problem statement and overview of Spain

Problem statement. A prover and a verifier agree on a single numerical computation, F. The verifier sends input *in*, the prover executes F on that input, and claims that the output is *out*. The prover then wants to persuade the verifier that *out* indeed could have been produced by $F(in)$, given the kinds of approximation that are inherent in fixed- or floating-point numerical computation. The verifier should spend fewer computational resources than if it ran $F(in)$ itself; otherwise, the verifier could disregard *out*, and simply use the result of executing F on *in*. We will sometimes work in an amortized setting where multiple instances of F are outsourced on $in_1, \ldots, in_L$ different inputs, getting different outputs $out_1, \ldots, out_L$.

The core conflict. On the one hand, existing proof machinery ensures that an execution proceeded *exactly* as it was supposed to. On the other hand, in the numerical context "supposed to" means "certain *error*". To resolve this conflict between exactness and error, any protocol must reject truly wrong executions, namely those that include any operations with more than δ error (§2.1). At the same time, the protocol must not be too much of a scold: if all operations do obey the δ error bound, then the given execution should be accepted.

As noted in Section 2.2, some works handle this tension by representing the digital logic of the approximate computation in the arithmetization (the R1CS instance). A related approach is for the prover to embed in the R1CS instance variables that capture each operation's error, thereby making the error incurred by each operation part of the statement to be validated [59]. Both approaches bring expense.

Overview of Spain. In contrast to existing work, Spain's arithmetizations are agnostic to the error in each operation, yet the prover is forced to adhere to the bounds. Spain combines several new techniques, in the front-end and back-end. Figure 1 depicts Spain, contrasting it with the most natural baseline.

Front-end (§5): Spain introduces a new kind of arithmetization; it translates F to a set of constraints over the *rational numbers* $\mathbb{Q}$ (rather than a finite field $\mathbb{F}_q$) in a way that satisfying all constraints up to some ϵ corresponds to error up to δ in each operation. Each constraint $i \in \{1, \ldots, m\}$ enforces: $\left|\left(A_{i,1}z_1 + \cdots + A_{i,n}z_n\right) \cdot \left(B_{i,1}z_1 + \cdots + B_{i,n}z_n\right) - \left(C_{i,1}z_1 + \cdots + C_{i,n}z_n\right)\right| \leq \epsilon$. We call such constraints *approximate constraints*; compare to the constraints in §2.2 that enforce equality, which we call *traditional constraints*.

Approximate constraints are dramatically more concise than the alternative; in fact, each low-level operation in F generally corresponds to one or a small number of constraints. The intuition is that constraints in Spain are freed from operating on the bits of operands, being expressed over $\mathbb{Q}$, and are freed from explicitly encoding approximation error, as "$\leq \epsilon$" encodes slackness directly. The trade-off is that a user of Spain must perform certain analyses not needed when working with traditional constraints. This requirement is a manifestation of the conflict stated earlier: the user must show that satisfying the constraints means the prover adhered to "$\leq \delta$" error, and also show that if the prover does execute the operation with bounded error, then it *can* satisfy the approximate constraints.

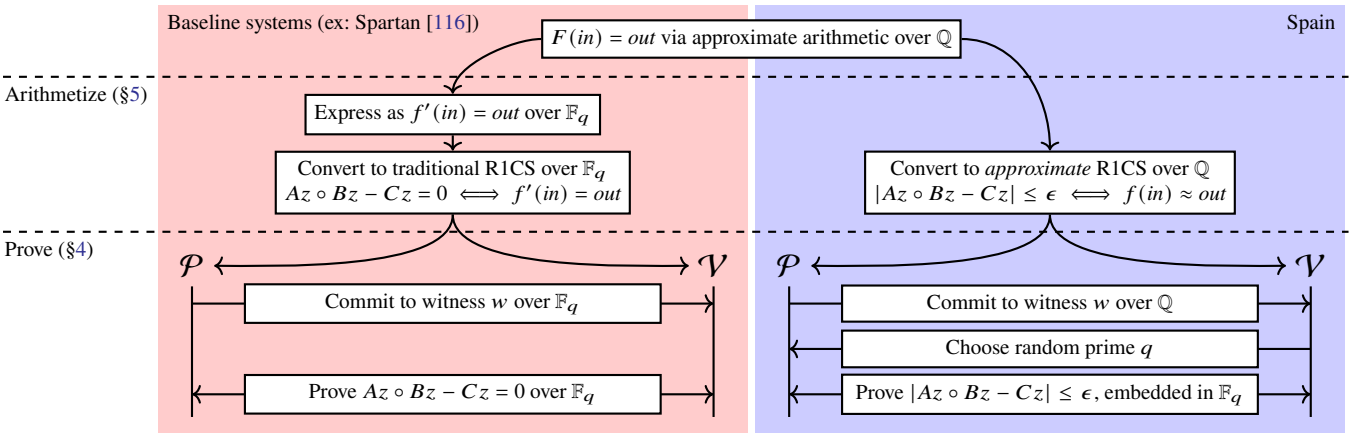


Figure 1: High-level comparison between existing proof systems (left) and Spain (right). Given an initial claim, the protocol is divided into two phases: arithmetization and proving. Similar steps are vertically aligned to facilitate comparison.

Back-end (§4): Spain's back-end enables a prover to establish that a given assignment (to a given set of constraints) meets the ϵ bounds. We call such an assignment ϵ-*accurate*.

Compared to prior work, Spain's back-end has three interlocked aspects. First, Spain maps the constraints over $\mathbb{Q}$ to a finite field, which is the kind of domain typically used in sum-check protocols (§2.2). However, preserving the semantics of the constraints in the finite field requires care; a key mechanism is that the verifier chooses the finite field (via prime q) only after the prover commits to the assignment.

Second, Spain establishes a statement that implies ϵ-accuracy. Specifically, Spain modifies Spartan [116] to prove a statement about the sum of each constraint's squared error, for a given assignment; this sum will be notated as $\|E_{\mathbb{X},z}\|_2^2$. As we argue later, by proving that $\|E_{\mathbb{X},z}\|_2^2$ is small, the protocol guarantees that the assignment obeyed the required ϵ bounds. Third, Spain uses a polynomial commitment protocol (§2.2) that is geared to integers, but makes several technical adjustments, as naive use of this primitive in our context would be prohibitive.

4 Spain's approximation-friendly back-end

This section presents the core statement that Spain's back-end proves (§4.1) and the techniques that it uses to do so (§4.2–§4.3). First, though, we must define what it means for a back-end to be correct in Spain's context. We do so by modifying the existing back-end notions of soundness and completeness (§2.2). The modified properties are as follows, with respect to an R1CS instance $\mathbb{X} = (A, B, C, in, out)$:

- *Back-end ϵ-soundness.* If there does not exist a w such that $z := (in, out, 1, w)$ is ϵ-accurate for $\mathbb{X}$, then the probability κ (over the verifier's random choices) that the verifier accepts is negligible. κ is known as *soundness error*.
- *Back-end ϵ_C-completeness.* A prover with access to w such that $z := (in, out, 1, w)$ is ϵ_C-accurate for $\mathbb{X}$ can make the verifier accept with probability 1, over the prover's and verifier's random choices in the protocol.

These definitions embed two different levels of ϵ-accuracy; at the end of Section 4.1 below, we will see that $\epsilon_C < \epsilon$. To preview the practical consequence, an honest prover will have to execute with greater precision (for completeness) than what the protocol guarantees to the verifier (via soundness).

4.1 What claim about error should be proved?

Spain's back-end must establish the ϵ-accuracy of an assignment z. Before highlighting the challenges, we give a compact notation for ϵ-accuracy. Define the *error vector* $E_{\mathbb{X},z}$ for an R1CS instance $\mathbb{X} = (A, B, C, in, out)$ and an assignment $z = (in, out, 1, w)$ as:

$$E_{\mathbb{X},z} := (A \cdot z) \circ (B \cdot z) - (C \cdot z).$$

This error vector has m components. Its ith component is $(A_i \cdot z) \cdot (B_i \cdot z) - (C_i \cdot z)$, where A_i, B_i, and C_i are the ith rows of A, B, and C. Notice that if the component with maximum absolute value is upper-bounded by ϵ, that is equivalent to saying that the assignment z for the instance $\mathbb{X}$ is ϵ-accurate (§3). Meanwhile, the maximum absolute value over all components of $E_{\mathbb{X},z}$ is, by definition, the ℓ^∞ norm of $E_{\mathbb{X},z}$, denoted $\|E_{\mathbb{X},z}\|_\infty$. Thus, ϵ-accuracy can be written:

$$\|E_{\mathbb{X},z}\|_\infty \leq \epsilon.$$

One challenge is that we don't know how to apply proving machinery to this inequality directly. Sum-check protocols (§2.2), for example, work over polynomial expressions. Although Spartan [116] shows how to cast R1CS satisfiability as suitable polynomial expressions, the ℓ^∞ norm cannot be turned into polynomial expressions, because of the absolute value and maximum operations.

Instead, our insight here is that *the square of the ℓ^2 norm* is compatible with sum-check protocols, and this quantity upper-bounds the square of the ℓ^∞ norm. Specifically, for a vector v,

the square of the ℓ^2 norm, $\|v\|_2^2$, is $\sum_{i=1}^{m} v_i^2$. Also, the definitions of the norms imply that for any vector v, $\|v\|_\infty^2 \leq \|v\|_2^2$. So, if the prover could persuade the verifier that

$$\|E_{\mathbb{X},z}\|_2^2 = J, \tag{1}$$

for some J, and if the verifier then checked that $J \leq \epsilon^2$, that would suffice to establish that $\|E_{\mathbb{X},z}\|_\infty \leq \epsilon$.

However, there is another difficulty. While $\|E_{\mathbb{X},z}\|_2^2 \leq \epsilon^2$ implies $\|E_{\mathbb{X},z}\|_\infty \leq \epsilon$, the converse is not true. Consequently, an assignment z could well satisfy $\|E_{\mathbb{X},z}\|_\infty \leq \epsilon$, but Equation (1) would not hold for any $J \leq \epsilon^2$. In a full protocol, an honest prover would then have no way to convince the verifier.

To address this, the execution by the prover has to be more accurate than ϵ. Specifically, while Spain must meet Back-end ϵ-soundness, it must meet Back-end ϵ_C-completeness, where $\epsilon_C = \epsilon/\sqrt{m}$. (The gap between ϵ and ϵ_C can be narrowed; §D.3.) Under these conditions, there does exist $J \leq \epsilon^2$ with $\|E_{\mathbb{X},z}\|_2^2 = J$. To see this, notice that the definitions of the norms imply that for any vector v, $\|v\|_2^2 \leq m \cdot \|v\|_\infty^2$. Consequently, requiring that the prover produces an ϵ_C-accurate z yields $\|E_{\mathbb{X},z}\|_\infty \leq \epsilon_C = \epsilon/\sqrt{m}$ and thus:

$$\|E_{\mathbb{X},z}\|_2^2 \leq m \cdot \|E_{\mathbb{X},z}\|_\infty^2 \leq m \cdot \epsilon_C^2 \leq \epsilon^2.$$

4.2 Main protocol

Spain's back-end aims to establish Equation (1). Spain adapts three existing tools, specifically *Spartan* [116], which is a sum-check protocol (§2.2) that targets R1CS instances, and forms the core of Spain's back-end; *DARK* [40], which is a polynomial commitment protocol (§2.2) with the ability to bind the prover to a polynomial over $\mathbb{Q}$; and *Zaratan* [42], which is a framework for proving traditional R1CS satisfiability over the integers, as opposed to over a finite field.

One of the configurations explicitly considered by Zaratan is combining Spartan and DARK, and we will do likewise. Specifically, whereas Spartan assumes that an R1CS structure is defined over a finite field (typically $\mathbb{F}_q$; §2.2), Zaratan observes that q can be chosen at run-time by the verifier, after the prover commits to an encoded version of w via DARK.

Spain's full protocol is given in Appendix B. Figure 2 provides an overview. This protocol embeds three major changes compared to prior work, as outlined below.

4.2.1 Proving statements over $\mathbb{Q}$

Constraints in Spain are expressed over the rational numbers, $\mathbb{Q}$. However, the sum-check primitive is typically applied over a finite field (§1, §2.2). Accordingly, Spain maps Equation (1) into a corresponding claim over a finite field, namely $\mathbb{F}_q$. But, this mapping is fraught; to highlight the challenge, we introduce some mathematical language.

Define $\mathbb{Q}^{(q)}$ as $\{(a, b) \in \mathbb{Q} \mid q \nmid b\}$. This set is the rational numbers without multiples of q in the denominator.[1] Spain

[1] Mathematically, this set is known as a localization [141], and could be written $(\mathbb{Z} \setminus q\mathbb{Z})^{-1}\mathbb{Z}$, or $\mathbb{Z}_{(q)}$.

1. The prover runs an agreed-upon computation F, thereby producing w.
2. Using DARK, the prover encodes w as a polynomial, $\widetilde{w}$; *commits* (§2.2) to that polynomial; and sends this commitment to the verifier, effectively binding itself to the value of $E_{\mathbb{X},z}$.
3. The prover computes and sends J to the verifier; J is purportedly $\|E_{\mathbb{X},z}\|_2^2$.
4. The verifier checks that J is less than ϵ^2.
5. The verifier sends a prime q to the prover randomly chosen from a set of large primes.
6. The prover and verifier map all rationals to elements of $\mathbb{F}_q$ and apply the sum-check protocol to the claim $\rho_q(\|E_{\mathbb{X},z}\|_2^2) = \rho_q(J)$, reducing it to a claim about $\widetilde{w}$.
7. The verifier checks consistency between the claim in Step 6 and the commitment to $\widetilde{w}$ in Step 2.

Figure 2: Spain's back-end protocol (simplified).

maps these numbers into a finite field, via:

$$\begin{aligned} \rho_q \colon \mathbb{Q}^{(q)} &\longrightarrow \mathbb{F}_q \\ (a, b) &\longmapsto (a \bmod q)(b \bmod q)^{-1}, \end{aligned}$$

and the Spain prover and verifier apply a sum-check protocol to the claim:

$$\rho_q(\|E_{\mathbb{X},z}\|_2^2) = \rho_q(J). \tag{2}$$

However, for this approach to make sense, there are two requirements. First, arithmetic must stay in $\mathbb{Q}^{(q)}$, otherwise ρ_q is undefined. Second, unequal quantities in $\mathbb{Q}^{(q)}$ should stay unequal after the mapping. That is, if $\|E_{\mathbb{X},z}\|_2^2 \neq J$, then $\rho_q(\|E_{\mathbb{X},z}\|_2^2) = \rho_q(J)$ should not occur, except with negligible probability (over the verifier's choice of q).

Spain achieves both requirements through a combination of convention, DARK, and the verifier's checks. The starting convention is that all rational numbers in the A, B, C matrices that define an R1CS instance $\mathbb{X}$ are presumed to have a denominator D that is a power of two; thus q does not divide D. Furthermore, the assignment z is supposed to follow this convention as well. DARK partially preserves the convention, by ensuring that all committed rationals (Step 2) have the same denominator but not necessarily that it is exactly D. Surprisingly, this partial preservation does not give the prover room to cheat without detection (§B). For simplicity, our description below assumes that DARK preserves the convention fully.

One consequence is that arithmetic begins in $\mathbb{Q}^{(q)}$, and stays there. Thus, ρ_q is defined on all intermediate values and the result. To see this, notice that the convention ensures that the entries of $A \cdot z$, $B \cdot z$, and $C \cdot z$ have denominator D^2; the entries of $(A \cdot z) \circ (B \cdot z)$ have denominator D^4, and the values of $((A_i \cdot z) \cdot (B_i \cdot z) - (C_i \cdot z))^2$, and hence $\|E_{\mathbb{X},z}\|_2^2$, have denominator D^8. The verifier completes the enforcement by requiring J to have denominator D^8.

Another consequence is that this approach avoids collisions. If $\|E_{\mathbb{X},z}\|_2^2$ and J are not equal, then given that they have the same denominator, $\rho_q(\|E_{\mathbb{X},z}\|_2^2) = \rho_q(J)$ only if the numerator N of $\|E_{\mathbb{X},z}\|_2^2 - J$ is a multiple of q. Analysis shows that by limiting the numerators of the entries of A, B, C, and z, N can be kept small enough that the probability of a randomly chosen large prime, like q, dividing N is low (Appx. B.5).

Both Spain and Zaratan [42] map a claim from a larger domain (in their case $\mathbb{Z}$, in Spain's case $\mathbb{Q}$) to some $\mathbb{F}_q$. However, the introduction of a denominator in Spain significantly complicates the analysis of collisions.

The bottom line is that Spain ensures that if the prover sends a $J \neq \|E_{\mathbb{X},z}\|_2^2$ (Step 3, Figure 2), then with overwhelming probability over the choice of q in Step 5, $\rho_q(\|E_{\mathbb{X},z}\|_2^2) \neq \rho_q(J)$. Consequently, if the prover did lie in Step 3, it would be stuck trying to prove a false statement in Step 6.

4.2.2 Proving a new type of claim

To prove that $\rho_q(\|E_{\mathbb{X},z}\|_2^2) = \rho_q(J)$, Spain observes that an adaptation of Spartan can directly prove statements about the squared ℓ^2 norm of the errors in an R1CS instance.

To explain how Spain both borrows and diverges from Spartan, we must present some further mathematical detail. One can view $A, B, C \in \mathbb{F}_q^{m \times n}$, and $z \in \mathbb{F}_q^n$ as *functions*. Specifically, letting $s = \lceil \log m \rceil$ and $k = \lceil \log n \rceil$, we view the matrices as functions $\{0,1\}^s \times \{0,1\}^k \to \mathbb{F}_q$. For example, $A(r, c)$ is the entry of A at the row given by the binary representation of r and the column given by the binary representation of c. We do the same for z, viewing it as a function $\{0,1\}^k \to \mathbb{F}_q$.

An important concept is a polynomial *extension* of a discrete function. The polynomial extension of a function agrees with that function everywhere the function is defined, but the polynomial is defined over a larger domain. A polynomial extension can be thought of as an *encoding* of that function. As a simple example, imagine defining a function f at points 0 and 1, and drawing a line (a first-degree polynomial) through $f(0)$ and $f(1)$. The entire line – its description, or any two points on it – encodes the values of $f(0)$ and $f(1)$. We denote *multilinear extensions* with tildes. For example, for a function $f(\cdot)$ defined over $x \in \{0,1\}^s$, the multilinear extension of f is $\tilde{f} : \mathbb{F}_q^s \to \mathbb{F}_q$, which is a polynomial that equals f on $\{0,1\}^s$, with degree no more than 1 in each variable.

Now, let M stand in for A, B, or C, and let $\widetilde{g_M}(t) := \sum_{c \in \{0,1\}^k} \widetilde{M}(t, c) \cdot \widetilde{z}(c)$, where $\widetilde{M}$ and $\widetilde{z}$ are the multilinear extensions of M and z, respectively (when mapped to $\mathbb{F}_q$). Also, let $Eq(\cdot,\cdot)$ return 1 if its two arguments, viewed as s-bit strings, are the same and 0 otherwise; denote the multilinear extension of Eq as $\widetilde{Eq}$.

In Spartan (and Zaratan), the prover aims to persuade the verifier of the following, where τ is chosen randomly [116, §4]:

$$\sum_{r \in \{0,1\}^s} (\widetilde{g_A}(r) \cdot \widetilde{g_B}(r) - \widetilde{g_C}(r)) \cdot \widetilde{Eq}(\tau, r) = 0. \qquad (3)$$

In Spain, by contrast, the prover aims to persuade the verifier (in Step 6) that the following holds:

$$\sum_{r \in \{0,1\}^s} (\widetilde{g_A}(r) \cdot \widetilde{g_B}(r) - \widetilde{g_C}(r))^2 = \rho_q(J). \qquad (4)$$

Notice from the definition of extension that the left-hand side above sums the squared error for each constraint, given assignment z, when the constraints and z are mapped to $\mathbb{F}_q$. That sum is $\rho_q(\|E_{\mathbb{X},z}\|_2^2)$. Thus, the protocol is indeed establishing Equation (2). After this step, Spain follows Spartan.

4.2.3 Accelerating and adapting DARK

Spain makes two sets of modifications to DARK. First, whereas DARK normally works over a group of order unknown to both prover and verifier, Spain exploits interactivity to have the verifier generate a group for which it alone knows the order. Appendix C details this technique.

Second, DARK as originally proposed had an error: it provided only a weak binding property, namely it ensured only that the committed-to polynomial had *rational* coefficients [29]. This was problematic for Zaratan and other uses of DARK (for example, in combination with Spartan), as those uses require a polynomial with *integer* coefficients. Thus, Zaratan (and other protocols) layer techniques [29, 42] atop DARK, which have extra costs. However, in our context, the original (weaker) form of DARK suffices. Appendix B gives details.

4.3 Support for batching and "just in time" R1CS

Spain supports two variants of R1CS. The first is a *SIMD-R1CS instance* [134], which is a combination of L R1CS instances with the same structure (that is, multiple executions of the same program on different inputs). The second is an *I-R1CS instance* [97], where the prover and verifier interact to construct the full instance. Given a partial R1CS instance (one with some variables missing), the prover and verifier take turns filling it in. In round i, the prover commits to w_i, and the verifier sends additional in_i. The final R1CS instance has assignment $z := (in_1, \ldots, in_R, out, 1, w_1, \ldots, w_R)$.

Spain extends I-R1CS in a natural way: in Spain, the verifier may also supply *constraints*. Thus, the prover and verifier start with a partial structure, and as the prover commits to w_i in round i, the verifier submits in_i or new constraints. Further details of Spain's support for R1CS variants are in Appendix D.

5 Encoding numerical operations in R1CS

This section describes how, by taking advantage of the backend claim of ϵ-accuracy (§4), Spain produces dramatically more concise arithmetizations versus the traditional approach.

Using approximate constraints (§3) requires a new kind of correspondence between a numerical function F and an

R1CS structure that allegedly arithmetizes F. Roughly speaking, (a) any assignment that is ϵ-accurate (§3) for the constraints should correspond to an execution of F with operations bounded by δ (§2.1), and (b) for every execution of F with operations bounded by δ_{wg} (where δ_{wg} is a constant smaller than δ) there should be an ϵ_C-accurate assignment. The need for separate δ and δ_{wg} reflects the same asymmetry that leads to $\epsilon_C < \epsilon$ (§4). Appendix E formalizes (a) and (b) as *translation fidelity*, and proves that the constraints described in this section meet these properties.

Note that translation fidelity concerns the error in *each* operation, not the accumulated error in the output of a computation. This is analogous to how the IEEE floating-point specification [2] bounds the error in each operation and relies on numerical analysis to derive bounds on the accumulated error.

Below we present arithmetizations using approximate constraints, notating them with $\approx_\epsilon$. For example, $z_1 \cdot z_2 \approx_\epsilon z_3$ enforces $|z_1 \cdot z_2 - z_3| \leq \epsilon$.

Division and square root. Recall that checking $z \leftarrow x/y$ traditionally requires dozens of constraints (§2.2). Spain uses *one* constraint: $y \cdot z \approx_\epsilon x$.

In traditional constraints, the square root operation ($y \leftarrow \sqrt{x}$) is encoded as $y \cdot y = x - r$ plus dozens of constraints to bound r in terms of x and y. Spain again requires only one constraint: $y \cdot y \approx_\epsilon x$. This takes advantage of the fact that all real square roots have arbitrarily close rational approximations. If one wants a specifically positive or negative root, one applies the square root operation again! For example, suppose we want y to be the negative square root of x. Then we use an additional constraint: $t \cdot t \approx_\epsilon -y$, exploiting the fact that $-y$ must be non-negative to have a real square root.

Note that this encoding of the square root operation can be satisfied when x is a small negative number, in contrast to usual numerical computations, where Not a Number (NaN) would arise. If this is problematic, the constraints and/or numerical analysis should ensure that the function's input is strictly non-negative.

The power of approximate square roots. In traditional arithmetization, bottlenecks include range checks, branching, max, and min. As we show next, Spain arithmetizes approximate versions of these operations orders of magnitude more efficiently, using square roots.

assert ($\mathbf{x} \geq \mathbf{y}$). Recall from Section 2.2 that this operation required k additional variables and $k+1$ constraints. Spain requires only 1 constraint and 1 additional variable, regardless of bit width: $t \cdot t \approx_\epsilon x - y$, for some witness variable t. This is not a strict translation of assert but instead an approximate version that allows x to be ϵ smaller than y but no smaller.

$\mathbf{b} \leftarrow (\mathbf{x} \geq \mathbf{y})$. Spain requires only 3 constraints and 2 auxiliary variables: $b \cdot (1-b) \approx_\epsilon 0$, $(2b-1) \cdot (x-y) \approx_\epsilon t$, and $s \cdot s \approx_\epsilon t$. The first constraint ensures that b is Boolean. The second ensures that t is non-negative only when b correctly reflects the comparison. The third ensures that t is indeed non-negative. As with square root and assert, this comparison is an approximate version; Appendix E.2 delves into the semantics.

max. Consider $z \leftarrow \max(v)$ for an array $v = [v_1, \ldots, v_L]$. Spain's approach here is inspired by Distiller [76], which encodes a *checker* for min, rather than embedding logic to identify the minimum. However, Distiller requires $L \cdot (k+3)+1$ constraints and $L \cdot (k+2)$ witness variables, where k is the bit-widths of the elements in the array. Spain's are far more concise because it pays so little for comparison operations:

- for all $i \in \{1, \ldots, L\}$: $t_i \cdot t_i \approx_\epsilon z - v_i$
- for all $i \in \{1, \ldots, L\}$: $b_i \cdot (1 - b_i) \approx_\epsilon 0$
- for all $i \in \{1, \ldots, L\}$: $b_i \cdot (z - v_i) \approx_\epsilon 0$
- $\sum_{i=1}^{L} b_i \approx_\epsilon 1$

The first set ensures that z is greater than or equal to all elements in the array. The next two sets ensure that b_i is Boolean and that b_i can be 1 only if $v_i = z$ (to support multiple equal maxima, b_i is allowed to be 0 even when $v_i = z$). The last ensures that exactly one of the b_i is 1. This requires $3 \cdot L + 1$ constraints and $2 \cdot L$ auxiliary variables.

Piecewise functions, sorting, and beyond. The encoding of comparisons naturally leads to efficient translations of piecewise functions; ReLU will demonstrate this below. Similarly, the previous best solutions for sorting or for string manipulation paid relative to the number of bits in their inputs due to comparisons, while Spain's approach more closely matches the number of instructions in a typical CPU execution.

ReLU is a primitive in machine learning contexts. The operation is $z \leftarrow \max(0, x)$. This function can be represented in constraints as $s \cdot s \approx_\epsilon z - x$, $t \cdot t \approx_\epsilon z$, and $(z - x) \cdot z \approx_\epsilon 0$. The first two ensure that $z \geq \max(0, x)$. The last one ensures that $z = 0$ or $z = x$.

Transcendental functions. Arithmetizing transcendental functions, such as e^x or $\tanh(x)$, over some interval traditionally uses *polynomial approximation* [50, 51, 78, 82, 105, 124, 128, 151]. Specifically, for a function $f(x)$ to be approximated, one identifies a polynomial $P(x)$ such that $|P(x) - f(x)| \leq \delta$ for x in the given interval. The motivation is that add and multiply, which are the operations that polynomials require, are the easiest to represent in traditional constraints.

However, there is an additional issue (besides the overhead from traditional enforcement of numerical operations in constraints): polynomials require many terms to converge. Drawing on its support for division, Spain uses *rational approximations*: a ratio of polynomials $P(x)/Q(x)$ such that $|P(x)/Q(x) - f(x)| \leq \delta$ for x in the interval. In Spain, rational approximations cost nearly nothing extra over polynomial approximations, but rational approximations converge faster. To illustrate, compare the degree-4 Taylor series for e^x:

$$g(x) = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24}$$

to the degree-$[4/4]$ Padé approximant [98]:

$$h(x) = \frac{1 + \frac{x}{2} + \frac{x^2}{10} + \frac{x^3}{120} + \frac{x^4}{1680}}{1 - \frac{x}{2} + \frac{x^2}{10} - \frac{x^3}{120} + \frac{x^4}{1680}}.$$

Note that the number of constraints for the two approaches is the same (for translating $h(x)$, Spain memoizes the powers of x). One can compare accuracy (using a computer algebra system or a large piece of paper), and consider the size of the intervals for which $|g(x) - e^x| \leq \delta$ and $|h(x) - e^x| \leq \delta$ respectively. In this example, take $\delta = 0.01$. The Taylor Series (g) has error at most 0.01 over the interval $[-1.07, 1.00]$ and larger errors outside this interval. The Padé approximant (h) has error at most 0.01 over $[-8.11, 2.84]$. In fact, the degree $[2/2]$ Padé approximant has error at most 0.01 over $[-2.18, 1.16]$: better than the degree-4 Taylor series.

Better approximations (more accuracy, wider interval) for the same number of terms also translate to fewer variables and hence fewer constraints to achieve the same accuracy. Going beyond exponentials, the technique of rational approximation works for a large class of functions.

Concurrent work [124] makes a similar observation; they find that for highly precise approximations, arithmetizations (of high-degree polynomials) can be even more expensive than traditional arithmetizations of division. These authors then incorporate Padé approximations to existing proof pipelines, obtaining a small speedup. Their paper contains many intriguing examples of approximation using this technique. All of these examples can be used in Spain directly, only with lower cost, because division is so inexpensive in Spain.

6 Implementation of Spain

6.1 Back-end implementation

Spain's back-end implements the protocol described in Section 4.2. It also supports SIMD-R1CS and I-R1CS (§4.3).

Spain implements fixed-size large integer arithmetic with inline limb storage; this avoids the performance cost of existing large integer libraries' [3–5] heap allocation of limb arrays. This functionality is used to compute and check J (Steps 3 and 4 in Fig. 2). Spain also implements 64- and 128-bit finite field arithmetic, with support for fast type conversion from large integers (Step 6); Spain does not use existing finite field libraries [11] because they optimize at compile time for a specific field, whereas in Spain, the finite field is determined only at run time. Finally, Spain implements fixed-size RSA group arithmetic, to do fast modular arithmetic for moduli not known at compile time (Steps 2 and 7).

Spain's back-end is implemented in 12,135 lines of Rust, including the new libraries.

6.2 Front-end implementation

We implemented three front-ends for Spain.

Gadget front-end. This front-end provides a library of gadgets [113, 150] (pairs of functions where the first generates constraints and the second generates witness values satisfying them), including the primitives in Section 5. The framework is modular and extensible.

ONNX front-end. ONNX [95, 132] is a specification for machine learning models. In ordinary use, one describes a model in ONNX's format (a collection of nodes), and then a highly optimized run-time executes the model on the available hardware. Spain's ONNX front-end translates from the ONNX format to approximate constraints. For the accompanying witness generator (§2.2), we had hoped to reuse ONNX-compatible run-times and tools; however, existing run-times neither produce satisfying assignments to auxiliary variables of the kind that arise in arithmetization (§5) nor run with the high-precision floating-point types that Spain uses (§6.3).

Spain solves this challenge with two sub-components. The first is a *model translator* that converts an ONNX model to a verbose version of the model, whose nodes produce output that includes the auxiliary information. In the process, the model translator rewrites transcendental functions to use rational approximations (§5), using the textbook implementation of the Remez algorithm [104]. The translator also modifies the operations in the model to specify higher-precision data types (for example, double- or quadruple-precision floating point instead of single-precision floating point). The second sub-component is a from-scratch ONNX executor that supports these high-precision data types. This executor implements each ONNX operator that appears in Spain's benchmarks (§7).

Linear programming front-end. This front-end is specialized to linear programming problems for comparison with prior work (§7), specifically Otti [9].

Spain's front-ends are implemented in, respectively, 2898 lines of Rust; 4747 lines of Python and 2025 lines of Rust; and 1524 lines of Rust.

6.3 Numerical considerations

A user of Spain begins with a particular computation F and desired error δ. Note that the choice of δ is exogenous to Spain, and should be driven by numerical considerations (§2.1). The user must then determine: (a) ϵ, to inform the verifier's check of J (§4), (b) δ_{wg}, the precision at which the prover must run F (§5, Appx. E), and (c) D, the denominator used in the arithmetization of F (§4.2.1, Appx. B). The general recipe for doing so is as follows. First, translate F to R1CS. Next, perform numerical analysis on each operation in the translation and compose the analyses to obtain an upper bound on ϵ in terms of δ (Appx. E.2). This ϵ, along with the number of constraints, m, upper-bounds ϵ_C (§4). Finally, returning to the numerical analysis, ϵ_C enforces an upper bound on δ_{wg} and on $1/D$, and thus a lower bound on D (Appx. E.2).

One could perform this last step analytically; essentially one would determine the maximum magnitude of values in the computation over all possible inputs, and then choose small-

enough δ_{wg} and $1/D$ to ensure each constraint has error upper-bounded by ϵ_C. For convenience, we instead guess (by running the witness generator at an estimated high precision) and check (by observing that it produces a witness for which $\|E_{\mathbb{X},z}\|_2^2 \leq \epsilon$). This approach is not dangerous: choosing δ_{wg} and $1/D$ too large – meaning underestimating the required precision – cannot affect soundness (which holds regardless of the prover's behavior). Furthermore, if there were such an underestimate, the prover could recalibrate and run witness generation at higher precision for all subsequent proof generations; witness generation is not the primary bottleneck for the prover (§7.2). For the same reason, accidentally overestimating precision does not unduly increase costs.

In Section 7, we provide values of δ, ϵ, ϵ_C, and D for the benchmarks in our experiments. Here, we work through their derivation for one example, a fluid simulation with $\delta = 2^{-32}$.

By analyzing the translation to R1CS (Appx. E), we get that $\epsilon = \delta/4$ suffices: an ϵ-accurate assignment to the constraints ensures absolute error of $\delta = 2^{-32}$. Supposing that $m \leq 2^{26}$, the prover needs a witness with error $\epsilon_C \leq \epsilon/\sqrt{2^{26}} = 2^{-47}$ (§4.1). For witness generation, this example uses double-precision floating-point arithmetic, which has *relative* error about 2^{-53}. In this application, the magnitude of intermediate values is sufficiently small that this relative error keeps the absolute error of each operation (δ_{wg}) below 2^{-47}. Note that the next choice up, quadruple-precision floating-point arithmetic, has relative error 2^{-112}, and would suffice if the intermediate values were larger.

Furthermore, most constraints are satisfied with error far smaller than ϵ_C, so the "closeness" of δ_{wg} and the floating-point relative error is not a concern in practice, and one can even run witness generation in a way where *some* constraints violate ϵ_C provided their totality satisfies $\|E_{\mathbb{X},z}\|_2^2 \leq \epsilon$. To counterbalance δ_{wg} being close to ϵ_C, this example chooses $1/D$ to be significantly smaller than ϵ_C: 2^{-70}.

Spain's back-end scales these floating-point values and converts them to large fixed-width integers to commit via DARK and to compute J exactly (Steps 2 and 3, Figure 2). For these steps, Spain combines 256-, 512-, and 768-bit integer arithmetic. Spain then maps these integers to field elements to execute the sum-check protocol steps (Step 6, Figure 2).

7 Experimental evaluation

We aim to answer the following questions experimentally:

1. How does Spain perform compared to the best proof systems for numerical computations?
2. What are the contributions to Spain's costs?
3. In what regimes does Spain meet the performance goals stated in Section 1?

Baselines. We compare Spain to five baselines, listed below. Two are end-to-end general-purpose systems, two are synthetic general-purpose systems with back-ends very similar to Spain's (for isolating the effect of Spain's front-end), and one is a special-purpose system.

Otti [9]: This is a general-purpose proof system in which fixed-point numerical computations are translated to finite field operations (§2.2). Otti aims at linear programming and semi-definite programming. Otti's back-end is a non-interactive and zero-knowledge variant of Spartan [116] so Otti offers qualitative properties that Spain does not (§7.4). Our experiments run Otti's released code [10].

ZKLP [51]: This general-purpose proof system encodes IEEE 754 floating-point logic in I-R1CS (§4.3, §D.2). ZKLP's back-end is the gnark [32] implementation of Groth16 [70], which (like Otti's back-end) is non-interactive and zero-knowledge. Our experiments run ZKLP's released code [52].

Otti-FE: This baseline uses Otti's front-end, meaning the same constraints as in Otti, and couples it with Spain's back-end implementation (including DARK), adjusted to prove a statement in the form of Equation (3) (§4.2.2). This setup results in a small fraction of Otti's constraints not being satisfied, because they rely on inverses over Otti's original finite field, rather than the one that Spain uses for sum-check, which is smaller. To be pessimistic to Spain, we filter out these constraints in our measurements.

ZKLP-FE: This baseline uses the same back-end as Otti-FE. For the front-end, we adopt a synthetic approach because ZKLP's front-end is not compatible with Spain's back-end, and making it so would have been a significant engineering effort. To apply this baseline to a benchmark, we count the operation types in the benchmark; for each operation type, we borrow constraint counts from ZKLP, and use the total number of constraints to construct an R1CS structure. Although this structure is semantically meaningless, its *size* (which is what drives performance) reflects the expected number of constraints that would be produced by using ZKLP's arithmetization of single-precision floating-point computations.

zkGPT [105]: This baseline is a special-purpose proof system for GPT-2 inference (described below). We choose it because it is, to our knowledge, the fastest application-specific prover for any numerical computation in the literature; however, the proof system itself mirrors the structure of GPT-2 inference, so it cannot be adapted to other computations.

Benchmarks. Our benchmarks are in four families:

Linear Programming. This family includes several linear programming instances from the Netlib library [61] that are used by Otti [9] (specifically, adlittle, afiro, sc105, scagr7, and scsd8).

Machine Learning. This family includes common machine learning primitives (Softmax, LayerNorm, and GELU) as well as the inference phase of GPT-2 [106].

Softmax maps $x = [x_1, \ldots, x_L]$ to $y = [y_1, \ldots, y_L]$ as follows: $y_i \leftarrow e^{x_i}/(\sum_{j=1}^{L} e^{x_j})$, for $i \in \{1, \ldots, L\}$. Spain arithmetizes this computation with Remez approximations of $e^{x_1}, \ldots, e^{x_L}$ (§5). This microbenchmark runs on each row of a 32×32 tensor.

F	δ	ϵ	ϵ_C	D
Linear Programming	2^{-32}	2^{-32}	$2^{-38.26}$	2^{50}
Machine Learning	2^{-20}	2^{-40}	$2^{-46.75}$	2^{75}
Fluid Simulation	2^{-32}	2^{-34}	$2^{-44.27}$	2^{70}
Geolocation	2^{-13}	2^{-26}	$2^{-35.10}$	2^{70}

Figure 3: Numerical parameters for benchmark families. ϵ_C varies within a family as it depends on the number of constraints m (§4.1) and batch size L (§4.3, §D.1) for the particular instance(s). We report the minimum ϵ_C across the family. δ_{wg} is omitted because all witness generation is performed using double- or quadruple-precision floating point, which has relative rather than absolute error guarantees (§6.3). Specifically, the Linear Programming, Fluid Simulation, and Geolocation families use double-precision, and the Machine Learning family uses quadruple-precision.

LayerNorm involves computing the mean and variance of a vector and then normalizing the vector. Specifically, $x = [x_1, \ldots, x_L]$ is transformed to $y = [y_1, \ldots, y_L]$ via $y_i \leftarrow \frac{x_i - \mu}{\sqrt{v+\delta}}$, where μ and v are the mean and variance of the inputs x, and δ provides numerical stability. Spain arithmetizes this benchmark with square root, division, and multiplication operations. We run this microbenchmark on a 32×768 tensor.

GELU is a popular activation function used in GPTs, defined as $x \cdot \Phi(x)$, where $\Phi(x)$ is the Gaussian CDF. Spain arithmetizes this with a piecewise linear approximation (§5). We run this microbenchmark on a 32×3072 tensor.

GPT-2 [106] is an open source LLM that uses several machine learning primitives, including Softmax, LayerNorm, GELU, and matrix multiplication (MatMul). For MatMul, Spain includes an optimization not described earlier: Spain translates to a *checker* that uses the Freivalds algorithm [56]. This requires an approximate version of Freivalds [74] together with Spain's extension to I-R1CS (§4.3, Appx. D.2). This benchmark runs the inference phase of GPT-2, and is parameterized by a length of input tokens (*seq*), and a number of iterations (*passes*).

Fluid Simulation. This family is a Stable-Fluids-style [125] incompressible fluid simulation that approximately solves the 2D Navier-Stokes equations over square grids. Spain arithmetizes this benchmark with multiplication, max, and min operations. We benchmark 8×8 and 16×16 grids, each simulated for 10 timesteps, denoted *small* and *large* respectively. Even at such small sizes, these benchmarks are computationally intensive; simulations of this form often run on GPUs in practice.

Geolocation. This family includes the geospatial computation in ZKLP [51], which checks consistency between spherical coordinates (latitude, longitude) and coordinates in a hierarchical Discrete Global Grid System [110] called Uber H3 [37]. This benchmark required simulating modular arithmetic, by arithmetizing the $\lfloor \cdot \rfloor$ operation.

Figure 3 gives the numerical parameters for each family.

Metrics, measurement, and method. We report the proving time, verification time, constraint counts (m, from §2.2), and proof size of Spain and baseline systems on the benchmarks above. We also report the cost of native execution. In addition, we measure the memory usage of Spain's prover, and fixed and setup costs for the Spain verifier.

We run all experiments in single-threaded configurations, on CPUs. We measure time based on the wall clock, with a harness that runs and times all phases of Spain. The exception is witness generation. That is timed differently depending on the benchmark. For the Linear Programming benchmarks, we time a custom Rust program that solves the linear program and uses the solution to generate the witness values. For ONNX benchmarks, we time the execution of the transformed ONNX model on Spain's high-precision ONNX executor (§6.2). For Fluid Simulation, similar to the Linear Programming benchmarks, we time a custom Rust program that generates witness values.

We report the average of at least 5 runs for each experiment. The standard deviations are within 11% of the means.

Testbed. For all experiments, we run the verifier on a 2.1 GHz 64-Core AMD Opteron 6272 with 256 GB RAM, running Red Hat Enterprise Linux 9.8. To reflect the motivation for succinct proofs, we run the prover on a more powerful machine: a 64-Core Intel Xeon Platinum 8592+ with 3 TB RAM, running Red Hat Enterprise Linux 9.6. When running Otti and ZKLP on our hardware, we use the artifacts provided by the authors.

7.1 Spain versus baselines

To compare Spain to the baselines, we run the aforementioned benchmarks. For ZKLP-FE run on GPT-2, the constraint counts are too large to run the back-end. Instead, we estimate prover and verifier time, as well as proof size, by executing ZKLP-FE on synthetic instances of various sizes and extrapolating times linearly ($R^2 > 0.99$ for both).

Figure 4 depicts the results. Spain shows dramatic improvement in number of constraints (m). This owes to the techniques in Section 5; for example, with the Linear Programming benchmarks, the baseline needs verbose constraints to represent comparisons ($\geq$, $\leq$, and so on) whereas Spain does not.

The lowered m for Spain yields prover speedups of 8–2700×, bringing the prover's overhead down in some cases to our goal of 3 orders of magnitude (§1). In fact, notice that Spain's back-end is *worse* than the baselines on a per-constraint basis; for example, Otti outperforms Otti-FE (which has a Spain-like backend) when running with substantially the same constraints. Yet, the reduction in m is more than enough to compensate. Indeed, the only baseline that Spain's prover does not exceed is zkGPT. But zkGPT does not apply beyond that one benchmark, and it has qualitative disadvantages, described later (§7.3).

Similarly, Spain's verifier improves on all baselines except zkGPT (see above) and ZKLP. ZKLP's back-end is specifically aimed at verifier performance, but it has high setup cost (§7.3, §7.4). Spain's improvement over all other baselines

			Constraints (m)			Prover time				Verifier time			Proof size		
Benchmark	**Baseline**	**Native time**	**Baseline**	**Spain**	**Ratio**	**Baseline**	**Spain**	**Speedup**	**Spain/Native**	**Baseline**	**Spain**	**Speedup**	**Baseline**	**Spain**	**Ratio**
adlittle	Otti	2.5 ms	181,000	292	620×	930 ms	62 ms	15×	69×	480 ms	17 ms	29×	29 KB	9.3 KB	3.1×
adlittle	Otti-FE	2.5 ms	142,000	292	490×	7.8 s	62 ms	130×	69×	190 ms	17 ms	11×	16 KB	9.3 KB	1.8×
afiro	Otti	420 μs	36,800	111	330×	270 ms	33 ms	8.2×	180×	160 ms	14 ms	11×	20 KB	7.4 KB	2.7×
afiro	Otti-FE	420 μs	29,100	111	260×	2.1 s	33 ms	62×	180×	59 ms	14 ms	4.3×	15 KB	7.4 KB	2.0×
sc105	Otti	3.1 ms	113,000	372	300×	490 ms	62 ms	8.0×	64×	320 ms	17 ms	19×	21 KB	9.3 KB	2.2×
sc105	Otti-FE	3.1 ms	80,000	372	220×	3.9 s	62 ms	63×	64×	110 ms	17 ms	6.6×	16 KB	9.3 KB	1.7×
scagr7	Otti	6.2 ms	229,000	455	500×	900 ms	62 ms	14×	34×	540 ms	17 ms	31×	29 KB	9.3 KB	3.1×
scagr7	Otti-FE	6.2 ms	159,000	455	350×	7.6 s	62 ms	120×	34×	200 ms	17 ms	12×	16 KB	9.3 KB	1.8×
scsd8	Otti	300 ms	3.42e6	5,900	580×	9.9 s	320 ms	31×	4.0×	8.9 s	30 ms	290×	81 KB	13 KB	6.2×
scsd8	Otti-FE	300 ms	3.42e6	5,900	580×	120 s	320 ms	380×	4.0×	4.0 s	30 ms	130×	20 KB	13 KB	1.5×
Softmax	ZKLP-FE	41 μs	1.31e6	7,230	180×	15 s	200 ms	74×	48,000×	1.5 s	49 ms	31×	22 KB	15 KB	1.4×
LayerNorm	ZKLP-FE	230 μs	1.01e7	98,400	100×	120 s	1.3 s	92×	52,000×	12 s	160 ms	73×	24 KB	19 KB	1.3×
GELU	ZKLP-FE	1.7 ms	4.71e7	1.47e6	32×	470 s	19 s	25×	120,000×	49 s	1.6 s	31×	26 KB	22 KB	1.2×
GPT-2 (*seq*=2, *passes*=1)	ZKLP-FE	79 ms	1.85e10	1.63e6	11,000×	*37 h*	50 s	2,700×	1,500×	*4.7 h*	22 s	770×	*860 KB*	1.5 MB	0.56×
GPT-2 (*seq*=32, *passes*=1)	ZKLP-FE	440 ms	2.26e10	4.19e7	540×	*45 h*	750 s	210×	11,000×	*5.7 h*	78 s	260×	*860 KB*	1.6 MB	0.52×
GPT-2 (*seq*=32, *passes*=1)	zkGPT	440 ms	—	4.19e7	—	64 s	750 s	0.085×	11,000×	5.8 s	78 s	0.075×	88 KB	1.6 MB	0.053×
GPT-2 (*seq*=32, *passes*=2)	zkGPT	870 ms	—	4.19e7	—	130 s	1,600 s	0.081×	12,000×	12 s	78 s	0.15×	180 KB	3.2 MB	0.053×
GPT-2 (*seq*=32, *passes*=4)	zkGPT	1.7 s	—	4.19e7	—	260 s	3,200 s	0.081×	12,000×	23 s	78 s	0.30×	350 KB	6.4 MB	0.054×
GPT-2 (*seq*=32, *passes*=8)	zkGPT	3.5 s	—	4.19e7	—	510 s	1.8 h	0.081×	12,000×	46 s	78 s	0.60×	710 KB	13 MB	0.054×
GPT-2 (*seq*=32, *passes*=16)	zkGPT	7.0 s	—	4.19e7	—	1,000 s	3.3 h	0.086×	11,000×	93 s	79 s	1.2×	1.4 MB	26 MB	0.054×
Fluid Sim. (small)	ZKLP-FE	3.3 ms	2.32e7	291,000	80×	240 s	4.4 s	54×	5,500×	25 s	440 ms	58×	25 KB	21 KB	1.2×
Fluid Sim. (large)	ZKLP-FE	13 ms	1.26e8	1.53e6	83×	920 s	17 s	53×	4,800×	140 s	1.9 s	71×	27 KB	23 KB	1.2×
Geoloc. (1 Location)	ZKLP	790 ns	25,500	596	43×	660 ms	76 ms	8.7×	96,000×	8.6 ms	36 ms	0.24×	0.38 KB	13 KB	0.030×
Geoloc. (512 Locations)	ZKLP	410 μs	1.02e7	596	17,000×	220 s	4.8 s	46×	12,000×	9.7 ms	50 ms	0.19×	0.38 KB	20 KB	0.019×

Figure 4: Constraint counts, native execution time, prover time, verifier time, proof sizes, and overhead of proving (ratio of prover:native) for baselines and benchmarks. "Native time" refers to the verifier's hardware while the Spain/native ratio is measured on the prover's hardware. Times, proof sizes, and ratios are displayed with two significant figures. Constraints are displayed with three significant figures. The italicized numbers for ZKLP-FE mean that the times and proof sizes are extrapolated (see text); the back-end could not run at that scale. zkGPT does not support a batched configuration, so we scale zkGPT's measurements by *passes* where appropriate. Spain improves over baselines in constraint counts (from 32× to 4 orders of magnitude). With two exceptions, Spain improves over baselines in prover and verifier time, by at least an order of magnitude. The first exception is zkGPT, which outperforms Spain on all metrics, except for verifier time when *passes* = 16, where Spain's verifier is slightly better. The second exception is ZKLP, whose verifier outperforms Spain's verifier by 4–5×.

stems from substantial reduction in the number of auxiliary variables in the arithmetizations, making the witness w correspondingly smaller. Meanwhile, recall that one contribution to the verifier's costs is a component logarithmic in $|w|$ (§2.2).

7.2 Costs of Spain

We examine the relative costs of the protocol phases, for the prover and verifier. Figures 5 and 6 depict the results. For most benchmarks, DARK is the dominant cost. For GPT-2, witness generation (compute w) has a larger share of prover time, due almost entirely to the difference in complexity between executing matrix multiplication (which must happen to generate the witness) and checking matrix multiplication with the Freivalds algorithm (which is what the constraints enforce, as described in the GPT-2 benchmark earlier).

The prover's primary resource bottleneck is memory. For example, with GPT-2, our largest benchmark, the prover requires ≈41 GB for *seq*=32, *passes*=1, and ≈267 GB for *seq*=32, *passes*=16. The verifier's bottleneck for small computations (small m) is DARK evaluation. The bottleneck for larger computations is evaluating $\widetilde{A}$, $\widetilde{B}$, and $\widetilde{C}$ at a random point.

7.3 Breaking even and batching

Recall that to justify outsourcing on performance grounds, the verifier must be cheaper than native execution (§1). For the largest linear program, scsd8, Figure 4 shows that Spain's verifier *does* meet this condition. However, for most of the other benchmarks, this condition is not met.

In these cases, Spain must amortize fixed costs (§2.2) via *batching*. This is the SIMD-R1CS model (§4.3, Appx. D.1): the same computation repeated on different input/output pairs. This structure is natural for many numerical computations. For example, when running an LLM, people don't just want one token; they want a whole paragraph (or more...).

Spain's verifier indeed benefits from batching; notice that for the rows in Figure 4 featuring GPT-2 with various *passes*, the verifier's work grows less than linearly in batch size. Indeed, Figure 6 shows that the fixed costs – namely fixed work in DARK (§4.2.3, Appx. C) and evaluating $\widetilde{A}$, $\widetilde{B}$, $\widetilde{C}$ at a random point (Step 6, Figure 2) – are the dominant contribution when verifying single instances, and thus amortization is beneficial. These results agree with the theory. Specifically, Spain's fixed-cost is $O(m)$ for the entire batch (§2.2). Meanwhile, Spain's variable costs comprise a component logarithmic in the total witness size across all instances in the batch and a component linear in the number of variables in *in* and *out* across all instances. Given this cost profile, there is usually a batch size for which, in principle, Spain's verifier breaks even once the batch size crosses a threshold. We say *usually* because, for very small computations with very large *in* and/or *out*, the

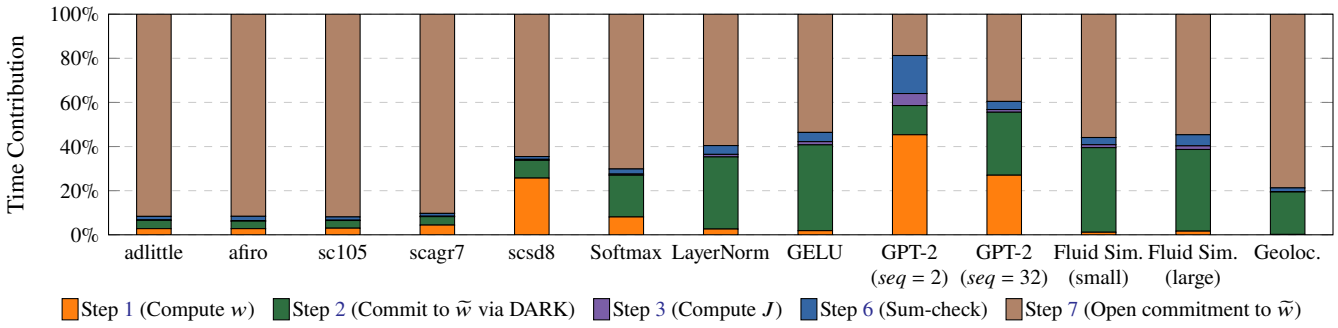


Figure 5: Prover phase breakdown for Spain. Steps refer to those in Figure 2. Steps 4 and 5 are omitted as they incur no cost for the prover.

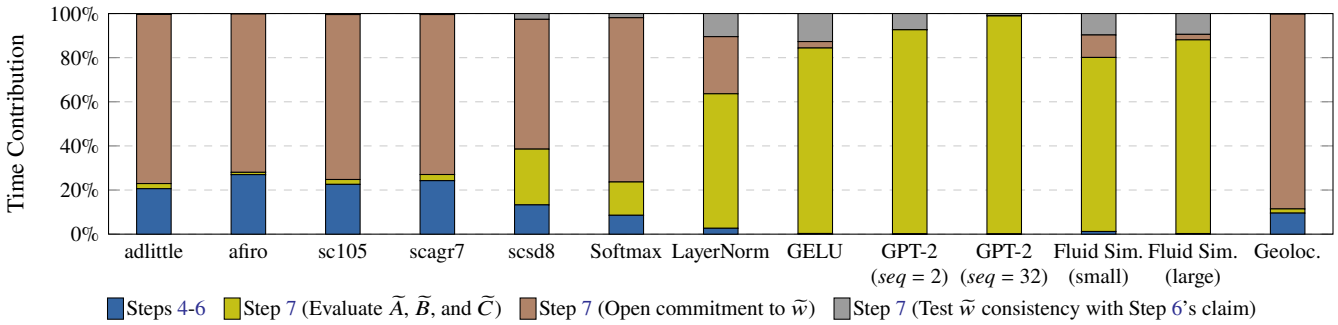


Figure 6: Verifier phase breakdown for Spain. Steps refer to those in Figure 2. Steps 1-3 are omitted as they incur no cost for the verifier. The majority of the verifier's time is spent in Step 7, which decomposes into three sub-steps: evaluating $\widetilde{A}$, $\widetilde{B}$, $\widetilde{C}$; opening the commitment to $\widetilde{w}$; and checking consistency between the claim in Step 6 about $\widetilde{w}$ and the opening of $\widetilde{w}$, using *in* and *out*.

variable costs alone may exceed the cost of native execution.

In contrast to Spain, zkGPT does not amortize with *passes*; as shown in Figure 4 (the GPT-2 rows), zkGPT's verifier is more expensive than Spain's verifier at *seq*=32, *passes*=16. For ZKLP, the comparison is equivocal. On the one hand, ZKLP's fixed costs (which are not included in Figure 4) amortize over all future uses of an R1CS structure, not merely a synchronized batch as in Spain; on the other hand, ZKLP's fixed costs are dramatically higher (over 50 thousand times larger than the verifier's work on a single instance [112]).

A loose end is *setup costs*, as distinct from fixed costs (§2.2); these are not depicted. However, on our benchmarks, Spain's setup costs range from negligible to 5× the verifier's work to check a single instance, so this cost quickly amortizes.

7.4 Summary and discussion

We believe that Spain meets the goal of being general-purpose (§1): it easily handled an array of natural benchmarks, and we did not have to discard any benchmarks because Spain lacked sufficient expressiveness.

At its core, Spain bakes approximation into the constraints themselves, eliminating the need to enforce numerical semantics explicitly, which in turn improves on the number of constraints (m) in an arithmetization, by 32× to 17,000× (§7.1).

This reduction lowers the prover overhead versus natural baselines by 1–3 orders of magnitude, down to 3–5 orders of magnitude versus native execution (§7.1). Spain's prover isn't the fastest in the literature; that honor belongs to zkGPT [105], but as we have noted, zkGPT is specialized to a specific machine learning model architecture, and its verifier neither breaks even nor benefits from amortization (§7.3).

Spain's verifier beats native execution for some reasonably-sized numerical problems (§7.3), and is the only system we experimented with to do so. Given their cost profiles (§7.2), Spain's verifier and prover would directly benefit from improvements in integer polynomial commitment protocols or rational ones [6, 121]. There are additional avenues for improving the verifier's costs. Most saliently, the verifier could shift some work to the prover using SPARK [116] or one of its descendants [38, 117, 119].

A structural disadvantage of Spain is that, for the time being, its new arithmetizations *require* its back-end. Traditional constraints are more portable; they can be combined with back-ends that provide different properties. Otti's back-end, for example, provides zero-knowledge and non-interactivity. ZKLP's back-end does too, and it inherits from Groth16 [70] an extremely fast verifier (albeit with high fixed costs; §7.3). If Spain's arithmetizations could be ported to alternative back-ends (§9), Spain could derive end-to-end speedups and enhanced properties.

8 Other related work

We have described Spain's intellectual antecedents throughout. Here we focus on the intersection of proof systems and numerical computations; we divide the work into two strands.

General-purpose (§1) proof systems. The first attempt to represent numerical operations in succinct proofs was Ginger [115], which supported fixed-point and primitive floating-point operations, by mapping them to operations in a finite field $\mathbb{F}_q$. Ginger used static analysis, bit-wise decompositions of values, and a large field (large q) to ensure a unique representation of each rational encountered during execution.

Subsequent works applied variations on this encoding to specific numerical problems, for example Otti [9] to linear programs, zkQMC [50] to Monte Carlo simulations, and KL24 [82] to transcendental functions. All of these works target general-purpose back-ends [8, 21–24, 46, 47, 49, 58, 62, 70, 85, 90, 100, 116, 117] and express computations as traditional constraints over a finite field.

A few recent works have explored alternative approaches, focusing on arithmetizing floating-point arithmetic. The works in this vein range between strict IEEE 754 compliance with support for all rounding modes and exceptions [51] (see §7) and more relaxed semantics [59]. All such works translate numerical operations using more constraints than Spain; however, they offer relative error guarantees that Spain does not.

Recently, interest has renewed in a class of proof systems known as *zkVMs* [15, 25, 26, 53, 83, 145]. These proof systems allow for proving the correct execution of any programs written in a given instruction set, typically RISC-V [107]. Although RISC-V includes numerical operations via floating-point instructions in an extension, we do not know of any zkVMs that currently support this extension.

Like Spain, some works have back-ends that work over domains other than finite fields. Chen et al. [45] arithmetize numerical computations as arithmetic circuits (addition and multiplication gates) over Galois rings, by encoding a rounding operation as a sequence of additions and multiplications within the ring. They then run GKR [66, 67] (§2.2) over the Galois ring. Bitan et al. [27] observe that some sum-check protocols can be modified so that the protocol messages themselves become approximate. Although we have yet to do a detailed analysis, we estimate that the costs of these approaches would be in between Spain's and the approaches derived from Ginger.

Finally, concurrent with this work, others have refined Zaratan [42] (discussed in §4.2), which is geared to integer computations. These works design polynomial commitment protocols over rational numbers that are significantly faster than DARK [6, 121].[2] While these works apply their faster polynomial commitments to general-purpose proof systems for exact integer and rational arithmetic, rather than approximate computations, we observe that the polynomial commitments themselves are compatible with Spain. We anticipate that replacing DARK with the fastest of them will improve the most expensive components of Spain by at least an order of magnitude (§7.2).

Special-purpose proof systems. Here we consider systems designed around specific applications. These systems rely on the observation that checkers for some numerical computations can be directly represented as sum-check protocols (§2.2).

The first in this line of work is Thaler's protocol for proving matrix multiplication [129]. Some works use Thaler's protocol as a primitive in proof systems for machine learning [19, 63, 77, 78, 86, 105, 127, 128, 151] (see the survey of Peng et al. [101] for a taxonomy). These protocols typically quantize numerical values, and alternate between checking exact operations (such as matrix multiplications, convolutions, and activation functions) and rounding to return to a quantized domain. From our testing, the most efficient of these systems is zkGPT [105], which was discussed and evaluated (§7).

9 Conclusion

Spain meets the goal of proving numerical computations in a general-purpose way, while in some regimes meeting the goals (§1) of a prover with 3 orders of magnitude overhead and a verifier less expensive than native (§7.4).

Spain also introduces a new form of arithmetization, approximate constraints (§5). We believe this is of independent interest, as efficient arithmetizations are an object of intense study among those interested in succinct proofs. In principle, approximate constraints are usable in conjunction with the two dominant forms of arithmetization: traditional constraints and lookup tables (§E.4, §2.2).

A natural area for future work is adapting other proof back-ends (both those based on sum-check protocols and those *not* based on sum-check protocols) to handle approximate constraints. Other future work includes reducing the cost of polynomial commitment in Spain, slashing the cost of Spain's verifier by shifting work to the prover (§7.4), making Spain zero-knowledge and non-interactive, supporting relative error in constraints, and adding numerical support to zkVMs (§8) [15, 53, 83, 145]. Many of these are separate research efforts, which we hope our work here has motivated.

Acknowledgments

We are grateful to our shepherd, Andi Quinn, who went above and beyond with close readings that fundamentally improved the presentation of this work. In a similar vein, this paper was substantially improved by the thoughtful suggestions of the anonymous OSDI reviewers. Andrew Blumberg gave useful comments. All errors and problems are the fault of the authors. This project used computing resources at the Courant Institute and NYU's HPC organization, and was partially supported by gifts from Google and Stellar.

[2] An older work in this family, Zinc [60], relies on results from another paper [28]. However, under scrutiny, it turned out that the latter paper's performance claims are about a protocol without any provable guarantees.

References

[1] Carmichael function. https://en.wikipedia.org/wiki/Carmichael_function. Wikipedia, Wikimedia foundation.

[2] IEEE standard for floating-point arithmetic. *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pages 1–84, 2019.

[3] FLINT: Fast Library for Number Theory. http://flintlib.org, 2025.

[4] GNU MP: The GNU Multiple Precision Arithmetic Library. https://gmplib.org, 2025.

[5] MPFR: The GNU Multiple Precision Floating-Point Reliable Library. https://www.mpfr.org, 2025.

[6] Alexander Abdugafarov, Albert Garreta, Amit Kumar, Michał Osadnik, Psi Vesely, Ilia Vlasov, and Kai Zhe Zheng. Zinc+: SNARKs for polynomial rings. Cryptology ePrint Archive, Paper 2026/855, 2026.

[7] Miguel Ambrona, Anne-Laure Schmitt, Raphael R. Toledo, and Danny Willems. New optimization techniques for PlonK's arithmetization. Cryptology ePrint Archive, Paper 2022/462, 2022.

[8] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *ACM Conference on Computer and Communications Security (CCS)*, 2017.

[9] Sebastian Angel, Andrew J. Blumberg, Eleftherios Ioannidis, and Jess Woods. Efficient representation of numerical optimization problems for SNARKs. In *USENIX Security*, 2022.

[10] Sebastian Angel, Andrew J. Blumberg, Eleftherios Ioannidis, and Jess Woods. Otti: A zkSNARK compiler, solver, prover and verifier for optimization problems. https://github.com/eniac/otti, 2022.

[11] arkworks contributors. arkworks zkSNARK ecosystem. https://arkworks.rs, 2022.

[12] Sanjeev Arora and Boaz Barak. *Computational Complexity: A modern approach*. Cambridge University Press, 2009.

[13] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM*, 45(3):501–555, May 1998.

[14] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: a new characterization of NP. *Journal of the ACM*, 45(1):70–122, January 1998.

[15] Arasu Arun, Srinath Setty, and Justin Thaler. Jolt: SNARKs for virtual machines via lookups. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2024.

[16] László Babai. Trading group theory for randomness. In *ACM Symposium on the Theory of Computing (STOC)*, May 1985.

[17] László Babai, Lance Fortnow, Leonid A Levin, and Mario Szegedy. Checking Computations in Polylogarithmic Time. In *ACM STOC*, 1991.

[18] László Babai, Lance Fortnow, and Carsten Lund. Nondeterministic exponential time has two-prover interactive protocols. *computational complexity*, 1(1):3–40, Mar 1991.

[19] David Balbás, Dario Fiore, Maria Vasco, Damien Robissout, and Claudio Soriente. Modular sumcheck proofs with applications to machine learning and image processing. In *ACM Conference on Computer and Communications Security (CCS)*, 2023.

[20] Anubhav Baweja, Pratyush Mishra, Tushar Mopuri, Karan Newatia, and Steve Wang. Scribe: Low-memory SNARKs via read-write streaming. Cryptology ePrint Archive, Paper 2024/1970, 2024.

[21] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza. Scalable zero knowledge via cycles of elliptic curves. In *IACR International Cryptology Conference (CRYPTO)*, August 2014.

[22] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast Reed-Solomon Interactive Oracle Proofs of Proximity. In *ICALP*, 2018.

[23] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Report 2018/046, 2018.

[24] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In *IACR International Cryptology Conference (CRYPTO)*, August 2013.

[25] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. TinyRAM architecture specification, v0.991, 2013.

[26] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von Neumann architecture. In *USENIX Security*, 2014.

[27] Dor Bitan, Zachary DeStefano, Shafi Goldwasser, Yuval Ishai, Yael Tauman Kalai, and Justin Thaler. Sum-check protocol for approximate computations. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2026.

[28] Alexander R. Block, Zhiyong Fang, Jonathan Katz, Justin Thaler, Hendrik Waldner, and Yupeng Zhang. Field-Agnostic SNARKs from Expand-Accumulate Codes. In *IACR International Cryptology Conference (CRYPTO)*, 2024.

[29] Alexander R. Block, Justin Holmgren, Alon Rosen, Ron D. Rothblum, and Pratik Soni. Time- and Space-Efficient Arguments from Groups of Unknown Order. In *IACR International Cryptology Conference*

(CRYPTO), 2021.

[30] Andrew J. Blumberg, Justin Thaler, Victor Vu, and Michael Walfish. Verifiable computation using multiple provers. Cryptology ePrint Archive, Paper 2014/846, 2014.

[31] Jonathan Bootle, Andrea Cerulli, Jens Groth, Sune Jakobsen, and Mary Maller. Arya: Nearly Linear-Time Zero-Knowledge Proofs for Correct Program Execution. In *ASIACRYPT*, 2018.

[32] Gautam Botrel, Thomas Piellard, Youssef El Housni, Ivo Kubjas, and Arya Tabaie. Consensys/gnark: v0.11.0, September 2024.

[33] Sean Bowe, Jack Grigg, and Daira Hopwood. Recursive proof composition without a trusted setup. Cryptology ePrint Archive, Paper 2019/1021, 2019.

[34] G. Brassard, D. Chaum, and C. Crépeau. Minimum disclosure proofs of knowledge. *Journal of Computer and System Sciences*, 37(2):156–189, October 1988.

[35] B. Braun, A. J. Feldman, Z. Ren, S. Setty, A. J. Blumberg, and M. Walfish. Verifying computations with state. In *ACM Symposium on Operating Systems Principles (SOSP)*, November 2013.

[36] Benjamin Braun. Compiling computations to constraints for verified computation. UT Austin Honors thesis HR-12-10, December 2012.

[37] Isaac Brodsky. H3: Uber's hexagonal hierarchical spatial index. Uber Blog, June 2018.

[38] Benedikt Bünz, Jessica Chen, Zachary DeStefano, and Binyi Chen. Almost linear-time permutation check. Cryptology ePrint Archive, Paper 2025/1850, 2025.

[39] Benedikt Bünz and Ben Fisch. Multilinear Schwartz-Zippel mod N and lattice-based succinct arguments. In *Theory of Cryptography Conference (TCC)*, 2023.

[40] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent SNARKs from DARK compilers. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2020.

[41] Matteo Campanelli, Dario Fiore, and Rosario Gennaro. Natively compatible super-efficient lookup arguments and how to apply them: Natively compatible super-efficient lookup arguments. *J. Cryptol.*, 38(1), January 2025.

[42] Matteo Campanelli and Mathias Hall-Andersen. Fully-succinct arguments over the integers from first principles. Cryptology ePrint Archive, Paper 2024/1548, 2024.

[43] R. D. Carmichael. Note on a new number theory function. *Bulletin of the American Mathematical Society*, 16(5):232–238, 1910.

[44] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. HyperPlonk: Plonk with Linear-Time Prover and High-Degree Custom Gates. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2023.

[45] Shuo Chen, Jung Hee Cheon, Dongwoo Kim, and Daejun Park. Interactive proofs for rounding arithmetic. *IEEE Access*, 10, 2022.

[46] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In *IACR International Cryptology Conference (CRYPTO)*, 2020.

[47] Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. Fractal: Post-quantum and transparent recursive proofs from holography. In *IACR International Cryptology Conference (CRYPTO)*, 2020.

[48] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Practical verified computation with streaming interactive proofs. In *Innovations in Theoretical Computer Science (ITCS)*, pages 90–112, January 2012.

[49] Craig Costello, Cédric Fournet, Jon Howell, Markulf Kohlweiss, Benjamin Kreuter, Michael Naehrig, Bryan Parno, and Samee Zahur. Geppetto: Versatile verifiable computation. In *IEEE Symposium on Security and Privacy*, May 2015.

[50] Zachary DeStefano, Dani Barrack, and Michael Dixon. zkQMC: Zero-knowledge proofs for (some) probabilistic computations using quasi-randomness. Cryptology ePrint Archive, Paper 2022/1007, 2022.

[51] Jens Ernstberger, Chengru Zhang, Luca Ciprian, Philipp Jovanovic, and Sebastian Steinhorst. Zero-knowledge location privacy via accurate floating-point SNARKs. In *IEEE Symposium on Security and Privacy*, 2025.

[52] Jens Ernstberger, Chengru Zhang, Luca Ciprian, Philipp Jovanovic, and Sebastian Steinhorst. Zero-knowledge location privacy via accurate floating-point SNARKs. https://github.com/tumberger/zk-Location, 2025.

[53] Ethereum Foundation. Ethereum foundation zkVM list. https://ethproofs.org/zkvms, 2025.

[54] Filecoin Foundation. Filecoin: A decentralized, efficient, and robust foundation for humanity's information. https://filecoin.io.

[55] Mina Foundation. Mina protocol. https://minaprotocol.com, 2026.

[56] R. Freivalds. Probabilistic machines can use less running time. In *Proceedings of the IFIP Congress*, pages 839–842, 1977.

[57] Ariel Gabizon and Zachary J. Williamson. plookup: A simplified polynomial protocol for lookup tables. Cryptology ePrint Archive, Report 2020/315, 2020.

[58] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge. Cryptology ePrint Archive, Report 2019/953, 2019.

[59] Sanjam Garg, Abhishek Jain, Zhengzhong Jin, and Yinuo Zhang. Succinct zero knowledge for floating

point computations. In *ACM Conference on Computer and Communications Security (CCS)*, 2022.

[60] Albert Garreta, Hendrik Waldner, Katerina Hristova, and Luca Dall'Ava. Zinc: Succinct arguments with small arithmetization overheads from IOPs of proximity to the integers. Cryptology ePrint Archive, Paper 2025/316, 2025.

[61] David Gay. netlib. https://portal.ampl.com/~dmg/netlib/lp/data/, 2013.

[62] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, May 2013.

[63] Zahra Ghodsi, Tianyu Gu, and Siddharth Garg. SafetyNets: verifiable execution of deep neural networks on an untrusted cloud. International Conference on Neural Information Processing Systems (NIPS), 2017.

[64] Oded Goldreich. Probabilistic proof systems – a primer. *Foundations and Trends in Theoretical Computer Science*, 3(1), 2008.

[65] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

[66] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: Interactive proofs for muggles. In *ACM Symposium on the Theory of Computing (STOC)*, May 2008.

[67] Shafi Goldwasser, Yael Tauman Kalai, and Guy N Rothblum. Delegating computation: interactive proofs for muggles. *J. ACM*, 62(4), 2015.

[68] Alexander Golovnev, Jonathan Lee, Srinath Setty, Justin Thaler, and Riad S. Wahby. Brakedown: Linear-time and field-agnostic SNARKs for R1CS. In *CRYPTO*, 2023.

[69] Google. Longfellow ZK. https://github.com/google/longfellow-zk.

[70] Jens Groth. On the size of pairing-based non-interactive arguments. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2016.

[71] Ulrich Habock. Multivariate lookups based on logarithmic derivatives. Cryptology ePrint Archive, Paper 2022/1530, 2022.

[72] Daira-Emma Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. Zcash protocol specification. https://zips.z.cash/protocol/protocol.pdf, 2025.

[73] StarkWare Industries. Starkware. https://starkware.co.

[74] Hao Ji, Michael Mascagni, and Yaohang Li. Gaussian variant of freivalds' algorithm for efficient and reliable matrix product verification. *CoRR*, abs/1705.10449, 2017.

[75] Kunming Jiang, Fraser Brown, and Riad S. Wahby. CoBBL: Dynamic Constraint Generation for SNARKs. In *IEEE Symposium on Security and Privacy*, 2025.

[76] Kunming Jiang, Devora Chait-Roth, Zachary DeStefano, Michael Walfish, and Thomas Wies. Less is more: refinement proofs for probabilistic proofs. In *IEEE Symposium on Security and Privacy*, 2023.

[77] Chanyang Ju, Hyeonbum Lee, Heewon Chung, Jae Hong Seo, and Sungwook Kim. Efficient sum-check protocol for convolution. *IEEE Access*, 9:164047–164059, 2021.

[78] Daniel Kang, Tri Dao, and Matei Zaharia. ZKML: An optimizing system for ML inference in zero-knowledge proofs. *Proceedings of the ACM SIGMOD Conference*, 2024.

[79] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *ASIACRYPT*, 2010.

[80] J. Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *ACM Symposium on the Theory of Computing (STOC)*, pages 723–732, May 1992.

[81] Ahmed Kosba, Charalampos Papamanthou, and Elaine Shi. xJsnark: a framework for efficient verifiable computation. In *IEEE Symposium on Security and Privacy*, 2018.

[82] Kaarel August Kurik and Peeter Laud. Novel approximations of elementary functions in zero-knowledge proofs. Cryptology ePrint Archive, Paper 2024/859, 2024.

[83] Succinct Labs. SP1 zkVM. https://github.com/succinctlabs/sp1, 2025.

[84] Ryan Lavin, Xuekai Liu, Hardhik Mohanty, Logan Norman, Giovanni Zaarour, and Bhaskar Krishnamachari. A survey on the applications of zero-knowledge proofs, 2026.

[85] Jonathan Lee. Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. In *IACR Theory of Cryptography Conference (TCC)*, 2021.

[86] Seunghwa Lee, Hankyung Ko, Jihye Kim, and Hyunok Oh. vCNN: Verifiable convolutional neural network based on zk-SNARKs. *IEEE Transactions on Dependable and Secure Computing*, 21(4):4254–4270, 2024.

[87] Tianyi Liu, Tiancheng Xie, Jiaheng Zhang, Dawn Song, and Yupeng Zhang. Pianist: Scalable zkRollups via fully distributed zero-knowledge proofs. In *IEEE Symposium on Security and Privacy*, pages 1777–1793, Los Alamitos, CA, USA, May 2024. IEEE Computer Society.

[88] Loopring Foundation. Loopring: Ethereum's First zkRollup Layer 2. https://loopring.org.

[89] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *Journal of the ACM*, 39(4):859–868, 1992.

[90] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In *ACM Conference on Computer and Communications Security (CCS)*, 2019.

[91] Silvio Micali. Computationally sound proofs. *SIAM Journal on Computing*, 30(4):1253–1298, 2000.

[92] Microsoft. Spartan: High-speed zero-knowledge SNARKs without trusted setup. https://github.com/microsoft/Spartan, 2025.

[93] Vineet Nair, Justin Thaler, and Michael Zhu. Proving CPU executions in small space. Cryptology ePrint Archive, Paper 2025/611, 2025.

[94] Ihyun Nam. A survey of multivariate polynomial commitment schemes. arXiv:2306.11383, 2023.

[95] ONNX Contributors. Open Neural Network Exchange Intermediate Representation (ONNX IR) Specification. https://github.com/onnx/onnx/blob/main/docs/IR.md, 2026.

[96] Alex Ozdemir, Fraser Brown, and Riad Wahby. CirC: Compiler infrastructure for proof systems, software verification, and more. In *IEEE Symposium on Security and Privacy*, 2022.

[97] Alex Ozdemir, Evan Laufer, and Dan Boneh. Volatile and persistent memory for zksnarks via algebraic interactive proofs. In *IEEE Symposium on Security and Privacy*, 2025.

[98] H. Padé. Sur la représentation approchée d'une fonction par des fractions rationnelles. *Annales scientifiques de l'École Normale Supérieure*, 3e série, 9:3–93, 1892.

[99] Christodoulos Pappas and Dimitrios Papadopoulos. HOBBIT: space-efficient zkSNARK with optimal prover time. In *USENIX Security*, 2025.

[100] Bryan Parno, Craig Gentry, Jon Howell, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *IEEE Symposium on Security and Privacy*, May 2013.

[101] Zhizhi Peng, Chonghe Zhao, Taotao Wang, Guofu Liao, Zibin Lin, Yifeng Liu, Bin Cao, Long Shi, Qing Yang, and Shengli Zhang. A survey of zero-knowledge proof based verifiable machine learning. *Artificial Intelligence Review*, Apr 2026.

[102] Pepper Project. Pequin: An end-to-end toolchain for verifiable computation, SNARKs, and probabilistic proofs. https://github.com/pepper-project/pequin, 2018.

[103] Polygon Labs UI. Polygon. https://polygon.technology.

[104] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 2 edition, October 1992.

[105] Wenjie Qu, Yijun Sun, Xuanming Liu, Tao Lu, Yanpei Guo, Kai Chen, and Jiaheng Zhang. zkGPT: an efficient non-interactive zero-knowledge proof framework for LLM inference. In *USENIX Security*, 2025.

[106] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf, 2019.

[107] RISC-V International. RISC-V: Ratified Specification. https://riscv.org/specifications/ratified/, 2025.

[108] Michael Rosenberg, Tushar Mopuri, Hossein Hafezi, Ian Miers, and Pratyush Mishra. Hekaton: Horizontally-scalable zkSNARKs via proof aggregation. In *ACM Conference on Computer and Communications Security (CCS)*, 2024.

[109] J. Barkley Rosser and Lowell Schoenfeld. Approximate formulas for some functions of prime numbers. *Illinois Journal of Mathematics*, 6:64–94, 1962.

[110] Kevin Sahr, Denis White, and Jon A. Kimerling. Geodesic discrete global grid systems. *Cartography and Geographic Information Science*, 30:121–134, 2003.

[111] J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, October 1980.

[112] SCIPR Lab. libsnark ppzkSNARK empirical performance README. https://github.com/scipr-lab/libsnark/blob/master/libsnark/zk_proof_systems/ppzksnark/README.md, 2017. GitHub repository documentation.

[113] SCIPR Lab and contributors. libsnark: a C++ library for zksnark proofs. https://github.com/scipr-lab/libsnark, 2026.

[114] S. Setty, B. Braun, V. Vu, A. J. Blumberg, B. Parno, and M. Walfish. Resolving the conflict between generality and plausibility in verified computation. In *European Conference on Computer Systems (EuroSys)*, pages 71–84, April 2013.

[115] S. Setty, V. Vu, N. Panpalia, B. Braun, A. J. Blumberg, and M. Walfish. Taking proof-based verified computation a few steps closer to practicality. In *USENIX Security*, August 2012.

[116] Srinath Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In *CRYPTO*, 2020.

[117] Srinath Setty and Jonathan Lee. Quarks: Quadruple-efficient transparent zkSNARKs. Cryptology ePrint Archive, Paper 2020/1275, 2020.

[118] Srinath Setty, Justin Thaler, and Riad Wahby. Customizable constraint systems for succinct arguments. Cryptology ePrint Archive, Paper 2023/552, 2023.

[119] Srinath Setty, Justin Thaler, and Riad Wahby. Unlocking the Lookup Singularity with Lasso. In *Annual International Conference on the Theory and Applications of*

Cryptographic Techniques (EUROCRYPT), 2024.

[120] Adi Shamir. IP = PSPACE. *Journal of the ACM*, 39(4):869–877, October 1992.

[121] Alireza Shirzad, Sriram Sridhar, Dimitrios Papadopoulos, and Charalampos Papamanthou. Relaxed modular PCS from arbitrary PCS and applications to SNARKs for integers. Cryptology ePrint Archive, Paper 2026/347, 2026.

[122] Michael Sipser. *Introduction to the Theory of Computation*. Cengage Learning, 3 edition, 2013.

[123] Eduardo Soria-Vazquez. Doubly efficient interactive proofs over infinite and non-commutative rings. In *IACR Theory of Cryptography Conference (TCC)*, 2022.

[124] Sriram Sridhar, Shravan Srinivasan, Dimitrios Papadopoulos, and Charalampos Papamanthou. Efficiently provable approximations for non-polynomial functions. In *USENIX Security*, 2026.

[125] Jos Stam. Stable fluids. In *SIGGRAPH*, 1999.

[126] Alan Stapelberg. Opening up 'Zero-Knowledge Proof' technology to promote privacy in age assurance. https://blog.google/technology/safety-security/opening-up-zero-knowledge-proof-technology-to-promote-privacy-in-age-assurance/, July 2025.

[127] Haochen Sun, Tonghe Bai, Jason Li, and Hongyang Zhang. zkDL: Efficient zero-knowledge proofs of deep learning training. *IEEE Transactions on Information Forensics and Security*, 2024.

[128] Haochen Sun, Jason Li, and Hongyang Zhang. zkLLM: Zero knowledge proofs for large language models. In *ACM Conference on Computer and Communications Security (CCS)*, 2024.

[129] Justin Thaler. Time-optimal interactive proofs for circuit evaluation. In *IACR International Cryptology Conference (CRYPTO)*, August 2013.

[130] Justin Thaler. Proofs, Arguments, and Zero-Knowledge. http://people.cs.georgetown.edu/jthaler/ProofsArgsAndZK.html, 2023.

[131] Justin Thaler. Sum-check is all you need: An opinionated survey on fast provers in SNARK design. Cryptology ePrint Archive, Paper 2025/2041, 2025.

[132] The Linux Foundation. ONNX: Open Neural Network Exchange. https://onnx.ai, 2019.

[133] Louis Tremblay Thibault, Tom Sarry, and Abdelhakim Senhaji Hafid. Blockchain scaling using rollups: A comprehensive survey. *IEEE Access*, 10:93039–93054, 2022.

[134] Ioanna Tzialla, Abhiram Kothapalli, Bryan Parno, and Srinath Setty. Transparency dictionaries with succinct proofs of correct operation. In *Network and Distributed System Security Symposium (NDSS)*, 2022. https://eprint.iacr.org/2021/1263.

[135] V. Vu, S. Setty, A. J. Blumberg, and M. Walfish. A hybrid architecture for interactive verifiable computation. In *IEEE Symposium on Security and Privacy*, May 2013.

[136] Riad S. Wahby, Max Howald, Siddharth Garg, abhi shelat, and Michael Walfish. Verifiable ASICs. In *IEEE Symposium on Security and Privacy*, 2016.

[137] Riad S. Wahby, Ye Ji, Andrew J. Blumberg, abhi shelat, Justin Thaler, Michael Walfish, and Thomas Wies. Full accounting for verifiable outsourcing. In *ACM Conference on Computer and Communications Security (CCS)*, 2017.

[138] Riad S. Wahby, Srinath Setty, Max Howald, Zuocheng Ren, Andrew J. Blumberg, and Michael Walfish. Efficient RAM and control flow in verifiable outsourced computation. In *Network and Distributed System Security Symposium (NDSS)*, 2015.

[139] Riad S. Wahby, Ioanna Tzialla, abhi shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zkSNARKs without trusted setup. In *IEEE Symposium on Security and Privacy*, 2018.

[140] Michael Walfish and Andrew J. Blumberg. Verifying computations without reexecuting them: from theoretical possibility to near practicality. *Communications of the ACM (CACM)*, 58(2), 2015.

[141] Wikimedia Foundation Wikipedia. Localization (commutative algebra). https://en.wikipedia.org/wiki/Localization_(commutative_algebra).

[142] Howard Wu, Wenting Zheng, Alessandro Chiesa, Raluca Ada Popa, and Ion Stoica. DIZK: a distributed zero knowledge proof system. In *USENIX Security*, 2018.

[143] Tiacheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. Libra: Succinct Zero-Knowledge Proofs with Optimal Prover Computation. In *IACR International Cryptology Conference (CRYPTO)*, 2019.

[144] Tiancheng Xie, Jiaheng Zhang, Zerui Cheng, Fan Zhang, Yupeng Zhang, Yongzheng Jia, Dan Boneh, and Dawn Song. zkBridge: Trustless cross-chain bridges made practical. In *ACM Conference on Computer and Communications Security (CCS)*, 2022.

[145] RISC Zero. RISC0 zkVM. https://github.com/risc0/risc0, 2025.

[146] Collin Zhang, Zachary DeStefano, Arasu Arun, Joseph Bonneau, Paul Grubbs, and Michael Walfish. Zombie: Middleboxes that don't snoop. In *Symposium on Networked Systems Design and Implementation (NSDI)*, 2024.

[147] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. vSQL: Verifying arbitrary SQL queries over dynamic outsourced databases. In *IEEE Symposium on Security and Privacy*, 2017.

[148] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou.

A zero-knowledge version of vSQL. Cryptology ePrint Archive, Paper 2017/1146, 2017.

[149] Richard Zippel. Probabilistic algorithms for sparse polynomials. In Edward W. Ng, editor, *Symbolic and Algebraic Computation*, pages 216–226. Springer Berlin Heidelberg, 1979.

[150] zkcrypto. bellman: zk-SNARK library, 2026.

[151] Zkonduit. Easy Zero-Knowledge Inference. https://github.com/zkonduit/ezkl, 2024.

A Sum-check primitive

Given a prover, $\mathcal{P}$, and a verifier, $\mathcal{V}$, the sum-check primitive of Lund et al. [89] is an interactive protocol for proving that a claimed value H is equal to the sum of evaluations of a v-variate polynomial G over the Boolean hypercube: $\{0,1\}^v$. This primitive is depicted in Figure 7.

Spain relies on the following lemma concerning the soundness and completeness of this primitive [122, Chapter 10.4] [12, Chapter 8.5] [130, Chapter 4.1].

Lemma 1. *The primitive in Figure 7 is an interactive proof (the term is defined in the references above) for the claim that* $H = \sum_{b\in\{0,1\}^v} G(b)$. *If the initial claim is false,* $\mathcal{V}$ *rejects with probability at least* $1 - sd/q$. *If the initial claim is true,* $\mathcal{P}$ *can make* $\mathcal{V}$ *accept with probability* 1.

Setup: $\mathcal{P}$ and $\mathcal{V}$ agree on a field $\mathbb{F}_q$ and a degree d multi-variate polynomial G over $\mathbb{F}_q$ with v variables.
Online phase: If at any point $\mathcal{V}$'s check fails, it rejects.

1. $\mathcal{P}$ sends $\mathcal{V}$ a value H, claimed to equal
$$\sum_{b\in\{0,1\}^v} G(b)$$
and a univariate polynomial $g_1(x)$, claimed to equal
$$\sum_{(b_2,\ldots,b_s)\in\{0,1\}^{v-1}} G(x, b_2, \ldots, b_s).$$
2. $\mathcal{V}$ checks that $H = g_1(0) + g_1(1)$.
3. $\mathcal{V}$ sends $\mathcal{P}$ a random $r_1 \in \mathbb{F}_q$.
4. For $k = 2, \ldots, v$:
   (a) $\mathcal{P}$ sends $\mathcal{V}$ a univariate polynomial $g_k(x)$, claimed to equal
   $$\sum_{(b_{k+1},\ldots,b_s)\in\{0,1\}^{v-k}} G(r_1, \ldots, r_{k-1}, x, b_{k+1}, \ldots, b_s).$$
   (b) $\mathcal{V}$ checks that $g_{k-1}(r_{k-1}) = g_k(0) + g_k(1)$.
   (c) $\mathcal{V}$ sends $\mathcal{P}$ a random $r_k \in \mathbb{F}_q$.
5. Finally, $\mathcal{V}$ checks that $g_s(r_s) = G(r_1, \ldots, r_s)$.

If all checks pass, $\mathcal{V}$ accepts.

Figure 7: The sum-check primitive of Lund et al. [89].

B Spain's full back-end protocol and analysis

Here we provide background on polynomial commitment schemes and define the properties required by Spain (§B.1); provide Spain's full back-end protocol (§B.2) and its correctness proofs (§B.3); quantify the guarantees based on the specific parameters in our implementation and evaluation (§B.4); and finally prove some supporting lemmas (§B.5).

For clarity in exposition, when we mention a polynomial commitment scheme, we are referring to a polynomial commitment scheme for multilinear polynomials. Additionally, when we use the term "coefficients" for a v-variate multilinear polynomial, these refer to the coefficients in the multilinear Lagrange basis [130, §3.5] with the interpolating set being the v-variate Boolean hypercube: $\{0,1\}^v$. These are exactly the *evaluations* of the polynomial over the Boolean hypercube.

B.1 Spain-compatible polynomial commitment schemes

A multilinear polynomial commitment scheme [79, 94] *PC* is a tuple of protocols (*Setup*, *Commit*, *Open*, *Verify*), defined as follows.

- *Setup*: Given security parameter λ and a bound on the number of variables in polynomials to be committed to, produce public parameters *pp*.
- *Commit*: Given public parameters *pp* and a multilinear polynomial $\tilde{f}$, produce a commitment $com_{\tilde{f}}$.
- *Open*: Given public parameters *pp*, a multilinear polynomial $\tilde{f}$, and a point x, produce a result y and a proof π_y that $y = \tilde{f}(x)$. Together, the result and proof are called an *opening* of $com_{\tilde{f}}$ at x.
- *Verify*: Given public parameters *pp*, a commitment $com_{\tilde{f}}$, a point x, and an alleged opening (y, π_y), output "accept" if π_y is a valid proof that $y = \tilde{f}(x)$ and "reject" otherwise.

A polynomial commitment scheme is *binding* if it is intractable for a prover to produce two distinct openings (y, π_y) and $(y', \pi_{y'})$ for the same commitment $com_{\tilde{f}}$ such that the verifier accepts both openings. For Spain, intractable means that a polynomial-time adversary cannot succeed with probability greater than κ_{PC}.

Spain requires, as a building block, a binding polynomial commitment scheme, *PC*, that satisfies an additional property, which we call *Weak Integer Binding*. Weak Integer Binding is parameterized by 2 constants, $N_{\max}$ and $D_{\max}$. Weak Integer Binding has two sub-properties, given below. To state the sub-properties we define *Integer-limited*: a v-variate multilinear polynomial $\tilde{f}$ is called *Integer-limited* if all of its coefficients are integers with magnitude at most $N_{\max}$.

- *Weak Integer Binding, sub-property 1.* It is impossible for a prover to produce a commitment $com_{\tilde{f}}$ unless for some positive integer D' and multilinear polynomial $\tilde{g}$, $\tilde{f} = \tilde{g}/D'$, where $D' \leq D_{\max}$ and $\tilde{g}$ is Integer-limited. (Here $\tilde{g}/D'$

means a polynomial with the same coefficients as $\tilde{g}$ but with an extra D' factor in each coefficient's denominator.)

- *Weak Integer Binding, sub-property 2.* Given an honest (polynomial-time) prover, for any Integer-limited polynomial $\tilde{f}$, invoking *Commit* causes the prover to produce a valid commitment, and for any point x, invoking *Open* causes the prover to produce a corresponding opening that causes *Verify* to accept

Weak Integer Binding is similar to, but weaker than, the notion of a *Relaxed Mod-PCS* introduced by Zinc [60]. A Relaxed Mod-PCS imposes an additional requirement, *extractability* (essentially, that the prover must "know" $\tilde{f}$ of the right form in order to produce $com_{\tilde{f}}$).

Also, note that Weak Integer Binding is an idealized property in that sub-property 1 states that it is *impossible*, rather than computationally *intractable*, for a prover to produce a commitment $com_{\tilde{f}}$ that does not satisfy the stated condition. When instantiating a multilinear commitment primitive (§B.4), we will handle probabilistic deviation from the ideal.

B.2 Spain's back-end protocol

Spain's full protocol is in Figure 8. This protocol is parameterized by a constant ϵ; an R1CS instance, $\mathbb{X}$, containing rationals with unreduced denominator D and unreduced numerators with magnitude at most $N_{\max}$; a polynomial commitment scheme PC satisfying Weak Integer Binding with parameters $N_{\max}$ and $D_{\max}$ (§B.1); and a large set of primes, P. The R1CS structure underlying $\mathbb{X}$ has m constraints and n variables. Recall from Section 4.2.2 that s and k are defined to be $\lceil \log_2 m \rceil$ and $\lceil \log_2 n \rceil$ respectively.

Note that Weak Integer Binding means an honest prover commits to a polynomial $\tilde{f}$ with *integer* coefficients, yet Spain has been described (Fig. 2, §4) as committing to a polynomial $\widetilde{w}$ with *rational* coefficients. In fact, in Step 1 (Fig. 8), Spain's prover obtains $\tilde{f}$ by multiplying $\widetilde{w}$ by D. Meanwhile the elements of w are expected to have unreduced denominator D (§4.2.1) and hence the same holds for the coefficients of $\widetilde{w}$. Thus, $\tilde{f}$ indeed has integer coefficients.

B.3 Correctness of Spain's back-end protocol

In this section, we prove the correctness, meaning completeness and soundness (§4), of Spain's back-end, specifically the protocol in Figure 8. The figure depicts, and the proof is about, an idealized version of Spain in which the polynomial commitment scheme meets the properties in Appendix B.1. In addition to quantifying parameters, Appendix B.4 instantiates the polynomial commitment scheme with a concrete protocol, namely DARK [40], and handles the deviation from the ideal.

En route to the proof, we define $U_{m,n,N_{\max},D,D_{\max}}$ (written as U for brevity) to be

$$U := m \cdot n^2 \cdot N_{\max}^4 \cdot D_{\max}^4 \cdot (n \cdot N_{\max}^2 + D^2)^2 + D^8 \cdot D_{\max}^4.$$

Additionally, we use $\min P$ to denote the smallest prime in P, and $|P|$ to denote the number of primes in P.

The definitions of completeness and soundness (§4) reference ϵ-accuracy, which was described informally in Section 3. Below, we state ϵ-accuracy more formally. Let M_i denote row i of matrix M.

Definition (ϵ-accuracy)**.** *An assignment z for an R1CS structure $\mathbb{S} = (A, B, C)$ is* ϵ-accurate *if*

$$|(A_i \cdot z) \cdot (B_i \cdot z) - (C_i \cdot z)| \leq \epsilon, \text{ for all } i \in \{1, \ldots, m\}.$$

We are now ready to state the central correctness theorem for Spain.

Theorem 1. *The protocol in Figure 8 satisfies Back-end $(\epsilon/\sqrt{m})$-completeness. The protocol also satisfies Back-end ϵ-soundness, with soundness error κ upper-bounded by*

$$\frac{\lfloor \log_{\min P} D_{max} \rfloor + \lfloor \log_{\min P} U \rfloor}{|P|} + \frac{4s + 2 + 2k}{\min P} + \kappa_{PC}.$$

Proof. We use two facts about ρ_q, both proved in Appendix B.5.

Lemma 2 (ρ_q ill-defined)**.** *Let $\mathbb{X}$, z, and $\widetilde{w}$ be as in Figure 8, and let q be sampled uniformly from P. Then*

$$\Pr_{q \sim P}\left[\rho_q \text{ is undefined on } \|E_{\mathbb{X},z}\|_2^2\right] \leq \frac{\lfloor \log_{\min P} D_{max} \rfloor}{|P|}.$$

Lemma 3 (ρ_q collision)**.** *Let $\mathbb{X}$, z, $\widetilde{w}$, and J be as in Figure 8 with $J \neq \|E_{\mathbb{X},z}\|_2^2$, and let q be sampled uniformly from P. Conditioned on ρ_q being defined for $\|E_{\mathbb{X},z}\|_2^2$, we have*

$$\Pr_{q \sim P}\left[\rho_q(J) = \rho_q(\|E_{\mathbb{X},z}\|_2^2)\right] \leq \frac{\lfloor \log_{\min P} U \rfloor}{|P|}.$$

Back-end $(\epsilon/\sqrt{m})$-completeness. If $\mathcal{P}$ has an assignment z that is $(\epsilon/\sqrt{m})$-accurate for the instance $\mathbb{X}$, by the argument of Section 4.1, $\|E_{\mathbb{X},z}\|_2^2 \leq \epsilon^2$. Thus if $\mathcal{P}$ sends messages prescribed by the protocol, every check by $\mathcal{V}$ passes, and $\mathcal{V}$ accepts with probability 1.

Back-end ϵ-soundness. Suppose that there does not exist an ϵ-accurate assignment for $\mathbb{X}$. We bound the probability that $\mathcal{V}$ accepts.

First, an adversarial prover need not use *Commit* in Step 1 to produce $com_{\tilde{f}}$ nor use *Open* in Step 8 to produce an alleged opening (y, π_y) of $com_{\tilde{f}}$ at $\hat{\beta}$. To handle all adversarial prover behavior involving the polynomial commitment, we define $\mathsf{E}_{\mathsf{bind}}$ to capture the event that $\mathcal{V}$ accepts the opening in Step 8 but this opening is inconsistent with the commitment provided in Step 1. By inconsistent, we mean that $y \neq \tilde{f}(\hat{\beta})$. By the binding property of the polynomial commitment scheme,

$$\Pr[\mathsf{E}_{\mathsf{bind}}] \leq \kappa_{\mathrm{PC}}.$$

We now consider the case where $\mathsf{E}_{\mathsf{bind}}$ does not occur, so the binding property of the polynomial commitment scheme fixes a unique $\tilde{f}$ and thus a unique w (and unique z).

Setup: Prover $\mathcal{P}$ and Verifier $\mathcal{V}$ agree on the parameters of the protocol, which are as follows:

- a small constant $\epsilon \ll 1$
- an instance, $\mathbb{X} = (A, B, C, in, out)$, consisting of rationals with unreduced denominator D (a power of 2).
- a polynomial commitment scheme PC satisfying Weak Integer Binding with parameters $N_{\max}$ and $D_{\max}$.
- a large set of primes, P, from which $\mathcal{V}$ will sample in the protocol.

Preprocessing: $\mathcal{P}$ and $\mathcal{V}$ perform the setup required by PC.
Online phase: If at any point $\mathcal{V}$'s check fails, it rejects.

1. Using PC, $\mathcal{P}$ commits to a multilinear polynomial $\tilde{f}$ whose coefficients are purportedly integers. $\mathcal{P}$ sends this commitment to $\mathcal{V}$.
   - Define $\widetilde{w} := \tilde{f}/D$. Define w as the coefficients of $\widetilde{w}$, and define $z := (in, out, 1, w)$. If $\mathcal{P}$ is honest, then w is chosen to make z an ϵ-accurate assignment to $\mathbb{X}$.
2. $\mathcal{P}$ computes $\|E_{\mathbb{X},z}\|_2^2$ (over $\mathbb{Q}$), and sends $S := \|E_{\mathbb{X},z}\|_2^2 \cdot D^8$ to $\mathcal{V}$.
3. $\mathcal{V}$ checks that S is an integer, computes $J := S/D^8$, and checks that $0 \le J \le \epsilon^2$.
4. $\mathcal{V}$ samples a random prime $q \sim P$, sends it to $\mathcal{P}$, and $\mathcal{P}$ and $\mathcal{V}$ map all quantities in $\mathbb{Q}$ into $\mathbb{F}_q$ via ρ_q.
5. $\mathcal{V}$ and $\mathcal{P}$ run a degree-4 sum-check primitive on
$$\rho_q(J) = \sum_{r \in \{0,1\}^s} (\widetilde{g_A}(r) \cdot \widetilde{g_B}(r) - \widetilde{g_C}(r))^2,$$
where $\widetilde{g_A}$, $\widetilde{g_B}$, and $\widetilde{g_C}$ are defined as in Section 4.2.2.
The result of this primitive is three purported claims over $\mathbb{F}_q$:
$$v_a = \widetilde{g_A}(\alpha), \quad v_b = \widetilde{g_B}(\alpha), \quad \text{and} \quad v_c = \widetilde{g_C}(\alpha)$$
for a random $\alpha \in \mathbb{F}_q^s$.
6. $\mathcal{V}$ samples a random $\gamma \in \mathbb{F}_q$, sends γ to $\mathcal{P}$, and computes
$$v \leftarrow v_a + v_b \cdot \gamma + v_c \cdot \gamma^2.$$
7. $\mathcal{P}$ and $\mathcal{V}$ run a degree-2 sum-check primitive on
$$v = \sum_{c \in \{0,1\}^k} (\widetilde{A}(\alpha, c) + \widetilde{B}(\alpha, c) \cdot \gamma + \widetilde{C}(\alpha, c) \cdot \gamma^2) \cdot \widetilde{z}(c).$$
This results in the following four purported claims over $\mathbb{F}_q$:
$$u_a = \widetilde{A}(\alpha, \beta), \quad u_b = \widetilde{B}(\alpha, \beta), \quad u_c = \widetilde{C}(\alpha, \beta), \quad \text{and} \quad u_z = \widetilde{z}(\beta)$$
for a random $\beta \in \mathbb{F}_q^k$.
8. $\mathcal{P}$ maps the point β into $\mathbb{Z}$ as $\hat{\beta}$, opens the commitment to $\tilde{f}$ at $\hat{\beta}$, and sends the result y (and accompanying proof π_y) to $\mathcal{V}$.
9. $\mathcal{V}$ uses π_y to check that the result y is a valid opening; if it is, $y = \tilde{f}(\hat{\beta})$ over the rationals. $\mathcal{V}$ computes $\widetilde{in}(\beta)$ and $\widetilde{out}(\beta)$ (over $\mathbb{F}_q$), and combines them with $y/D = \tilde{f}(\hat{\beta})/D = \widetilde{w}(\hat{\beta})$ (mapped to $\mathbb{F}_q$ via ρ_q) to produce $\tilde{z}(\beta)$. $\mathcal{V}$ then checks the purported claims from the end of Step 7.

If all checks pass, $\mathcal{V}$ accepts.

Figure 8: Spain's back-end protocol for proving that an R1CS instance has an ϵ-accurate assignment.

Because $\mathbb{X}$ has no ϵ-accurate assignment, $\|E_{\mathbb{X},z}\|_\infty > \epsilon$ and thus $\|E_{\mathbb{X},z}\|_2^2 > \epsilon^2$. However, in Step 3, $\mathcal{V}$ accepts only if $J \le \epsilon^2$. Consequently, $J \ne \|E_{\mathbb{X},z}\|_2^2$.

For $\mathcal{V}$ to accept the final check in Step 9, it must be the case that, in some step of the protocol, a false claim is transformed into a true claim. We define one bad event per step that captures this transformation.

- E_{ρ_q}: the event that, in Step 4, $J \ne \|E_{\mathbb{X},z}\|_2^2$, but either ρ_q is undefined on $\|E_{\mathbb{X},z}\|_2^2$ or $\rho_q(J) = \rho_q(\|E_{\mathbb{X},z}\|_2^2)$.
- $\mathsf{E}_{\mathsf{sc1}}$: the event that, in Step 5, $\rho_q(J) \ne \rho_q(\|E_{\mathbb{X},z}\|_2^2)$, but after the first sum-check primitive, the claims $v_a = \widetilde{g_A}(\alpha)$, $v_b = \widetilde{g_B}(\alpha)$, and $v_c = \widetilde{g_C}(\alpha)$ are all true.
- $\mathsf{E}_{\mathsf{agg}}$: the event that, in Step 6, $v_a \ne \widetilde{g_A}(\alpha)$, $v_b \ne \widetilde{g_B}(\alpha)$, or $v_c \ne \widetilde{g_C}(\alpha)$, but the aggregated claim $v = v_a + v_b \cdot \gamma + v_c \cdot \gamma^2$ is true.

- $\mathsf{E}_{\mathsf{sc2}}$: the event that, in Step 7,

$$v \neq \sum_{c \in \{0,1\}^k} (\widetilde{A}(\alpha, c) + \widetilde{B}(\alpha, c) \cdot \gamma + \widetilde{C}(\alpha, c) \cdot \gamma^2) \cdot \widetilde{z}(c),$$

but after the second sum-check primitive, the claims $u_a = \widetilde{A}(\alpha, \beta)$, $u_b = \widetilde{B}(\alpha, \beta)$, $u_c = \widetilde{C}(\alpha, \beta)$, and $u_z = \widetilde{z}(\beta)$ are all true.

If none of these occur, then ρ_q is defined on $\|E_{\mathbb{X},z}\|_2^2$, $\rho_q(J) \neq \rho_q(\|E_{\mathbb{X},z}\|_2^2)$, and each step of the protocol transforms a false claim into a false claim. As a result, in the final step, at least one of the following must be false:

$$u_a = \widetilde{A}(\alpha, \beta)$$
$$u_b = \widetilde{B}(\alpha, \beta)$$
$$u_c = \widetilde{C}(\alpha, \beta)$$
$$u_z = \widetilde{z}(\beta)$$

Either the verifier's explicit evaluation of $\widetilde{A}$, $\widetilde{B}$, or $\widetilde{C}$ at (α, β) will expose a false claim, or the opening of the commitment to $\widetilde{w}$ will expose a false claim about $\widetilde{z}(\beta)$. Hence $\mathcal{V}$ rejects if none of the bad events occur.

By a union bound, the probability that $\mathcal{V}$ accepts is at most the sum of the probabilities of these bad events, plus $\Pr[\mathsf{E}_{\mathsf{bind}}]$. We now bound the probability of each bad event.

For $\mathsf{E}_{\rho_\mathbf{q}}$, by Lemma 2, the probability that ρ_q is undefined on $\|E_{\mathbb{X},z}\|_2^2$ is at most

$$\frac{\lfloor \log_{\min P} D_{\max} \rfloor}{|P|}.$$

By Lemma 3, the probability that $\rho_q(J) = \rho_q(\|E_{\mathbb{X},z}\|_2^2)$, conditioned on $J \neq \|E_{\mathbb{X},z}\|_2^2$ and ρ_q being defined on $\|E_{\mathbb{X},z}\|_2^2$, is at most

$$\frac{\lfloor \log_{\min P} U \rfloor}{|P|}.$$

Thus, by a union bound,

$$\Pr[\mathsf{E}_{\rho_q}] \leq \frac{\lfloor \log_{\min P} D_{\max} \rfloor + \lfloor \log_{\min P} U \rfloor}{|P|}.$$

For $\mathsf{E}_{\mathsf{sc1}}$, this event involves an invocation of the sum-check primitive on a degree-4 polynomial with s variables where the claimed sum is false but the final polynomial evaluation claims are all true. By Lemma 1 and $\min P \leq q$,

$$\Pr[\mathsf{E}_{\mathsf{sc1}}] \leq 4s/q \leq 4s/\min P.$$

For $\mathsf{E}_{\mathsf{agg}}$, this event treats three claims as a degree 2 polynomial and evaluates this polynomial at a random point γ. If at least one constituent claim is false, the resulting polynomial is non-zero, and thus, by the Factor Theorem (a univariate case of the Schwartz-Zippel lemma [111, 149]) and $\min P \leq q$,

$$\Pr[\mathsf{E}_{\mathsf{agg}}] \leq 2/q \leq 2/\min P.$$

For $\mathsf{E}_{\mathsf{sc2}}$, this event involves an invocation of the sum-check primitive on a degree-2 polynomial with k variables where the claimed sum is false but the final polynomial evaluation claims are all true. By Lemma 1 and $\min P \leq q$,

$$\Pr[\mathsf{E}_{\mathsf{sc2}}] \leq 2k/q \leq 2k/\min P.$$

By a union bound over all bad events, we get Back-end ϵ-soundness holds with error at most

$$\frac{\lfloor \log_{\min P} D_{\max} \rfloor + \lfloor \log_{\min P} U \rfloor}{|P|} + \frac{4s + 2 + 2k}{\min P} + \kappa_{\mathrm{PC}}.$$

□

B.4 Quantifying the guarantees of Spain's back-end

The implementation described in Sections 6 and 7 uses specific parameters. The following corollary characterizes the soundness and completeness guarantees of Spain's back-end when instantiated with DARK as the polynomial commitment scheme, and in a regime that covers the aforementioned parameters.

Corollary 1. *When Figure 8 is instantiated with the following parameters:*

- $m \leq 2^{32}$, $n \leq 2^{32}$;
- $D \leq 2^{96}$;
- *PC is the DARK multilinear polynomial commitment scheme [40] with* $N_{max} \leq 2^{96}$; *and*
- P *is the set of* 128*-bit primes,*

the resulting protocol has Back-end $(\epsilon/\sqrt{m})$*-completeness and Back-end* ϵ*-soundness, with soundness error* $\kappa \leq 2^{-40}$.

Proof. Back-end $(\epsilon/\sqrt{m})$*-completeness.* DARK satisfies sub-property 2 of Weak Integer Binding, so the completeness result of Theorem 1 applies directly to this corollary.

Back-end ϵ*-soundness.* For soundness, DARK only satisfies sub-property 1 of Weak Integer Binding *with high probability* (explicitly bounded further below), so we have to modify the soundness analysis of Theorem 1.

Suppose that there does not exist an ϵ-accurate assignment for $\mathbb{X}$. We bound the probability that $\mathcal{V}$ accepts. Define the event $\mathsf{E}_{\mathsf{weak}}$ to be the event that, over randomness internal to DARK's *Verify* function, $\mathcal{V}$ accepts an alleged opening of a commitment $com_{\tilde{f}}$ at $\hat{\beta}$ where $\tilde{f}$ does not satisfy the conditions detailed in sub-property 1 of Weak Integer Binding.

Conditioned on $\mathsf{E}_{\mathsf{weak}}$ not occurring, the verifier's acceptance probability is upper-bounded by the soundness error stated in Theorem 1. Thus, the overall probability that $\mathcal{V}$ accepts is at most

$$\frac{\lfloor \log_{\min P} D_{\max} \rfloor + \lfloor \log_{\min P} U \rfloor}{|P|} + \frac{4s + 2 + 2k}{\min P} + \kappa_{\mathrm{PC}} + \Pr[\mathsf{E}_{\mathsf{weak}}].$$

It remains to fill in the parameters and bound $\Pr[\mathsf{E}_{\mathsf{weak}}]$.

First, we quantify DARK-related parameters. We explicitly set $N_{\max} \le 2^{96}$ and derive κ_{PC}, $D_{\max}$, and $\Pr[\mathsf{E}_{\mathsf{weak}}]$ using a combination of the analysis in the corrected DARK paper [40] and a subsequent work [39]. In the corrected DARK paper [40], it is proved that κ_{PC} for the DARK polynomial commitment scheme is at most $(3\log|w|)/2^{\lambda}$, where λ is a parameter internal to DARK that introduces a tradeoff between κ_{PC} and $D_{\max}$. By choosing $\lambda = 50$, we get that $\kappa_{\mathrm{PC}} \le (3\log|w|)/2^{\lambda} = 3 \cdot 32/2^{50} < 2^{-43}$. From the analysis and scripts provided in the subsequent work [39], we get that for $D_{\max} \le 2^{1422}$, $\Pr[\mathsf{E}_{\mathsf{weak}}] \le \kappa_{\mathrm{PC}}$. Note that the analysis below is not very sensitive to the magnitude of $D_{\max}$ or $N_{\max}$; however, it is very sensitive to κ_{PC}, and λ is chosen with that in mind.

Next, we need to determine the size of P, the set of 128-bit primes. The number of 128-bit primes is exactly $\pi(2^{128}) - \pi(2^{127})$, where $\pi(x)$ is the prime counting function. By the following standard bound [109]:

$$\pi(2x) - \pi(x) \ge 3x/(5 \cdot \ln x) \quad \forall x \ge 21,$$

we have that

$$|P| \ge \frac{3 \cdot 2^{127}}{5 \cdot 127 \cdot \ln 2} > \frac{3 \cdot 2^{127}}{5 \cdot 128 \cdot \ln 2} = \left(\frac{3}{5 \cdot \ln 2}\right) \cdot 2^{120} > 2^{119}.$$

Additionally, $\min P > 2^{127}$ by definition.

The remainder of the proof proceeds by computing

$$\frac{\lfloor \log_{\min P} D_{\max} \rfloor + \lfloor \log_{\min P} U \rfloor}{|P|} \quad \text{and} \quad \frac{4s + 2 + 2k}{\min P}$$

using the parameters above. This calculation is tedious but mechanical. By definition:

$$\begin{aligned} U &\le m \cdot n^2 \cdot N_{\max}^4 \cdot D_{\max}^4 \cdot (n \cdot N_{\max}^2 + D^2)^2 + D^8 \cdot D_{\max}^4 \\ &\le 2^{32+2\cdot 32+4\cdot 96+4\cdot 1422} \cdot (2^{32+2\cdot 96} + 2^{2\cdot 96})^2 + 2^{8\cdot 96+4\cdot 1422} \\ &\le 2^{32+64+384+5688} \cdot (2^{32+192} + 2^{192})^2 + 2^{768+5688} \\ &\le 2^{6168} \cdot (2^{224} + 2^{192})^2 + 2^{6456} \\ &< 2^{6168} \cdot 2^{450} + 2^{6456} \\ &< 2^{6619}. \end{aligned}$$

It follows that

$$\frac{\lfloor \log_{\min P} D_{\max} \rfloor + \lfloor \log_{\min P} U \rfloor}{|P|} < \frac{\lfloor \frac{1422}{127} \rfloor + \lfloor \frac{6619}{127} \rfloor}{2^{119}} = \frac{63}{2^{119}}.$$

Additionally, we have the following bound on the second term:

$$\frac{4s + 2 + 2k}{\min P} < \frac{4 \cdot 32 + 2 + 2 \cdot 32}{2^{127}} = \frac{194}{2^{127}}.$$

By summing these two terms, $\kappa_{\mathrm{PC}} < 2^{-43}$, and $\Pr[\mathsf{E}_{\mathsf{weak}}] \le \kappa_{\mathrm{PC}}$, we get that the probability that the verifier accepts a non-ϵ-accurate assignment is less than 2^{-40}, as claimed. □

B.5 Deferred proofs

Here we prove Lemma 2 and Lemma 3. We start with two counting lemmas that they use.

Lemma 4. *Given an integer x of magnitude at most M and a set P of primes,*

$$\Pr_{q \sim P}[x \equiv 0 \pmod q] \le \frac{\lfloor \log_{\min P} M \rfloor}{|P|}.$$

Proof. For x to be divisible by a prime q, q must be one of the prime factors of x. Suppose by contradiction that x had more than $\lfloor \log_{\min P} M \rfloor$ prime factors from P (counting multiplicity). Then x would have magnitude at least $(\min P)^{\lfloor \log_{\min P} M \rfloor + 1} > M$, a contradiction. Thus, x has at most $\lfloor \log_{\min P} M \rfloor$ prime factors from P, and the probability that a random prime from P divides x is at most $\lfloor \log_{\min P} M \rfloor / |P|$. □

Lemma 5. *Consider instance $\mathbb{X}$, multilinear polynomial $\tilde{f}$, and vector z, as defined in the protocol in Figure 8. Let D' be the unreduced denominator shared by the coefficients of $\tilde{f}$. Then, the denominator of $\|E_{\mathbb{X},z}\|_2^2$ is exactly $D^8 \cdot D'^4$. The numerator of $J - \|E_{\mathbb{X},z}\|_2^2$ is at most U, where U is as defined in Section B.3, and the denominator of this difference has the same denominator as $\|E_{\mathbb{X},z}\|_2^2$.*

Proof. The proof proceeds by bookkeeping: tracking exact denominators and an upper bound on the magnitude of intermediate numerators. The accounting pessimistically assumes that no reduction occurs when performing arithmetic operations. For example, when adding two rationals (N_1, D_1) and (N_2, D_2) where $\gcd(D_1, D_2) = 1$, the numerator of the result is at most $|N_1| \cdot D_2 + |N_2| \cdot D_1$ in magnitude and the denominator is exactly $D_1 \cdot D_2$.

By definition (§4.1),

$$\|E_{\mathbb{X},z}\|_2^2 = \sum_{r=1}^{m} \left(\sum_{c=1}^{n} A_{r,c} z_c \cdot \sum_{c=1}^{n} B_{r,c} z_c - \sum_{c=1}^{n} C_{r,c} z_c \right)^2.$$

Note that these summations are over m and n summands respectively as opposed to 2^s and 2^k summands as described in Section 4.2.2. This is because we are working with A, B, C as matrices, rather than their multilinear extensions.

Now, recall that the entries of A, B, C, *in*, and *out* all have unreduced denominator D; recall also that the numerators of the entries of A, B, C, *in*, and *out* have magnitude at most $N_{\max}$ (§B.2).

z is a concatenation of *in*, *out*, and w (the definition of w is in Figure 8). To bound the components of w, Weak Integer Binding's sub-property 1 (§B.1) implies that, for the coefficients of $\tilde{f}$, their unreduced denominator, D', satisfies $D' \le D_{\max}$ and their numerators have magnitude at most $N_{\max}$. By definition of $\widetilde{w}$, its coefficients – that is, the components of w – share an unreduced denominator of $D \cdot D'$, and each has

an unreduced numerator of magnitude at most $N_{\max}$. From this, it follows that z, when written with a common unreduced denominator of $D \cdot D'$, has numerators of magnitude at most $N_{\max} \cdot D'$.

For any matrix M, $M_{r,c} z_c$ is a rational with an unreduced numerator of magnitude at most $N_{\max}^2 \cdot D'$ and an unreduced denominator exactly $D^2 \cdot D'$.

Thus, $\sum_{c=1}^{n} M_{r,c} z_c$ is a rational with a numerator of magnitude at most $n \cdot N_{\max}^2 \cdot D'$ and an unreduced denominator exactly $D^2 \cdot D'$.

It follows that $\sum_{c=1}^{n} A_{r,c} z_c \cdot \sum_{c=1}^{n} B_{r,c} z_c$ is a rational with a numerator of magnitude at most $n^2 \cdot N_{\max}^4 \cdot D'^2$ and an unreduced denominator of magnitude exactly $(D^2 \cdot D')^2 = D^4 \cdot D'^2$.

When subtracting $\sum_{c=1}^{n} C_{r,c} z_c$, a rational of the form $(N_1, D^2 \cdot D')$, from $\sum_{c=1}^{n} A_{r,c} z_c \cdot \sum_{c=1}^{n} B_{r,c} z_c$, a rational of the form $(N_2, D^4 \cdot D'^2)$, one needs to multiply the numerator and denominator of the first rational by $D^2 \cdot D'$ to get a common denominator (in the worst case). Thus, after performing the scaling and subtraction, the resulting rational has a numerator of magnitude at most

$$n^2 \cdot N_{\max}^4 \cdot D'^2 + n \cdot N_{\max}^2 \cdot D^2 \cdot D'^2$$

and an unreduced denominator of magnitude exactly $D^4 \cdot D'^2$.

Squaring this rational to get the contribution of row r to $\|E_{\mathbb{X},z}\|_2^2$ results in a rational with a numerator of magnitude at most

$$(n^2 \cdot N_{\max}^4 \cdot D'^2 + n \cdot N_{\max}^2 \cdot D^2 \cdot D'^2)^2$$

and an unreduced denominator of magnitude exactly $D^8 \cdot D'^4$.

Summing over m rows to get $\|E_{\mathbb{X},z}\|_2^2$ does not change the denominator so results in a rational with denominator $D^8 \cdot D'^4$ (as claimed) and a numerator of magnitude at most

$$m \cdot (n^2 \cdot N_{\max}^4 \cdot D'^2 + n \cdot N_{\max}^2 \cdot D^2 \cdot D'^2)^2$$

which simplifies to

$$m \cdot n^2 \cdot N_{\max}^4 \cdot D'^4 \cdot (n \cdot N_{\max}^2 + D^2)^2.$$

To bound the numerator and denominator of $J - \|E_{\mathbb{X},z}\|_2^2$, we first need to characterize the numerator and denominator of J. To do this, we consider $\mathcal{V}$'s checks in Step 3. To pass these checks, J must be a rational of the form (S, D^8) where S is a positive integer bounded above by D^8 (since $0 \le J \le \epsilon^2 \le 1$). To perform the subtraction $J - \|E_{\mathbb{X},z}\|_2^2$, one needs to multiply the numerator and denominator of J by D'^4 to get a common denominator (in the worst case). The resulting rational $J - \|E_{\mathbb{X},z}\|_2^2$ has a numerator of magnitude at most

$$m \cdot n^2 \cdot N_{\max}^4 \cdot D'^4 \cdot (n \cdot N_{\max}^2 + D^2)^2 + S \cdot D'^4$$

and an unreduced denominator of magnitude exactly $D^8 \cdot D'^4$, as claimed.

Given that $D' \le D_{\max}$ and $S \le D^8$, the numerator is upper-bounded by

$$m \cdot n^2 \cdot N_{\max}^4 \cdot D_{\max}^4 \cdot (n \cdot N_{\max}^2 + D^2)^2 + D^8 \cdot D_{\max}^4.$$

This is exactly U, as claimed. □

With the prime-divisibility bound (Lemma 4) and the numerator bound (Lemma 5) in hand, the two ρ_q facts quickly follow.

Proof of Lemma 2 (ρ_q ill-defined). ρ_q is undefined on $\|E_{\mathbb{X},z}\|_2^2$ exactly when q divides the denominator of $\|E_{\mathbb{X},z}\|_2^2$. By Lemma 5, the denominator is a product of powers of D and D' (where $D' \le D_{\max}$). By construction, q does not divide D; thus q divides the denominator of $\|E_{\mathbb{X},z}\|_2^2$ if and only if q divides D'. Applying Lemma 4 with $M = D_{\max}$ gives the bound. □

Proof of Lemma 3 (ρ_q collision). $\rho_q(J) = \rho_q(\|E_{X,z}\|_2^2)$ if and only if q divides the numerator of $J - \|E_{X,z}\|_2^2$, which, by Lemma 5, has magnitude at most U. Applying Lemma 4 with $M = U$ gives the desired bound. □

C DARK with verifier-known group order

This appendix details the technique mentioned in Section 4.2.3.

We consider an RSA group $\mathbb{G}$ with generator g and modulus $M = p_1 \cdot p_2$. The verifier knows p_1 and p_2, while the prover only knows g and M but not its factorization.

DARK requires, as a primitive, a way for the prover to convince the verifier that $g^z = g^x \cdot g^{c \cdot y}$ given g^x, g^y, and g^z and a constant c known to both parties. The problem is computing $(g^y)^c$; the verifier cannot efficiently compute an exponentiation this large, because it requires $\lceil \log c \rceil$ multiplications, which is potentially on the order of a million or a billion (the combined bit-length of all witness elements). DARK solves this with a proof of exponentiation protocol that is burdensome for the prover. In Spain, the verifier borrows an optimization from RSA signing algorithms when the signer knows the factorization of M.

The key observation is that knowledge of the factorization of M allows the verifier to compute $\lambda(M)$, where λ is Carmichael's function [1, 43]. Then the verifier can compute $c' = c \bmod \lambda(M)$, a value on the order of M rather than c itself, and compute $(g^y)^{c'}$ instead of $(g^y)^c$.

The correctness of this substitution is as follows. g^y is a group element and thus is relatively prime to M, so by the definition of Carmichael's function, we have $(g^y)^{\lambda(M)} \equiv 1 \pmod{M}$, and thus $(g^y)^{c'} \equiv (g^y)^c \pmod{M}$.

This substitution is efficient because $\log c' \approx \log M$, which is at worst a few thousand rather than a few billion.

At a high level, replacing the proof of exponentiation in DARK with an explicit exponentiation, independent of how the verifier computes, does not change the security of DARK.

Block et al. [29, Lemma 6.4] formally prove this in the context of a verifier that does not know the factorization of M and opts to compute $(g^y)^c$ directly. Their proof applies to our setting as well, as the verifier's computation of $(g^y)^{c'}$ is functionally equivalent to computing $(g^y)^c$ directly.

D Protocol extensions

Spain can be adapted and extended in various ways.

D.1 SIMD-R1CS

Per Section 4.3, Spain can be adapted to support common variations on R1CS, namely SIMD-R1CS and I-R1CS.

In SIMD-R1CS (§4.3), the prover has L instances, each with its own public input and witness, but sharing the same R1CS structure (A, B, C). That is, there are vectors $z_0, \ldots, z_{L-1}$, corresponding to instances $\mathbb{X}_0, \ldots, \mathbb{X}_{L-1}$, where all instances are the same R1CS structure. To support SIMD-R1CS, define $z(c, j)$ similarly to $z(c)$. Here j selects the particular instance and c selects the variable within that instance (similar to how c functioned earlier). Given the multilinear extension $\widetilde{z}(t_1, t_2)$, define

$$\widetilde{h_M}(t_1, t_2) := \sum_{c \in \{0,1\}^k} \widetilde{M}(t_1, c) \cdot \widetilde{z}(c, t_2),$$

Then there are two options for proving correctness across all L instances.

The first option is for the prover to make a single claim J about the sum of squared errors across all L instances. Then, the verifier can confirm that $J \leq \epsilon^2$ and the two parties can apply the sum-check protocol to prove that

$$\rho_q(J) = \sum_{j \in \{0,1\}^\ell} \sum_{r \in \{0,1\}^s} \left(\widetilde{h_A}(r, j) \cdot \widetilde{h_B}(r, j) - \widetilde{h_C}(r, j) \right)^2$$

where $\ell = \lceil \log_2 L \rceil$. This provides Back-end ϵ-soundness and Back-end $(\epsilon / \sqrt{L \cdot m})$-completeness. Notice that completeness degrades with L here.

The second option is to have the prover make $L - 1$ separate claims, where for $j = \{0, \ldots, L-1\}$, each claim J_j is about $\|E_{\mathbb{X}_j, z_j}\|_2^2$ for the respective instance $\mathbb{X}_j$. The verifier then checks that each J_j is at most ϵ^2. Then, the prover and verifier interpret $\rho_q(J) = [\rho_q(J_0), \ldots, \rho_q(J_{L-1})]$ as a function (from index bits to value) to get its multilinear extension: $\widetilde{J}(t)$. Finally, the verifier chooses a random $\tau \in \mathbb{F}_q^\ell$ and uses polynomial identity testing (similar to Spartan [116]) to prove that $\widetilde{J}(\tau)$ is equal to

$$\sum_{j \in \{0,1\}^\ell} \widetilde{Eq}(\tau, j) \sum_{r \in \{0,1\}^s} \left(\widetilde{h_A}(r, j) \cdot \widetilde{h_B}(r, j) - \widetilde{h_C}(r, j) \right)^2.$$

This provides Back-end ϵ-soundness and Back-end $(\epsilon / \sqrt{m})$-completeness. Compared to the first option, completeness does not degrade with L here; however, the expression is degree 5 in j rather than degree 4, which mildly increases prover and verifier time.

D.2 I-R1CS

For I-R1CS (§4.3), the modification to Spain is simply: at the end of the protocol, rather than opening a single commitment, the prover opens all commitments $\widetilde{w}_1, \ldots, \widetilde{w}_R$.

Recall that Spain introduces an additional optimization: Spain allows for the prover and verifier to collaboratively introduce not only new values for the *assignment* but also new *constraints on the witness* in each round. This allows the verifier to encode random challenges in constraints, or to partially specify constraints that are "filled-in" by the prover. These techniques reduce the cost of performing dot products between the witness and random vectors, which is the core operation in checkers for matrix multiplication [56, 74]

D.3 Different norms for error measurement

Spain can be generalized to have a smaller gap between soundness and completeness versus the one presented in the paper, namely $\epsilon_C = \epsilon / \sqrt{m}$. Doing so lowers the precision required of the prover's witness generation (see §6.3 for the interplay between "precision of witness generation" and ϵ_C) but makes the underlying back-end protocol more costly.

Specifically, by using the ℓ^{2t} norm rather than the ℓ^2 norm of $E_{\mathbb{X},z}$, for integer $t > 1$, Spain will continue to have Back-end ϵ-soundness but now it will have Back-end $(\epsilon / m^{1/(2t)})$-completeness. That is because the following chain of inequalities holds:

$$\|E_{\mathbb{X},z}\|_\infty^{2t} \leq \|E_{\mathbb{X},z}\|_{2t}^{2t} \leq m \cdot \|E_{\mathbb{X},z}\|_\infty^{2t}$$

for any integer $t > 1$. To adapt Spain to use the ℓ^{2t} norm, the prover and verifier start with the following equation in place of Equation (4) in Section 4.2.2 (Step 5 in Figure 8):

$$\rho_q(J) = \sum_{r \in \{0,1\}^s} \left(\widetilde{g_A}(r) \cdot \widetilde{g_B}(r) - \widetilde{g_C}(r) \right)^{2t}.$$

This is a degree-$(4t)$ polynomial in r, so as t increases, so does prover time, verifier time, and communication.

E Translation fidelity

This appendix defines translation fidelity and covers some technical details of this concept. The bulk of the appendix analyzes the translation fidelity of the arithmetizations presented in Section 5.

E.1 Preliminaries

Definitions. Given an R1CS structure $\mathbb{S} = (A, B, C)$, recall that all entries in A, B, C are presumed to have a given denominator D, and likewise with the assignment z (§4.2.1, Appx. B). We call such numbers η-*multiples*, where $\eta = 1/D$.

Let $F_\delta(x)$ denote the possible outputs of function F on input x when its individual operations have error bounded by δ.

Recall the distinction between δ and δ_{wg} (§3). δ is part of the underlying protocol's soundness guarantee: if any operation in an execution has error greater than δ, the verifier is supposed to reject with high probability. On the other hand, δ_{wg} connects to the protocol's *completeness* guarantee: we want to arrange the design and implementation so that if the prover limits the per-operation error to δ_{wg}, then it *can* satisfy all constraints with suitable error, and thereby cause the verifier to accept.

The reason for the asymmetry is technical. It relates to the definitions of soundness and completeness in the front-end (given in the next paragraph), and to the fact that ϵ and ϵ_C are different, which itself stems from the fact that $\|E_{\mathbb{X},z}\|_\infty \leq \epsilon$ does not imply $\|E_{\mathbb{X},z}\|_2^2 \leq \epsilon^2$ (§4.1).

Now, consider a purported translation of F to an R1CS structure $\mathbb{S} = (A, B, C)$. Translation fidelity is two properties:

- The translation is *ϵ-tf-sound* if: for all ϵ-accurate assignments $(in, out, 1, w)$ to $\mathbb{S}$, $out \in F_\delta(in)$.
- The translation is *ϵ_C-tf-complete* if: for all (in, out) such that $out \in F_{\delta_{wg}}(in)$ and (in, out) contains only η-multiples, there exists an accompanying w such that $(in, out, 1, w)$ is an ϵ_C-accurate assignment to $\mathbb{S}$.

It is helpful to keep the following ordering in mind:

$$\begin{aligned} \delta &\geq \epsilon && \text{(will be established in the examples below)} \\ &> \epsilon_C && (\S 4.1) \\ &\geq \delta_{wg} && \text{(will be established in the examples below)} \end{aligned}$$

Per-operation vs per-function translation fidelity. The definitions of tf-soundness and tf-completeness cover translation of an entire *function*. However, the arithmetizations that we present in the next section will be at the level of *individual operations*. For these lower-level translations to be relevant, Spain needs a way to combine the translations of individual operations. The needed concept is *composition* (of operations or functions), and specifically how translations combine under composition. The description below delves into detail; the high-level point is that focusing on the translations of individual operations is justified.

Given two functions F and G, with corresponding R1CS instances $\mathbb{X}_F = (A_F, B_F, C_F, in_F, out_F)$ and $\mathbb{X}_G = (A_G, B_G, C_G, in_G, out_G)$, the corresponding instance for $G \circ F$, $\mathbb{X}_{G\circ F}$, is (A, B, C, in, out) where $in = in_F$, $out = out_G$, and A, B, C contain the constraints of A_F, B_F, C_F and A_G, B_G, C_G, relabeled so that the output variables of $\mathbb{X}_F$ and the input variables of $\mathbb{X}_G$ refer to the same set of variables, which we call *med*; the variables in *med* are not part of either *in* or *out*. The lemma below states that translation fidelity is preserved by the composition of functions.

Lemma 6. *If F and $\mathbb{X}_F$ satisfy ϵ-tf-soundness and ϵ_C-tf-completeness, and if G and $\mathbb{X}_G$ satisfy ϵ'-tf-soundness and ϵ'_C-tf-completeness, then $G \circ F$ and $\mathbb{X}_{G\circ F}$ satisfy* $\min\{\epsilon, \epsilon'\}$*-tf-soundness and* $\max\{\epsilon_C, \epsilon'_C\}$*-tf-completeness.*

Proof. We first establish $\min\{\epsilon, \epsilon'\}$-tf-soundness, where $\epsilon^* := \min\{\epsilon, \epsilon'\}$. The constraints of $\mathbb{X}_{G\circ F}$ are those of $\mathbb{X}_F$ together with those of $\mathbb{X}_G$, sharing only the variables *med*. An ϵ^*-accurate assignment to the composite therefore restricts to an ϵ^*-accurate assignment to each part; as $\epsilon^* \leq \epsilon$ and $\epsilon^* \leq \epsilon'$, these are in particular ϵ- and ϵ'-accurate. Then ϵ-tf-soundness of F gives $med \in F_\delta(in)$, and ϵ'-tf-soundness of G gives $out \in G_\delta(med)$, so $out \in (G \circ F)_\delta(in)$.

For tf-completeness, set $\epsilon^* := \max\{\epsilon_C, \epsilon'_C\}$. Consider (in, out) where in and out contain η-multiples and $out \in (G \circ F)_{\delta_{wg}}(in)$. By definition of the composite, there must be *med*, consisting of η-multiples, with $med \in F_{\delta_{wg}}(in)$ and $out \in G_{\delta_{wg}}(med)$. Applying ϵ_C-tf-completeness of F, there exists w_F, where $(in, med, 1, w_F)$ is an ϵ_C-accurate assignment to (A_F, B_F, C_F). Likewise, there exists w_G, where $(med, out, 1, w_G)$ is an ϵ'_C-accurate assignment to (A_G, B_G, C_G). Combining w_F, w_G, and *med* into w, the assignment $(in, out, 1, w)$ satisfies each constraint of $\mathbb{X}_{G\circ F}$ with error upper-bounded by ϵ_C or ϵ'_C, hence upper-bounded by ϵ^*. □

E.2 Analysis of arithmetizations

The ϵ and ϵ_C described for the translations below depend on the magnitudes of the inputs to these subfunctions. This is not problematic because Spain enforces a strict bound on the magnitude of rationals in the proof protocol (§B). By considering this maximum magnitude, one derives absolute bounds on ϵ and ϵ_C.

$\mathbf{z \leftarrow x/y}$. Division is a special case where the error bound is inversely proportional to the magnitude of y, and thus one must place restrictions on the domain of y or perform static analysis to ensure y is not too close to 0 when using this operation. Here we will assume that $|y| \geq M$ for some known $M > 0$.

First, we establish ϵ-tf-soundness. Suppose we have an ϵ-accurate assignment to $y \cdot z \approx_\epsilon x$. That means $|y \cdot z - x| \leq \epsilon$. Rearranging, we have that $|z - (x/y)| \leq \epsilon/|y|$. This implies that z is within $\epsilon/|y|$ of x/y, and thus $|z - (x/y)| \leq \epsilon/M$ (using the assumed lower bound on $|y|$). Letting $\delta = \epsilon/M$ completes the proof of ϵ-tf-soundness.

Second, for ϵ_C-tf-completeness, consider all (x, y, z), restricted to η-multiples, such that $z = (x/y) + \widehat{\delta_\pm}$ where $|\widehat{\delta_\pm}| \leq \delta_{wg}$. The constraint has error at most

$$|y \cdot z - x| = |y \cdot ((x/y) + \widehat{\delta_\pm}) - x| = |y \cdot \widehat{\delta_\pm}| \leq \delta_{wg}|y|.$$

This establishes ϵ_C-tf-completeness provided that $\epsilon_C \geq \delta_{wg}|y|$ for all inputs y.

$\mathbf{y} \leftarrow \sqrt{\mathbf{x}}$. First, we establish ϵ-tf-soundness. Suppose we have an ϵ-accurate assignment to $y \cdot y \approx_\epsilon x$. That means $|y^2 - x| \leq \epsilon$. This can be rewritten as $|y - \sqrt{x}| \cdot |y + \sqrt{x}| \leq \epsilon$, where $\sqrt{x}$ is a true square root of x, and assume WLOG that $\sqrt{x}$ is the positive root. This implies that either $|y - \sqrt{x}|$ or $|y + \sqrt{x}|$ is less than or equal to $\sqrt{\epsilon}$. This, in turn, implies that y is within $\sqrt{\epsilon}$ of a true square root of x. Letting $\delta = \sqrt{\epsilon}$ finishes the proof for ϵ-tf-soundness.

Second, for ϵ_C-tf-completeness, consider all (x, y), restricted to η-multiples, such that $y = \sqrt{x} + \widehat{\delta_\pm}$ where $\widehat{\delta_\pm}$ is a real number with $|\widehat{\delta_\pm}| \leq \delta_{wg}$. The constraint $y \cdot y \approx_\epsilon x$ is satisfied with error at most

$$\left|(\sqrt{x} + \widehat{\delta_\pm})^2 - x\right| \leq 2\delta_{wg}|\sqrt{x}| + \delta_{wg}^2.$$

This establishes ϵ_C-tf-completeness provided that $\epsilon_C \geq 2\delta_{wg}|\sqrt{x}| + \delta_{wg}^2$ for all inputs x.

assert $(\mathbf{x} \geq \mathbf{y})$. Recall that this is an assert of the form "greater-than-or-approximately equal" where x and y are allowed (but not required) to be "approximately-equal" when they are within δ of each other.

First, we establish ϵ-tf-soundness. Suppose we have an ϵ-accurate assignment to $t \cdot t \approx_\epsilon x - y$. That means $|t^2 - (x - y)| \leq \epsilon$. By definition, $t^2 \geq 0$. This implies that $x - y \geq -\epsilon$. Letting $\delta = \epsilon$ finishes the proof for ϵ-tf-soundness.

Second, for ϵ_C-tf-completeness, consider all x, y, restricted to η-multiples, such that $x \geq y$. Let t be the nearest η-multiple to $\sqrt{x - y}$. Thus, $t = \sqrt{x - y} + \widehat{\eta_\pm}$ where $|\widehat{\eta_\pm}| \leq \eta/2$. Then the constraint $t \cdot t \approx_\epsilon x - y$ is satisfied with error at most

$$|t^2 - (x - y)| = |2\widehat{\eta_\pm}\sqrt{x - y} + \widehat{\eta_\pm}^2| \leq \eta\sqrt{x - y} + \eta^2/4.$$

This establishes ϵ_C-tf-completeness provided that $\epsilon_C \geq \eta\sqrt{x - y} + \eta^2/4$.

$\mathbf{b} \leftarrow (\mathbf{x} \geq \mathbf{y})$. First, we establish ϵ-tf-soundness for $\epsilon < 1/8$. Suppose we have an ϵ-accurate assignment to the constraints: $b \cdot (1 - b) \approx_\epsilon 0$, $(2b - 1) \cdot (x - y) \approx_\epsilon t$, and $s \cdot s \approx_\epsilon t$. Using the same reasoning as in proving ϵ-tf-soundness for the assert statement, the third constraint implies $t \geq -\epsilon$.

The first constraint implies $|b \cdot (1 - b)| \leq \epsilon$. By our bound on ϵ and the quadratic formula applied to $-\epsilon \leq b \cdot (1 - b) \leq \epsilon$, we have

$$b \in \left[\frac{1 - \sqrt{1 + 4\epsilon}}{2}, \frac{1 - \sqrt{1 - 4\epsilon}}{2}\right]$$

or

$$b \in \left[\frac{1 + \sqrt{1 - 4\epsilon}}{2}, \frac{1 + \sqrt{1 + 4\epsilon}}{2}\right].$$

Given that $\sqrt{1 - 4\epsilon} \geq 1 - 4\epsilon$ for $\epsilon < 1/4$, and $\sqrt{1 + 4\epsilon} \leq 1 + 4\epsilon$ for $\epsilon < 1/4$, this implies that $b \in [-2\epsilon, 2\epsilon] \cup [1 - 2\epsilon, 1 + 2\epsilon]$, namely that b is within 2ϵ of either 0 or 1.

The second constraint requires the most involved analysis. An ϵ-accurate assignment implies $|(2b - 1) \cdot (x - y) - t| \leq \epsilon$. Given that $t \geq -\epsilon$, this implies $(2b - 1) \cdot (x - y) \geq -2\epsilon$. Let $b' := 2b - 1$. From the bounds on b from before

$$b' \in [-1 - 4\epsilon, -1 + 4\epsilon] \cup [1 - 4\epsilon, 1 + 4\epsilon].$$

We also have $b' \cdot (x - y) \geq -2\epsilon$. We consider two cases, depending on which interval b' falls into.

Case 1: $b' \in [-1 - 4\epsilon, -1 + 4\epsilon]$. In this case, we have

$$x - y \leq \frac{2\epsilon}{1 - 4\epsilon}.$$

Given that $\epsilon \leq 1/8$, we have

$$\frac{2\epsilon}{1 - 4\epsilon} \leq \frac{2\epsilon}{1/2} = 4\epsilon.$$

Case 2: $b' \in [1 - 4\epsilon, 1 + 4\epsilon]$. In this case, we have

$$x - y \geq \frac{-2\epsilon}{1 - 4\epsilon}.$$

Given that $\epsilon \leq 1/8$, we have

$$\frac{-2\epsilon}{1 - 4\epsilon} \geq \frac{-2\epsilon}{1/2} = -4\epsilon.$$

Summarizing the two cases: $b \in [-2\epsilon, 2\epsilon]$ (which is case 1) implies $x - y \leq 4\epsilon$, while $b \in [1 - 2\epsilon, 1 + 2\epsilon]$ (which is case 2) implies $x - y \geq -4\epsilon$, or $y - x \leq 4\epsilon$.

Taking the contrapositive of both

$$x - y > 4\epsilon \implies b \in [1 - 2\epsilon, 1 + 2\epsilon]$$
$$y - x > 4\epsilon \implies b \in [-2\epsilon, 2\epsilon].$$

Finally, if $|x - y| \leq 4\epsilon$, no further conclusion can be drawn about b, except that it must remain in the intervals derived above. This means it must be within 2ϵ of either 0 or 1.

By taking $\delta = 4\epsilon$, the relationships among x, y, and b obey the intended semantics of the operation, and thus ϵ-tf-soundness holds.

Second, we establish ϵ_C-tf-completeness, provided that $\delta_{wg} \leq 1/2$. Consider all (x, y, b), restricted to η-multiples, such that $b = 1 + \widehat{\delta_\pm}$ if $x \geq y$ and $b = \widehat{\delta_\pm}$ otherwise, where $\widehat{\delta_\pm}$ is a real number with $|\widehat{\delta_\pm}| \leq \delta_{wg}$.

For the first constraint there are two cases to consider.

Case 1: suppose $x \geq y$ and thus $b = 1 + \widehat{\delta_\pm}$. The constraint is satisfied with error at most

$$|b \cdot (1 - b)| = |(1 + \widehat{\delta_\pm}) \cdot (-\widehat{\delta_\pm})| \leq \delta_{wg} + \delta_{wg}^2.$$

Case 2: suppose $x < y$ and thus $b = \widehat{\delta_\pm}$. The upper bound on error is the same:

$$|b \cdot (1 - b)| = |\widehat{\delta_\pm} \cdot (1 - \widehat{\delta_\pm})| \leq \delta_{wg} + \delta_{wg}^2.$$

For the second constraint, take $t = (2b - 1) \cdot (x - y)$. The constraint is then satisfied with zero error.

For the third constraint, take s to be the nearest η-multiple to $\sqrt{t}$. Thus, $s = \sqrt{t} + \widehat{\eta_\pm}$ where $|\widehat{\eta_\pm}| \le \eta/2$. Such a square root exists because $t \ge 0$. This follows because $(2b-1)$ and $(x-y)$ have the same sign, as long as $\delta_{wg} \le 1/2$. Now, we show $t \le 2 \cdot |x-y|$, as follows. If $x \ge y$, then $b = 1 + \widehat{\delta_\pm}$ and thus

$$t = (2b-1)\cdot(x-y) = (1+2\widehat{\delta_\pm})\cdot|x-y| \le 2\cdot|x-y|.$$

If $x < y$, then $b = \widehat{\delta_\pm}$ and thus

$$t = (2b-1)\cdot(x-y) = (1-2\widehat{\delta_\pm})\cdot|x-y| \le 2\cdot|x-y|.$$

Given this, the third constraint is satisfied with error at most

$$\begin{aligned}|s^2 - t| &= |(\sqrt{t} + \widehat{\eta_\pm})^2 - t| \\ &= |2\widehat{\eta_\pm}\sqrt{t} + \widehat{\eta_\pm}^2| \\ &\le \eta\sqrt{t} + \eta^2/4 \\ &\le \eta\sqrt{2\cdot|x-y|} + \eta^2/4.\end{aligned}$$

This establishes ϵ_C-tf-completeness provided that $\epsilon_C \ge \max\{\delta_{wg} + \delta_{wg}^2, \eta\sqrt{2\cdot|x-y|} + \eta^2/4\}$.

$\mathbf{z} \leftarrow \max\{\mathbf{v_1}, \ldots, \mathbf{v_L}\}$. Recall the constraints from Section 5:

- for all $i \in \{1,\ldots,L\}$: $t_i \cdot t_i \approx_\epsilon z - v_i$
- for all $i \in \{1,\ldots,L\}$: $b_i \cdot (1-b_i) \approx_\epsilon 0$
- for all $i \in \{1,\ldots,L\}$: $b_i \cdot (z - v_i) \approx_\epsilon 0$
- $\sum_{i=1}^L b_i \approx_\epsilon 1$

First, we establish ϵ-tf-soundness, for all $\epsilon \le 1/4$ when $\epsilon \cdot (2L+1) < 1$. Suppose we have an ϵ-accurate assignment to the constraints. For the first set of constraints, we can reuse the results from the assert operation to establish that $z \ge v_i - \epsilon$ for all i. From the second set of constraints, we can use the same reasoning as in $b \leftarrow (x \ge y)$ to establish that each b_i is within 2ϵ of either 0 or 1. Given that $\epsilon\cdot(2L+1) < 1$, the last constraint and the second set together imply that there exists some index j such that b_j is within 2ϵ of 1. This follows by contradiction, suppose this was not the case, then each b_i would be within 2ϵ of 0, and thus $\sum_{i=1}^L b_i \le 2L\cdot\epsilon < 1-\epsilon$, leaving the final constraint unsatisfied by more than ϵ.

This leaves the third set of constraints to analyze. Let j be the index for which $b_j \ge 1 - 2\epsilon$. Since the assignment is ϵ-accurate, $|b_j \cdot (z - v_j)| \le \epsilon$. Thus:

$$|z - v_j| \le \frac{\epsilon}{b_j} \le \frac{\epsilon}{1-2\epsilon}.$$

In particular:

$$z \le v_j + \frac{\epsilon}{1-2\epsilon} \le \max_i v_i + \frac{\epsilon}{1-2\epsilon}.$$

We already have $z \ge v_i - \epsilon$ for all i, which implies

$$z \ge \max_i v_i - \epsilon,$$

and since $\epsilon < \frac{\epsilon}{1-2\epsilon}$ also

$$z \ge \max_i v_i - \frac{\epsilon}{1-2\epsilon}.$$

Thus, $|z - \max_i v_i| \le \frac{\epsilon}{1-2\epsilon}$. Given that $\epsilon \le 1/4$, $\frac{\epsilon}{1-2\epsilon} \le \frac{\epsilon}{1/2} = 2\epsilon$. Setting $\delta = 2\epsilon$ completes the proof of ϵ-tf-soundness.

Second, we establish ϵ_C-tf-completeness. Consider all $(v_1,\ldots,v_L,z)$, restricted to η-multiples, such that $z = \max_i v_i + \widehat{\delta_\pm}$ for some $\widehat{\delta_\pm}$ where $|\widehat{\delta_\pm}| \le \delta_{wg}$.

Take $b_j = 1$ for the index j corresponding to the maximum value, and set all other b_i to 0.

For the first set of constraints, we use that $z - v_i \ge -\delta_{wg}$. This follows from the fact that $z = \max_i v_i + \widehat{\delta_\pm}$ and thus $z - v_i = \max_i(v_i) + \widehat{\delta_\pm} - v_i \ge \widehat{\delta_\pm} \ge -\delta_{wg}$. Now there are two cases to consider.

Case 1: suppose $z - v_i \ge 0$. In this case, set t_i to be the nearest η-multiple to $\sqrt{z-v_i}$. Thus, $t_i = \sqrt{z-v_i} + \widehat{\eta_{\pm i}}$, where $|\widehat{\eta_{\pm i}}| \le \eta/2$. These constraints are satisfied with error at most

$$\begin{aligned}|t_i^2 - (z-v_i)| &= |(\sqrt{z-v_i} + \widehat{\eta_{\pm i}})^2 - (z-v_i)| \\ &= |2\widehat{\eta_{\pm i}}\sqrt{z-v_i} + \widehat{\eta_{\pm i}}^2| \\ &\le \eta\sqrt{\max_i(v_i) - \min_i(v_i) + \delta_{wg}} + \eta^2/4.\end{aligned}$$

Case 2: suppose $z - v_i < 0$. In this case, take $t_i = 0$. These constraints are satisfied with error at most

$$|t_i^2 - (z-v_i)| = |z - v_i| \le \delta_{wg}.$$

The second and fourth sets of constraints are satisfied with zero error, by the choice of b_i above.

The third set of constraints breaks into two cases. When $b_i = 0$, the ith constraint in the set is satisfied with zero error. When $b_i = 1$ (that is, when $i = j$), constraint j in the set is satisfied with error at most δ_{wg}, as $b_j \cdot (z - v_j) = z - v_j = \widehat{\delta_\pm} \le \delta_{wg}$.

This establishes ϵ_C-tf-completeness provided that

$$\epsilon_C \ge \max\left\{\delta_{wg}, \eta\sqrt{\max_i(v_i) - \min_i(v_i) + \delta_{wg}} + \eta^2/4\right\}.$$

ReLU. First, we establish ϵ-tf-soundness for $\epsilon \le 1$. Suppose we have an ϵ-accurate assignment to the constraints: $s \cdot s \approx_\epsilon z - x$, $t \cdot t \approx_\epsilon z$, and $(z-x)\cdot z \approx_\epsilon 0$ From the first and second constraints, we have that $z \ge -\epsilon + \max(0,x)$. The third constraint is where things diverge from prior analyses. An ϵ-accurate assignment implies $|(z-x)\cdot z| \le \epsilon$. This can be decomposed as $|z-x|\cdot|z| \le \epsilon$. For the product on the left to be at most ϵ, then at least one of the two factors must be at most $\sqrt{\epsilon}$. Thus z is within $\sqrt{\epsilon}$ of either 0 or x.

Now there are two cases to consider.

Case 1: suppose $x \ge 0$. Here z is at least $x - \epsilon$ by the first two constraints, and at most $x + \sqrt{\epsilon}$ by the third constraint. Thus, as required, z is within $\sqrt{\epsilon}$ of x.

Case 2: suppose $x < 0$. Here z is at least $-\epsilon$ by the first two constraints, and at most $\sqrt{\epsilon}$ by the third constraint. Thus, as required, z is within $\sqrt{\epsilon}$ of 0.

By combining these cases, we get that z is within $\sqrt{\epsilon}$ of $\max(0, x)$. Setting $\delta = \sqrt{\epsilon}$ completes the proof.

Second, we establish ϵ_C-tf-completeness. Consider all (x, z), restricted to η-multiples, such that $z = \max(0, x) + \widehat{\delta_\pm}$ where $\widehat{\delta_\pm}$ is a real number with $|\widehat{\delta_\pm}| \leq \delta_{wg}$. We consider each constraint individually and take the maximum error across all constraints.

Constraint 1: $s \cdot s \approx_\epsilon z - x$. This constraint has error at most

$$|s^2 + x - (\max(0, x) + \widehat{\delta_\pm})| = |s^2 + \min(x, 0) - \widehat{\delta_\pm}|. = |s^2 + x'|,$$

where $x' := \min(x, 0) - \widehat{\delta_\pm}$. x' is an η-multiple $\leq \delta_{wg}$. There are two cases to consider.

Case 1: suppose $x' > 0$. In this case, setting $s = 0$ satisfies this constraint with error at most δ_{wg}.

Case 2: suppose $x' \leq 0$. In this case, set s to the closest η-multiple to $\sqrt{-x'}$. Thus, $s = \sqrt{-x'} + \widehat{\eta_\pm}$, for some $\widehat{\eta_\pm}$ where $|\widehat{\eta_\pm}| \leq \eta/2$. Then this constraint has error at most

$$\begin{aligned} |(\sqrt{-x'} + \widehat{\eta_\pm})^2 + x'| &= |2\widehat{\eta_\pm}\sqrt{-x'} + \widehat{\eta_\pm}^2| \\ &\leq \eta\sqrt{-x'} + \eta^2/4 \\ &\leq \eta\sqrt{|x| + \delta_{wg}} + \eta^2/4. \end{aligned}$$

Thus the first constraint is satisfied with error at most

$$\max\left\{\delta_{wg}, \eta\sqrt{|x| + \delta_{wg}} + \eta^2/4\right\}.$$

Constraint 2: $t \cdot t \approx_\epsilon z$. This constraint has error at most

$$|t^2 - (\max(0, x) + \widehat{\delta_\pm})|.$$

The analysis is similar to the one for the first constraint. The cases here are $z \leq 0$ (in which case let $t = 0$) and $z > 0$ (in which case let t be the closest η-multiple to $\sqrt{z}$). These settings satisfy this constraint with error at most

$$\max\left\{\delta_{wg}, \eta\sqrt{|x| + \delta_{wg}} + \eta^2/4\right\}.$$

Constraint 3: $(z - x) \cdot z \approx_\epsilon 0$. This constraint is satisfied with error at most

$$\left|(\max(0, x) + \widehat{\delta_\pm} - x) \cdot (\max(0, x) + \widehat{\delta_\pm})\right| \leq \delta_{wg}|x| + \delta_{wg}^2.$$

Combining the above, we have that all constraints are satisfied with error at most

$$\max\left\{\delta_{wg}, \eta\sqrt{|x| + \delta_{wg}} + \eta^2/4, \delta_{wg}|x| + \delta_{wg}^2\right\}.$$

This establishes ϵ_C-tf-completeness provided that

$$\epsilon_C \geq \max\left\{\delta_{wg}, \eta\sqrt{|x| + \delta_{wg}} + \eta^2/4, \delta_{wg}|x| + \delta_{wg}^2\right\}.$$

E.3 On choosing between alternative arithmetizations

When using traditional constraints, the fewer constraints in an arithmetization the better. However, with approximate constraints, this is not always the case. Take ReLU as an example. The ReLU arithmetization in Section E.2 is 3 constraints and satisfies ϵ-tf-soundness with $\delta = \sqrt{\epsilon}$. By contrast, if the following 5 constraints are used (derived from the max primitive):

- $s \cdot s \approx_\epsilon z - x$
- $t \cdot t \approx_\epsilon z$
- $b \cdot (1 - b) \approx_\epsilon 0$
- $b \cdot (z - x) \approx_\epsilon 0$
- $(1 - b) \cdot z \approx_\epsilon 0$

then ϵ-tf-soundness is satisfied with $\delta = 2\epsilon$.

Despite using almost twice as many constraints, the second arithmetization produces a significantly tighter relationship between ϵ and δ. This in turn enables a larger ϵ_C and δ_{wg} (which means lower precision is needed in witness generation). In settings where high-precision witness generation is a bottleneck, the second arithmetization may be preferable, in spite of using a larger number of constraints.

E.4 Hybrid arithmetizations

Spain exclusively handles approximate constraints. Here we sketch a short exploration of a hybrid setting where there are two structures, $\mathbb{S}_\epsilon$ and $\mathbb{S}_{\text{exact}}$, that enforce approximate and traditional constraints respectively on the same assignment z.

By appending to the definition of ϵ-accuracy the requirement that the traditional constraints (those in $\mathbb{S}_{\text{exact}}$) are satisfied with zero error, the notion of translation fidelity can be extended to this hybrid setting. To demonstrate, we step through the analysis of a simple operation, $b \leftarrow_{\text{exact}} (x \geq y)$, where b is exactly 0 or 1 and the comparison is approximate. Consider a hybrid arithmetization of this primitive that has two traditional constraints: $b \cdot (1 - b) = 0$ and $(2b - 1) \cdot (x - y) = t$, and one approximate constraint: $s \cdot s \approx_\epsilon t$.

$\mathbf{b} \leftarrow_{\text{exact}} (\mathbf{x} \geq \mathbf{y})$. First, we establish ϵ-tf-soundness. Suppose we have an ϵ-accurate assignment to the constraints.

From the first constraint, we have that b is either 0 or 1.

From the second constraint, we have that

$$\begin{aligned} b = 0 &\implies t = y - x \\ b = 1 &\implies t = x - y. \end{aligned}$$

Using the same reasoning as in proving ϵ-tf-soundness for the assert statement, the third constraint (the approximate one) implies $t \geq -\epsilon$.

By combining the above, we have that

$$\begin{aligned} x - y > \epsilon &\implies b = 1 \\ y - x > \epsilon &\implies b = 0. \end{aligned}$$

In the case where $|x - y| \leq \epsilon$, b can be either 0 or 1, which is consistent with the semantics of the approximate comparison.

By taking $\delta = \epsilon$, we have that b is consistent with the semantics of the approximate comparison, and thus ϵ-tf-soundness is established.

Second, we establish ϵ_C-tf-completeness. Consider all (x, y, b) restricted to η-multiples such that $b = 1$ if $x \geq y$ and $b = 0$ otherwise. Take t to be exactly $(2b - 1) \cdot (x - y)$ and s to be the nearest η-multiple to $\sqrt{t}$. Thus $s = \sqrt{t} + \hat{\eta}$ for some $\hat{\eta}$ where $|\hat{\eta}| \leq \eta/2$.

The traditional constraints are satisfied with zero error by the choice of t and the definition of b.

Because b is either 0 or 1, $t = (2b - 1) \cdot (x - y) = |x - y|$. Given this, the third constraint is satisfied with error at most

$$\begin{aligned} |s^2 - t| &= |(\sqrt{t} + \widehat{\eta_\pm})^2 - t| \\ &= |2\widehat{\eta_\pm}\sqrt{t} + \widehat{\eta_\pm}^2| \\ &\leq \eta\sqrt{t} + \eta^2/4 \\ &\leq \eta\sqrt{|x - y|} + \eta^2/4. \end{aligned}$$

This establishes ϵ_C-tf-completeness provided that $\epsilon_C \geq \eta\sqrt{|x - y|} + \eta^2/4$.

Note that both the tf-soundness and tf-completeness results here are stronger than those presented in the prior analysis of $b \leftarrow (x \geq y)$ where b is approximately Boolean and no traditional constraints are used.

Also, when exclusively using approximate constraints, it is extremely complex to confine b to be strictly Boolean, while with traditional constraints, doing so is straightforward. This example indicates the potential benefits of hybrid arithmetization, the full exploration of which we leave to future work.

F End-to-end correctness

We derive end-to-end soundness and completeness for Spain in the general case and for the parameters in our implementation. We do so by combining back-end correctness (Theorem 1, §B.3) and translation fidelity (§E.1). The statements below rely on δ and δ_{wg}, which are discussed in Appendix E.1.

Corollary 2. *Let F be a computation and (A, B, C) be an R1CS structure with m constraints and n variables. If (A, B, C) is an ϵ-*tf-sound *and $(\epsilon/\sqrt{m})$-*tf-complete *translation of F, then the protocol in Figure 8 satisfies the following two guarantees:*

1. *If out $\notin F_\delta(in)$, then the verifier accepts with probability at most*

$$\frac{\lfloor \log_{\min P} D_{max} \rfloor + \lfloor \log_{\min P} U \rfloor}{|P|} + \frac{4s + 2 + 2k}{\min P} + \kappa_{PC}.$$

2. *If out $\in F_{\delta'}(in)$, then the prover can always make an honest verifier accept.*

This corollary follows by straightforward combination of Theorem 1 with the definitions of tf-soundness and tf-completeness in Appendix E.1.

Corollary 3. *Let F be a computation from the list of benchmarks in Section 7, and (A, B, C) be an R1CS structure with $m \leq 2^{32}$ constraints and $n \leq 2^{32}$ variables. If (A, B, C) is an ϵ-*tf-sound *and $(\epsilon/\sqrt{m})$-*tf-complete *translation of F, then the protocol in Figure 8 with the parameters in Corollary 1 satisfies the following two guarantees:*

1. *If out $\notin F_\delta(in)$, then the verifier accepts with probability at most 2^{-40}.*
2. *If out $\in F_{\delta'}(in)$, then the prover can always make an honest verifier accept.*

The proof of this corollary follows mechanically from Corollaries 1 and 2.

G Artifact Appendix

Abstract

The purpose of this artifact is to disseminate our implementation of Spain, let others prove execution of numerical computations using Spain, and enable reproduction of the paper's experimental results.

Scope

The artifact comprises the implementation of Spain described in Section 6 as well as the experimental results and figures in Section 7.

Contents

The artifact includes implementations of Spain's front-ends and back-end (§6), implementations of the ZKLP-FE and Otti-FE baselines, scripts for reproducing experimental results, and ONNX files used for arithmetization.

The artifact is documented with a README that includes further detail and instructions for running numerical computations through Spain.

Hosting

The artifact is publicly available at `https://doi.org/10.5281/zenodo.20090527`.

Requirements

Docker is required to run this artifact. Memory requirements to reproduce results for benchmarks on the prover are as follows:

- For GPT-2, *seq*=2: at least 10 GB RAM
- For GPT-2, *seq*=32: at least 40 GB RAM
- For GPT-2, the largest *passes* with which we experiment, namely *passes*= 16: at least 270 GB RAM
- For ZKLP-FE: at least 92 GB RAM
- For all other benchmarks: experimental results can be reproduced with under 5 GB of RAM for the prover.