SPECTRA: Sensor Provenance and Execution Through Recursive Arguments

1st Marco Calipari
ESI
TUM
Munich, Germany
marco.calipari@tum.de

2nd Paul Timke Contreras
ESI
TUM
Munich, Germany
paul.timke@tum.de

3rd Mohammad Hamad
ESI
TUM
Munich, Germany
mohammad.hamad@tum.de

4th Sebastian Steinhorst
ESI
TUM
Munich, Germany
sebastian.steinhorst@tum.de

***Abstract*—Autonomous systems plan and act on sensor data, yet existing integrity measures protect that data only after it has been produced. Signature- or proof-based schemes trust every processing stage that runs prior to the protection, so an attacker with privileged access can manipulate the data before any cryptographic measure applies. We propose SPECTRA, a novel, general sensor model that moves the trust boundary as close as possible to the analog domain while attesting every stage in the digital domain. We formally define the SPECTRA sensor by leveraging the recursive arguments of Non-uniform Incrementally Verifiable Computation (NIVC), define when a stage is attested, with the resulting trust boundary, and compare it to the state of the art, identifying a trust gap, where current solutions protect a single stage, or trust the whole sensor until the data leaves it. Furthermore, we instantiate SPECTRA for the camera pipeline (SPECTRA-CAM), with Bayer-compliant step circuit implementation, and evaluate it on CPU and GPU, from an embedded NVIDIA Jetson to high-end server hardware, over a wide range of frame sizes. Finally, we expose the limitations of the state of the art with Pipeline Ordering Shuffling Attacks, showing that merely reordering the pipeline stages can mislead downstream classifiers while the output still carries a valid signature. In the context of this attack we show why the state of the art falls short, and motivate the need to rethink sensor attestation from a data-centric perspective. To the best of our knowledge, this is the first image pipeline proven with NIVC, optionally in zero-knowledge, and the first image-based proof system evaluated directly on commodity embedded hardware, the NVIDIA Jetson, which proves frames up to 4K in under 6 GiB of memory. With GPU acceleration, its proving time on utility frame sizes is on average $3.1\times$ lower than on its CPU, and on classifier-sized frames it stays within tens of seconds of server-grade hardware.**

***Index Terms*—NIVC, Sensor Security, Pipeline Integrity, Supernova**

1. Introduction

Sensor data integrity has gained traction over the last decade due to the rise of fully autonomous systems, which rely on this data as the basis for tasks such as planning and control. This gives malicious users a new attack surface, where the target is no longer the processing units or the algorithms, but the data that control them. This topic has been addressed through various approaches, ranging from forensics [1] to integrity protection via trusted manufacturer signatures [2], [3] and cryptographic measures utilizing Verifiable Computation (VC) [4]–[6]. However, these approaches suffer from many shortcomings. For example, forensics provides no guarantee when the verifier does not have direct access to the data, and cryptographic solutions (e.g., [2]–[6]) fully trust the sensor until its final processing stage. This leaves an open gap, where manipulations happen *before* any defensive measure, bypassing it.

Sensor architectures are composed of cascading stages [7]–[9] whose integrity directly governs the reliability of downstream tasks, including object classification, object detection, and odometry estimation [10]–[12]. Once attackers gain deep-level access to a sensor [13], [14], they can freely inject fabricated content [15], [16] or hijack its control flow [17]. We address this gap by rethinking attestation and, to the best of our knowledge, proposing the first abstract sensor model that allows an external verifier to verify *how* sensor data was produced, not only *where* it comes from. We define a stage as *attested* when the correctness of its execution and the provenance of its input can both be verified without trusting the component that runs it, and show that no existing approach attests any stage between the conversion and its signature. The model effectively pushes the trust boundary within the sensor rather than fully trusting it. We realize this model in Sensor Provenance and Execution Through Recursive Arguments (SPECTRA), which generates a single proof for the whole sensor pipeline, from the conversion stage onward, using Non-uniform Incrementally Verifiable Computation (NIVC) [18]. We implement SPECTRA for the camera sensor and evaluate it on both CPU and GPU across embedded Jetson devices and server-grade hardware.

In this paper, we make the following contributions:

- **SPECTRA sensor model.** We propose a general sensor model of the sensor as a composition of stages and push the trust boundary from the sen-

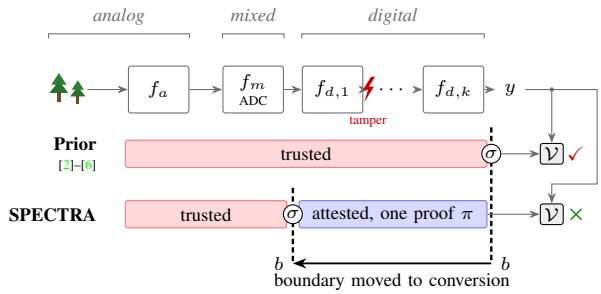


Figure 1. Overview of SPECTRA. Signing the output trusts every stage before σ; SPECTRA signs the readout and proves every digital stage, in order and with its parameters, in one proof π.

sor output to its conversion stage. On this model, we define when a stage is *attested* and show that no existing approach attests any stage between the conversion and its signature. SPECTRA realizes the model with NIVC, mapping each stage to a step circuit and anchoring the proof at the signed conversion output, so that an accepted proof implies that the output results from the prescribed stages, in the prescribed order and with the specified parameters, over the signed readout (Section 2, Section 3).

- **Camera instantiation.** SPECTRA-CAM, a four-stage camera pipeline implemented on SuperNova and proven row by row, so that it fits the memory of embedded devices, together with a cost model that selects the batch size under a memory budget (Section 4).
- **We evaluate SPECTRA-CAM**[1] on CPU and GPU, ranging from a Jetson Orin Nano to server-grade hardware, scaling up to 4K resolution. To the best of our knowledge, this makes SPECTRA-CAM the first image-based, optionally Zero Knowledge (ZK), proof system evaluated directly on embedded hardware (Section 5).
- **Pipeline Ordering Shuffling Attacks.** We show, on synthetic raw captures and three image classifiers, that rerouting correct stages, without injecting any content, suffices to mislead the classifier while the output still carries a valid signature (Section 5.3, Appendix B).

2. Problem Statement

Prior to our work, sensor data integrity solutions are based on cryptographic solutions such as Zero Knowledge Proof (ZKP), VC, and Identity and Access Management (IAM) or relying on additional hardware such as Trusted Execution Environment (TEE), and Trusted Platform Module (TPM). However, when it comes to the ZK, and VC solutions, the integrity protection covers the image manipulations *after* the data left the sensor from untrusted third parties. Hardware based solutions cover single elements of the sensors, and provide authentication.

1. https://anonymous.4open.science/r/SPECTRA-AE50/

TABLE 1. R1/R2 COMPLIANCE PER STAGE, LOWEST LEVEL PER COLUMN. CONV.: CONVERSION (f_m); APP.: APPLICATION. ● BOTH, ◑ ONE, ○ NEITHER.

Approach	Conv.	On-Chip	Ser/Des	DSP	App.
ZK-IMG [19]	○	○	○	◑	●
zk-REAL [20]	○	○	○	◑	●
VerITAS [6]	○	○	○	◑	●
Lin and Chang [21]	○	○	○	◑	○
JPSEC [22]	○	○	○	◑	○
YouProve [23]	○	○	○	◑	◑
TrustEYE.M4 [24]	○	◑	○	○	○
Khan et al. [25]	○	○	○	◑	○
Ko et al. [26]	○	○	○	○	●
Vronicle [27]	○	○	○	◑	◑
MIPI CSF [3]	○	◑	◑	○	○
ProvCam [28]	○	○	○	◑	○
Jang [29]	○	◑	◑	◑	○
VIMz [4]	○	○	○	◑	●
Della Monica et al. [5]	○	○	○	◑	●
Eva [30]	○	○	○	◑	●
Perazzo et al. [31]	○	○	○	◑	●
ZK-Disclosure [32]	○	○	○	○	○
zk-Cinema [33]	○	○	○	◑	●
ZK-JPEG [34]	○	○	○	◑	●
SPECTRA (ours)	◑	●	●	●	●*

*In combination with the methodologies above, or by introducing execution integrity to attest the application layer, out of Scope

Thus, the trust anchor is either atomic or disregards the steps required to create the data [7]. The processing steps before the integrity protection mechanism are inherently trusted, which leaves them open to attackers with full control over the sensor [15], [16]. Informally, we argue that the data should satisfy two requirements: **provenance (R1)**, *where* the data comes from, and **correctness (R2)**, *how* the data was produced. Without both R1 and R2, it would not be possible to verify the integrity of data produced by sensors in its entirety. To analyze the problem in more detail, we consider the camera as the target sensor, due to the plethora of available solutions and open standards. We consider the following macro stages of the camera pipeline (Table 1): Conversion (Conv.), responsible for the analog-to-digital conversion; On-Chip processing (On-Chip), where the first processing algorithms filter the digitized data (e.g., digital gain and dead-pixel correction); and Serialization (Ser/Des), where the preprocessed data are transmitted to the final stage, the Digital Signal Processor (DSP), for heavier processing such as demosaicing and white balancing [7]. We group the approaches in Table 1 by the point in the camera pipeline at which their integrity protection mechanism is applied. This allows us to highlight what we consider a still-open gap: *Can an external verifier verify how the data are produced (R2) and where they originate (R1)?* To get a clearer picture, we categorize state-of-the-art solutions according to these requirements: ○ when they satisfy neither

R1 nor R2, ◑ when they satisfy only one of them, and ● when they satisfy all of them.

It is possible to see that most mechanisms act at the bridge between the DSP and the application layer, where an authenticator is attached to the already processed image at or after the camera output [21]–[23], [25], [27], [28], [31], in some cases with trusted hardware enclosing the whole device or camera module [23], [27], [28], so that a verifier receiving it can bind the data to a signing key or a registered device. ZK and VC based mechanisms [4]–[6], [19], [20], [26], [30], [33], [34] add a layer on top of a signed original, proving that the published content results from allowed transformations performed by an untrusted editor. The remaining approaches anchor earlier, at the sensor and its link to the processor [3], [24], [29]. However, the processing stages that follow the anchor (Section 3) are either left unprotected or executed inside a trusted boundary, and in both cases an external verifier obtains no evidence that they were executed correctly. We identify the following gap:

Gap: Data integrity is protected either at a single point of the pipeline, with no evidence for an external verifier, or only at its output, where every stage that produced the data is trusted and not verified.

3. SPECTRA Sensor Model

In this section, we address the gap identified in the previous section by proposing the SPECTRA sensor model. We first provide an abstract model of a sensor architecture, defining the processing stage, the pipeline, when a stage is attested, and the threat model. Secondly, we derive the SPECTRA sensor model integrating the NIVC computational model. Finally, we show how the proof is generated and verified.

Processing Stages: We define a *processing stage* as a function

$$f\colon \mathcal{X} \times \mathcal{H} \to \mathcal{Y}, \tag{1}$$

where $\mathcal{X}$ and $\mathcal{Y}$ are the valid input and output domains given by the operational specifications of the stage, and $\mathcal{H}$ is the domain of its configuration parameters $\eta \in \mathcal{H}$. We call $(\mathcal{X}, \mathcal{Y})$ the *operational domain* of f.

We distinguish three types of stages: *Digital*, when both $\mathcal{X}, \mathcal{Y} \subseteq \{0,1\}^{\ell \times n}$, where ℓ is the length of the strings and n the number of dimensions. We denote a stage *Analog* when both $\mathcal{X}, \mathcal{Y} \subseteq \mathbb{R}^n$, and *Mixed* if one is digital, and the other is analog.

We denote by $\mathcal{M}$ the set of processing stages implemented by a sensor, and we split it by type: $\mathcal{M}_a$ groups the analog stages, $\mathcal{M}_m$ groups the mixed stages, and finally $\mathcal{M}_d$ groups the digital stages, so that $\mathcal{M} = \mathcal{M}_a \cup \mathcal{M}_m \cup \mathcal{M}_d$. For simplicity in the discussion, we will assume that the analog stages come first, followed by the conversion stage f_m, the only mixed one, and by the digital stages, so $\mathcal{M}_m = \{f_m\}$ and $\mathcal{M}_d = \{f_{d,1}, \ldots, f_{d,k}\}$. However, the rest of the derivations can be also extended to other types of configurations, depending on the specific sensor design. Additionally, for the sake of notation, since no analog stage is ever proven in this work, we write f_a for the composition of the stages in $\mathcal{M}_a$. A stage carries its subscript to its operational domain, its parameter domain and its parameters, e.g., $(\mathcal{X}_{d,i}, \mathcal{Y}_{d,i})$, $\mathcal{H}_{d,i}$ and $\eta_{d,i}$ for $f_{d,i}$.

Sensor Pipeline: Following [8] and [7], a *sensor pipeline* $\mathcal{S}$ applies the stages of $\mathcal{M}$ one after the other, so that the output of each stage is the input of the next. For this to be well-defined, consecutive stages must fit together, meaning that the output domain of every stage is contained in the input domain of the next, e.g., $\mathcal{Y}_{d,i} \subseteq \mathcal{X}_{d,i+1}$.

Given an input $x_a \in \mathcal{X}_a$ and the stage specific parameters $\eta = (\eta_a, \eta_m, \eta_d)$, with $\eta_d = (\eta_{d,1}, \ldots, \eta_{d,k})$, the pipeline computes

$$\begin{aligned} x_1 &= f_m\big(f_a(x_a, \eta_a), \eta_m\big), \\ x_{i+1} &= f_{d,i}(x_i, \eta_{d,i}), \qquad i = 1, \ldots, k, \end{aligned} \tag{2}$$

where x_1 is the digital readout, according to our sensor composition assumption, and x_i the input of $f_{d,i}$. The pipeline will have the following output:

$$y = x_{k+1} = f_{d,k}\big(\cdots f_{d,1}(x_1, \eta_{d,1}) \cdots, \eta_{d,k}\big) \in \mathcal{Y}_{d,k}.$$

Trust Anchor: We assume the existence of a collision-resistant hash function H, a knowledge-sound proof system, and a signature scheme $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vf})$ that is existentially unforgeable under chosen-message attacks [35]: $\mathsf{KeyGen}(1^\lambda)$, with λ the security parameter, outputs a signing key sk_σ and a public key pk_σ, $\mathsf{Sign}(\mathsf{sk}_\sigma, m)$ outputs a signature σ on a message m, and $\mathsf{Vf}(\mathsf{pk}_\sigma, m, \sigma)$ outputs 1 if σ is a valid signature on m, and 0 otherwise. A *trust anchor* is a trusted component at one point of the pipeline that holds a signing key sk_σ. Given the data x at that point, it emits the signature $\sigma = \mathsf{Sign}(\mathsf{sk}_\sigma, \mathsf{H}(x))$. A verifier holding pk_σ learns that x was present at that point, which is R1 for x, but learns neither how x was produced (R2) nor what is done with it afterwards.

Attested Stages: For $b \in \{1, \ldots, k+1\}$, we call the stages $f_{d,b}, \ldots, f_{d,k}$ *attested* if there exists a prover $\mathcal{P}$ that produces an evidence π satisfying both R1 and R2 for the sensor output y, and a verifier $\mathcal{V}$ such that, for every polynomial time attacker that controls these stages and outputs (y^*, π^*),

$$\Pr\left[\mathcal{V}(\eta, y^*, \pi^*) = 1 \wedge y^* \neq y\right] \leq \mathrm{negl}(\lambda). \tag{3}$$

Sensor Trust Boundary: We define the *sensor trust boundary* in Eq. 4, as the first digital stage from which all stages up to the output are attested

$$b = \min_{1 \leq b' \leq k+1} \{\, b' : f_{d,b'}, \ldots, f_{d,k} \text{ are attested} \,\}. \tag{4}$$

A trust anchor at the sensor output gives $b = k+1$. Since no proof takes an analog input, f_a is always trusted, and $b = 1$, right after the conversion stage, is the lowest boundary.

Threat Model: We consider a polynomial-time attacker with privileged access to the sensor and able to gain control

over it [13], [14]. The attacker controls every digital stage $f_{d,1}, \ldots, f_{d,k}$, the firmware and memory that run them, the buses between them, and, since attestation runs on the sensor, the prover that produces the evidence π. It can therefore read, delay and reroute the readout x_1 and every later x_i, feed a stage a value the pipeline did not produce [1], [15], [16], run the stages with other parameters, in another order, or not at all [17], and emit any pair (y^*, π^*) of its choice.

Attacker Goal: The goal of the attacker is to make the verifier accept an output y^* that the specified pipeline did not produce from the readout, and thereby alter the application that consumes y [36], such as, for example, the label of a classifier. The attacker knows the stages, their order, and their parameters, since it operates inside the pipeline and has only black-box access to the application.

The verifier trusts the analog stages f_a, and the conversion stage is the earliest component that can hold a key. SPECTRA places the trust anchor at its output, where it signs a digest of the readout x_1 and emits the signature σ. The anchor holds its signing key sk_σ in secure storage that the attacker cannot extract, which is the assumption that output-signing schemes make for the whole device [2], [3] and that in-sensor signing has shown realistic [24], [28]. Thus, the attacker cannot forge a signature over a digest the anchor has not emitted. The verifier is honest, knows the specified parameters η, and obtains the verification key material, including the public key pk_σ of the anchor, out of band. Attacks on the analog front end, such as optical injection, extraction of the anchor key, denial of service, and the replay of an earlier, honestly signed readout are out of scope; we return to replay and to parameters chosen at runtime in Section 6.

3.1. SPECTRA Sensor Model Derivation

We instantiate the model with NIVC [18], [37]. NIVC proves that a state z evolves through repeated applications of step functions from a set $\{F_i\}$, where at each step a selector φ chooses the function to apply. Its output is the tuple (z_0, z_N, Π_N): given the initial state z_0, the final state z_N and the proof Π_N, a verifier is convinced that z_N results from z_0 through N such steps.

From here on we write f_i for $f_{d,i}$ and η_i for $\eta_{d,i}$, and turn each digital stage f_i into a step function F_i by arithmetizing it into Rank One Constraint System (R1CS) constraints. The stages f_a and f_m are trusted, so they are not converted into step functions.

Intuition: Recalling the NIVC computational model, Eq. 5 shows the symmetry between the pipeline and the state vector. Both act recursively on the output of the previous step, and the stage index i becomes the index of the step function. The state vector z_j is updated by the step function F_i selected at step $j = i-1$. At each step, the prover extends the proof Π_j of the previous steps to a proof Π_{j+1} that also covers F_i, and passes it on together with the updated state.

At first sight, the state vector z_j is the counterpart of the data x_i, since both record how the data evolves along the pipeline. In NIVC, however, the data x_i carried between stages becomes the non-deterministic advice of the step function F_i, $\omega_j = x_i$, and what records the evolution of the computation is the state vector z_j.

$$\begin{aligned} x_{i+1} &= f_i(x_i, \eta_i), && \text{sensor pipeline,} \\ z_{j+1} &= F_{\varphi(z_j,\omega_j)}(z_j, \omega_j), && \text{NIVC [18].} \end{aligned} \tag{5}$$

In the following, we define the three components of the NIVC computational model and, from them, the SPECTRA sensor model.

Selector: In NIVC, at every step the selector φ takes the current state and the advice and returns the index of the next step function (Eq. 5). NIVC places no restriction on φ: if it depends on the advice, the prover chooses the next step function and can repeat, skip or reorder stages, so the proof would not bind the order of the pipeline. We therefore follow the machine-execution example of [18], where the selector reads a program counter stored in the state, and let it return this counter directly,

$$\varphi(z, \omega) = \mathsf{pc}, \tag{6}$$

and fix the schedule at setup: each step function F_i sets $\mathsf{pc} = i + 1$. Hence, after F_i only F_{i+1} can run, and the run ends at $\mathsf{pc} = k + 1$ after $N = k$ steps. Since the proof also covers the selection, the prover cannot skip, repeat, or reorder stages.

State Vector: The state vector z_j is shared by all step functions and updated at every step [18]. Besides the verification key and the number of steps N, the verifier receives only z_0, z_N and Π_N: the proof shows that the prover knows advice $\omega_0, \ldots, \omega_{N-1}$ and intermediate states $z_1, \ldots, z_{N-1}$ that lead from z_0 to z_N. Therefore, everything the verifier has to check against a known value must appear in z_0 or in z_N. This is why the proof alone does not provide R1. The data a stage consumes enters its step function as advice, so the proof guarantees that every stage was computed correctly on *some* input, which is R2, but does not cover its provenance. We therefore build the state vector following R1 and R2. Ideally, it would record the input, the output and the parameters η_d of every stage, so that the verifier could reconstruct the provenance chain of each stage and of the whole pipeline, on top of the correctness guaranteed by NIVC. However, every step function takes the whole state as input and returns it as output, so the size of the state adds to the cost of every step, and the data may be too large to be stored in it. In this case the state stores a digest of the data, computed with the collision-resistant hash function $\mathsf{H}\colon \{0,1\}^* \to \mathbb{F}$, in the same way as the RAM machine of [18] keeps in its state a commitment to the memory instead of the memory itself. The data still enters the step as advice, and the step function recomputes its digest and checks it against the one in the state. By collision resistance, the prover cannot pass this check with different data, so the digest binds the data as if the data itself were stored in the state. The state also holds the program counter pc read by the selector (Eq. 6). Because the program counter is part of

z_0 and of z_N, the stage from which the run starts and the stage at which it ends are visible to the verifier, so the prover can neither skip the first stages and start from an arbitrary one, nor stop before the last one. The state vector then has the same layout at every step,

$$\begin{gathered} z = (\mathsf{pc},\ D,\ \eta_d), \\ D = (d_0^{\mathsf{out}}, D_1, \ldots, D_k), \qquad D_i = (d_i^{\mathsf{in}}, d_i^{\mathsf{out}}), \\ \eta_d = (\eta_1, \ldots, \eta_k), \end{gathered} \tag{7}$$

where $\mathsf{pc} \in \{1, \ldots, k+1\}$ is the index of the next stage, η_d holds the parameters of the digital stages, and d_i^{in} and d_i^{out} are the digests of the input and of the output of stage i, so that an honest run gives $D_i = (\mathsf{H}(x_i), \mathsf{H}(x_{i+1}))$. The entry d_0^{out} is the digest of the output of the conversion stage, i.e., of the readout x_1, so that every digital stage, including the first, has a predecessor in the state.

The verifier checks that $z_0 = (1, d_1, 0, \ldots, 0, \eta_d)$, i.e., that the run starts from the first stage, with the specified parameters, with the digest $d_1 = \mathsf{H}(x_1)$ of the readout as d_0^{out} and with all the other digests set to zero. The link between consecutive stages, $d_i^{\mathsf{in}} = d_{i-1}^{\mathsf{out}}$ for $1 \le i \le k$, is checked by the step functions (Eq. 9), so the proof already guarantees that the digests in z_N form a chain, and the verifier is left with its two ends.

The first end is the readout x_1, from whose digest d_1 the chain starts. Since only the digital stages are proven, the proof cannot show where x_1 comes from, and its provenance rests on the trust anchor, which signs d_1: the verifier checks the signature on the d_0^{out} of z_0. The last end is the output y, which the verifier receives and hashes itself. The verifier thus accepts only if

$$d_k^{\mathsf{out}} = \mathsf{H}(y), \qquad \mathsf{pc} = k+1. \tag{8}$$

The first condition fixes the last end of the chain, and the second one, on the program counter of z_N, shows that the run did not stop before the last stage. By collision resistance and by the unforgeability of the signature, each stage then consumed exactly the output of its predecessor, the first one consumed the readout signed by the anchor and the last one produced y, which provides R1.

Step Functions: We define one step function per digital stage. Given the state $z = (\mathsf{pc}, D, \eta_d)$ of Eq. 7, the step function F_i of $f_i \in \mathcal{M}_d$ takes as advice the input of the stage, $\omega = x_i$, runs f_i on it with the parameters η_i read from the state, and writes in the state the digests of the input and of the result, and the index of the next stage:

$$\begin{gathered} F_i(z, \omega) = \big(i+1,\ D',\ \eta_d\big) \\ \text{if } \underbrace{\mathsf{pc} = i}_{\text{its turn}} \wedge \underbrace{\mathsf{H}(\omega) = d_{i-1}^{\mathsf{out}}}_{\text{link to its predecessor}}, \\ D_i' = \big(\mathsf{H}(\omega),\ \mathsf{H}(f_i(\omega, \eta_i))\big), \qquad D' = D \text{ elsewhere}, \end{gathered} \tag{9}$$

and $F_i(z, \omega) = \perp$ otherwise, where $\perp$ denotes an unsatisfiable step, i.e., a step with no satisfying witness, for which the prover cannot produce a proof. The first condition makes the schedule explicit: by Eq. 6, NIVC applies F_i exactly when $\mathsf{pc} = i$. The second one links the stage to its predecessor: the prover still chooses ω, but the step is satisfiable only if the digest of ω equals d_{i-1}^{out}, i.e., the digest recorded by F_{i-1} for $i > 1$ and, for $i = 1$, the digest d_1 of the readout, which z_0 holds and the anchor signs. In the implementation, the input of a stage does not fit in a single step, so we split each stage over several steps (Section 4.1), while the model treats each stage as one step, so $N = k$.

SPECTRA Sensor Model: Finally, we can define the SPECTRA sensor model. At every digital stage, the NIVC prover $\mathcal{P}_{\mathsf{NIVC}}$ extends the proof of the previous stages by one step [18]. Given the proving key pk, the statement (j, z_0, z_j), made of the number of steps proven so far, the initial state and the current state, the advice ω_j and the proof Π_j of that statement, it outputs the proof Π_{j+1} of $(j+1, z_0, z_{j+1})$. Since each stage is one step, f_i runs at step $j = i-1$ with advice $\omega_{i-1} = x_i$: it takes the state z_{i-1} and the proof Π_{i-1} and returns z_i and Π_i. The run starts from the initial state $z_0 = (1, d_1, 0, \ldots, 0, \eta_d)$, which the prover sets from the digest of the readout and the parameters of the sensor.

Definition 1 (Augmented Stage)**.** *We call* augmented stage f_i^**, with* $f_i \in \mathcal{M}_d$*, the sensor stage defined in Eq. 1 together with the corresponding step function, as defined in Eq. 9, and the proof* Π_i *of the stages up to* f_i*. It takes the data and the evidence* π_{i-1} *of the previous stage and returns those for the next one:*

$$\begin{aligned} f_i^*(x_i, \pi_{i-1}) &= (x_{i+1}, \pi_i), \\ \pi_i &= (\sigma, z_0, z_i, \Pi_i), \\ x_{i+1} &= f_i(x_i, \eta_i), \\ z_i &= F_i(z_{i-1}, x_i), \\ \Pi_i &= \mathcal{P}_{\mathsf{NIVC}}\big(\mathsf{pk},\ (i-1, z_0, z_{i-1}),\ x_i,\ \Pi_{i-1}\big), \end{aligned} \tag{10}$$

where the signature σ *of the trust anchor and the initial state* z_0 *are passed on unchanged,* $\mathcal{P}_{\mathsf{NIVC}}$ *is the prover of the NIVC scheme [18] and* pk *is the proving key that the scheme derives at setup, together with the verification key* vk*, from the selector* φ *and the step functions* $F_1, \ldots, F_k$*.*

Definition 2 (SPECTRA Sensor Model)**.** *The* SPECTRA sensor model *is the sensor pipeline of Eq. 2 whose conversion stage is followed by the trust anchor and whose digital stages are augmented (Figure 2):*

$$\begin{aligned} (y, \pi) &= f_k^*\big(\cdots f_1^*(x_1, \pi_0)\cdots\big), \\ \pi_0 &= (\sigma, z_0, z_0, \Pi_0), \\ \sigma &= \mathsf{Sign}\big(\mathsf{sk}_\sigma, \mathsf{H}(x_1)\big), \\ x_1 &= f_m\big(f_a(x_a, \eta_a), \eta_m\big), \end{aligned} \tag{11}$$

where σ *is the signature of the trust anchor on the digest of the readout,* Π_0 *is the proof of the base case, which is trivial since no step has been proven yet [18], and* $\pi = \pi_k = (\sigma, z_0, z_N, \Pi_N)$ *is the evidence of Eq. 3, since* $N = k$*.*

The analog stages and the conversion stage run as in Eq. 2. The trust anchor signs the digest $d_1 = \mathsf{H}(x_1)$ of the readout, and its signature σ travels to the verifier as the first element of the evidence, and the digital stages cannot forge it, and altering or dropping it only makes the verifier reject.

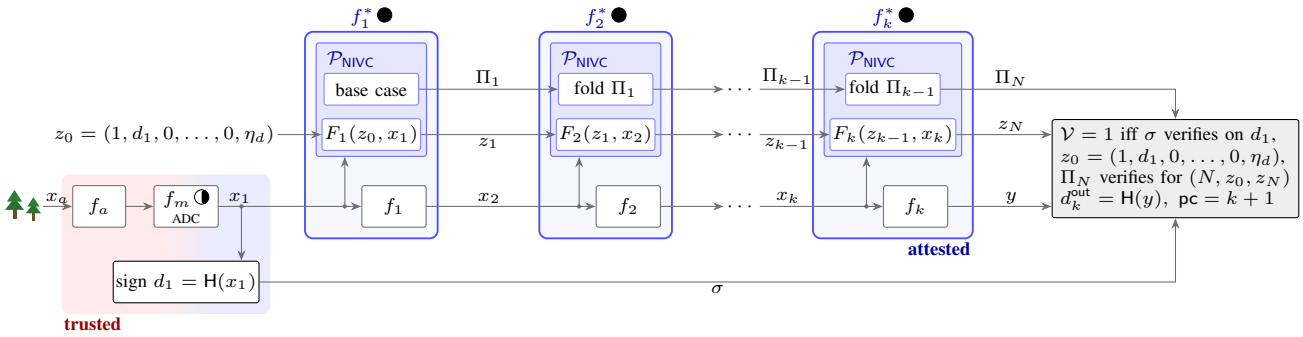


Figure 2. The SPECTRA sensor model (Definition 2); marks as in Table 1.

The output y is the same as in Eq. 2, so SPECTRA does not alter what the sensor delivers. It adds the evidence π, i.e., the signature of the anchor and the output (z_0, z_N, Π_N) of NIVC, where Π_N proves the statement (N, z_0, z_N). The verifier accepts if σ verifies on d_1, z_0 holds d_1 and the specified parameters, Π_N verifies and z_N satisfies Eq. 8, as formalized in Eq. 12.

3.2. Security Analysis

We now show that the verifier of SPECTRA attests the digital stages, i.e., that it satisfies Eq. 3 with $b = 1$. Given the parameters η, the output y and the evidence $\pi = (\sigma, z_0, z_N, \Pi_N)$, the verifier accepts if

$$\begin{aligned} \mathcal{V}(\eta, y, \pi) = 1 \iff \mathsf{Vf}\big(\mathsf{pk}_\sigma, d_1, \sigma\big) = 1 \\ \wedge\ z_0 = (1, d_1, 0, \ldots, 0, \eta_d) \\ \wedge\ \mathcal{V}_{\mathsf{NIVC}}\big(\mathsf{vk}, (N, z_0, z_N), \Pi_N\big) = 1 \\ \wedge\ z_N \text{ satisfies Eq. 8,} \end{aligned} \tag{12}$$

where $N = k$, vk is the verification key matching pk and pk_σ the public key of the anchor. The verifier takes $d_1 = d_0^{\mathsf{out}}$ from z_0 as the message and checks σ on it natively. The signature never enters a circuit, since d_1 is public, so SPECTRA needs no signature verification gadget, and the anchor only computes H over the readout and signs the result. We return to this choice in Section 6.

Completeness: The honest sensor of Eq. 11 starts from $z_0 = (1, d_1, 0, \ldots, 0, \eta_d)$, with $d_1 = \mathsf{H}(x_1)$, and runs $F_1, \ldots, F_k$ in turn, each on the output of its predecessor, so every step satisfies Eq. 9 and $D_i = (\mathsf{H}(x_i), \mathsf{H}(x_{i+1}))$ for $i = 1, \ldots, k$. The conditions of Eq. 8 then hold, with $d_k^{\mathsf{out}} = \mathsf{H}(y)$ and $\mathsf{pc} = k + 1$, set by F_k; by correctness of the signature scheme σ verifies on d_1, and by completeness of NIVC the proof verifies.

Soundness: Soundness rests on two facts: the program counter admits a single schedule (Lemma 1), and the digests in z_N contain a single chain of data (Lemma 2).

Lemma 1 (Schedule)**.** *Let z_0 be as in Eq. 12. A run of $N = k$ steps in which no step is $\perp$ executes $F_1, \ldots, F_k$ exactly once each, in this order, and ends with $\mathsf{pc} = k + 1$.*

Proof sketch. The program counter of z_0 is 1 and the selector returns the program counter (Eq. 6), so the first step runs F_1. Each F_i sets the program counter to $i + 1$ (Eq. 9), so the step after F_i runs F_{i+1}. After k steps the program counter is $k + 1$. □

Lemma 2 (Chain)**.** *Let the NIVC scheme be knowledge sound, H collision resistant and the signature scheme existentially unforgeable. If the verifier of Eq. 12 accepts (y^*, π^*), then, except with probability $\mathrm{negl}(\lambda)$, F_1 consumed the readout x_1 signed by the anchor, each F_{i+1}, for $1 \le i < k$, consumed the output of f_i on the input of F_i, computed with the specified η_i, and y^* is the output of f_k on the input of F_k, computed with the specified η_k.*

Proof sketch. By knowledge soundness, the prover knows an advice for every step of the accepted run [18], and by Lemma 1 these are the inputs of $F_1, \ldots, F_k$, in this order. Each F_i writes in D_i the digests of its input and of its output, which the other steps only copy, and reads η_i from the state, which every step copies from the z_0 checked by the verifier. The second condition of Eq. 9 makes the digest of the input of F_1 equal to the signed d_1 of z_0 and the digest of the input of F_{i+1} equal to the digest of the output of f_i, and Eq. 8 makes the digest of the output of f_k equal to $\mathsf{H}(y^*)$. By unforgeability, d_1 is the digest $\mathsf{H}(x_1)$ that the anchor signed, and by collision resistance equal digests mean equal data. □

Together, the two lemmas give $y^* = f_k\big(\cdots f_1(x_1, \eta_1) \cdots, \eta_k\big) = y$ for every accepted output, except with probability $\mathrm{negl}(\lambda)$, so Eq. 3 holds, the stages $f_1, \ldots, f_k$ are attested and, by Eq. 4, $b = 1$.

Corollary 1 (Rejection of injection)**.** *Let the attacker feed some digital stage f_i a value $x \neq x_i$, such that the resulting output y^* differs from y. No polynomial time attacker makes the verifier of Eq. 12 accept y^*, except with probability $\mathrm{negl}(\lambda)$.*

Proof. By Lemma 2, every stage of an accepted run consumed the output of its predecessor, starting from the readout x_1 signed by the anchor, so no stage consumed x. □

Corollary 2 (Rejection of reordering)**.** *Let g be any composition of the digital stages other than $f_k \circ \cdots \circ f_1$, including one that skips, repeats or reorders stages, such that $g(x_1) \neq f_k\big(\cdots f_1(x_1, \eta_1) \cdots, \eta_k\big)$. No polynomial time attacker makes the verifier of Eq. 12 accept $g(x_1)$, except with probability* $\mathrm{negl}(\lambda)$.

Proof. By Lemma 1, a run that starts at a later stage contradicts the program counter of z_0, one that stops early ends with $\mathsf{pc} \neq k+1$ and leaves d_k^{out} at zero, which fails both conditions of Eq. 8, and one that repeats or reorders a stage reaches a step whose function is not the one φ selects, which has no satisfying witness. By Lemma 2, the accepted output is $f_k\big(\cdots f_1(x_1, \eta_1) \cdots, \eta_k\big) \neq g(x_1)$. □

Attacks outside these guarantees are discussed in Section 6.

4. The Camera as a SPECTRA Sensor

So far, SPECTRA has been defined generically, where any sensor whose output results from a conversion stage followed by a chain of digital stages, be it a LiDAR, a Radar, or a camera, fits the model of Eq. 2. We now make the model concrete by instantiating SPECTRA-CAM, a camera following the stage categorization of Table 1.

We consider a representative pipeline composed of two algorithms for the On-Chip processing stage (On-Chip column in Table 1) and two algorithms for DSP (DSP column in Table 1). The Ser/Des stage is treated as an identity, which only has to preserve the integrity of the data in transit. The conversion stage and the anchor are not in the scope, and we assume a valid signed digest d_1, and the verifier holds the key material for the SPECTRA sensor verification.

Using the definition from Eq. 2, the conversion stage f_m is the Analog to Digital Converter (ADC) and the digital stages $f_1, \ldots, f_4$ are Defective Pixel Correction (DPC), digital gain (DGAIN), demosaicing (DEM) and white balancing (WBAL), so $k = 4$. Since no proof reaches before the conversion ($b \geq 1$), f_a is the identity for the sake of simplicity.

In order to retain R1 and R2 across all the stages, we fix the ordering of the stages as in Eq. 13, and for each of the stages we also fix its parameters η_i. The order is a design choice, however, needs to be fixed.

$$\begin{aligned} y &= \underbrace{f_4}_{\text{WBAL}}\Big(\underbrace{f_3}_{\text{DEM}}\Big(\underbrace{f_2}_{\text{DGAIN}}\big(\underbrace{f_1}_{\text{DPC}}(x_1, \eta_1), \eta_2\big), \eta_3\Big), \eta_4\Big), \\ x_1 &= \underbrace{f_m}_{\text{ADC}}(x_a, \eta_m). \end{aligned} \tag{13}$$

Anchor, Prover and Verifier: SPECTRA-CAM places the trust anchor within the ADC, where it binds the digest d_1 of the readout, and runs every digital stage as an augmented stage (Eq. 10). The camera then outputs the image y of Eq. 13 together with the evidence $\pi = (\sigma, z_0, z_N, \Pi_N)$, which is Eq. 11 with $k = 4$. In the model, the proof has one step per digital stage, with $\mathsf{pc} \in \{1, \ldots, 5\}$ and the state $z = (\mathsf{pc}, d_0^{\mathsf{out}}, D_1, \ldots, D_4, \eta_d)$ of Eq. 7.

4.1. Step Functions Implementation

Every stage of the SPECTRA-CAM pipeline (WBAL, DEM, DGAIN, DPC) is arithmetized as an R1CS. This implementation is meant to describe a representative camera pipeline as a proof of concept; for this reason, we make some simplifications, such as integer arithmetic only, in order to keep clarity in the discussion. However, with the necessary modification, it is possible to extend the step circuits to cover additional corner cases. Other sensors require arithmetizing their own stage algorithms.

Since images can reach significant sizes, sometimes on the order of millions of pixels, SPECTRA-CAM uses the folding properties of NIVC and processes frames not in their entirety, but in row-major order. In other words, the non-deterministic advice ω consists of the *rows* of each image. We call the computation of one row, or band, a *substep*. Then, each step circuit is invoked until its stage has consumed the whole frame. The state carries a substep counter ρ and a program counter pc_z (Eq. 21): after the last substep of a stage, which the flag *last* marks, ρ goes back to zero and the program counter is updated. Further discussion is provided in the Row Batching paragraph below.

The step circuits share *gadget* sub-circuits for common operations: BitDecompose for bit decomposition *and* range checking, SatMul for the saturating multiplication of two bytes, AbsorbRow and AbsorbBits for packing a row into field elements, with the lossless pixel compression of VIMz [4], and absorbing it into a running digest, and AdvanceStage for keeping track of the row that is processed and updating the program counter when the stage is finished. A detailed description of the gadgets can be found in Appendix A.

Since we process frames row-by-row, every digest within the state vector is thus a running digest $d' = \mathsf{H}(d, e)$, extended by each row in order. The handover is the only point where the program counter moves, and since the number of substeps of a stage is a constant of the shape, a stage cannot hand over early. We keep the notation of the model: x_i is the input of the stage f_i and x_{i+1} its output, with $x_5 = y$. W and H are the width and the height of the frame, and $x_i[t, j]$ is the pixel of x_i in row t and column j, with $0 \leq t < H$ and $0 \leq j < W$. Bold symbols denote rows of W field elements: in a substep, $\mathbf{p} = (p_0, \ldots, p_{W-1})$ is a row of the input of the stage and $\mathbf{o} = (o_0, \ldots, o_{W-1})$ a row of its output. In the algorithms, **assert** adds an R1CS constraint; every other assignment is either a linear combination of existing variables or a *hint*, computed natively by the prover and bound by the constraints that follow. All four step circuits share the same pattern (Algorithms 1–4): they range check every input pixel to 8 bits (Algorithm 5), constrain the output of their stage, absorb the input and output rows into d_i^{in} and d_i^{out} and, on the last substep and for $i > 1$, require $d_i^{\mathsf{in}} = d_{i-1}^{\mathsf{out}}$, the link of Eq. 9. Outputs are never range checked, since each is pinned to a range-checked value or to the constant 255.

In the following, we present a brief description of the arithmetized algorithm, motivating the simplifications we

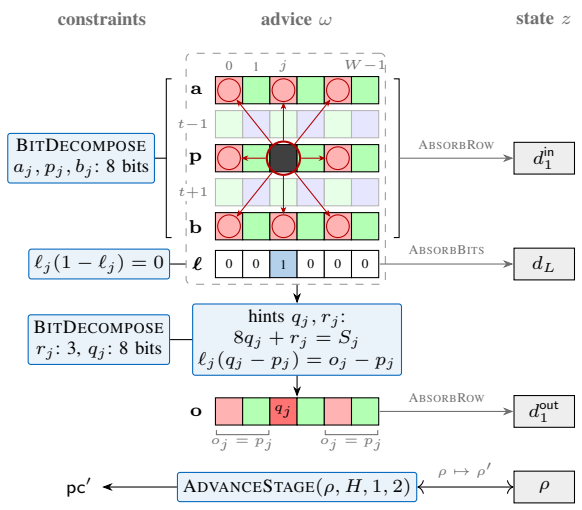


Figure 3. One DPC substep (Algorithm 1): defective pixel p_j (circled) and its eight same-color neighbors; faded rows are not part of the advice.

made.

Defective Pixel Correction:

We apply a static DPC, where the defective pixels are set by the manufacturer in a defect mask, so we perform only their correction, not their *detection*. We assume that there are no defective pixels at the border of a frame, and that each defective pixel is surrounded by healthy pixels. Following the reference algorithm of InfiniteISP [38], the DPC substitutes each defective pixel with the mean of its nearest pixels of the same color, which for raw Bayer images [39] lie at an offset of 2, so the grid spans the 5×5 window around the pixel. The defect mask has the same size as the image, is treated row-wise as well, and is stored in the state as a running digest d_L.

Given the readout x_1 and the defect mask $\mathcal{L}$, whose digest is the parameter η_1 and where $\mathcal{L}[t,j] = 1$ marks the pixel in row t and column j as defective, the output is

$$x_2[t,j] = \begin{cases} \lfloor S[t,j]/8 \rfloor & \mathcal{L}[t,j] = 1, \\ x_1[t,j] & \text{otherwise}, \end{cases} \tag{14}$$

where $S[t,j] = \sum_{(\delta_t,\delta_j)\neq(0,0)} x_1[t+\delta_t,\, j+\delta_j]$, with $\delta_t, \delta_j \in \{-2, 0, 2\}$, is the sum of the eight same-color neighbors.

In each substep, the step function F_1 processes one row t, using as advice the current row $\mathbf{p}$, the rows $\mathbf{a}$ and $\mathbf{b}$ two above and two below it, i.e., the rows $t-2$ and $t+2$ of x_1, the output row $\mathbf{o}$ of the corrected image and the mask row $\boldsymbol{\ell}$ (Algorithm 1, Figure 3). Since the row is fixed within a substep, we drop t and keep only the column index j, so we can write $p_j = x_1[t,j]$, $o_j = x_2[t,j]$, $\ell_j = \mathcal{L}[t,j]$ and $S_j = S[t,j]$. For $t < 2$ the row $t-2$ does not exist, and for $t \geq H-2$ the row $t+2$ does not; the current row then takes its place. Since no border pixel is defective, such a row never enters a correction.

We absorb all three input rows into d_1^{in}, and the anchor computes d_1 in the same three-row layout, so that the link $d_1^{\mathsf{in}} = d_1$ of Eq. 9 rejects forged neighbors. Likewise, $\boldsymbol{\ell}$ is

Algorithm 1 Defective pixel correction step circuit (one substep)

Require: State z; advice: input rows $\mathbf{a}, \mathbf{p}, \mathbf{b}$ (above, current, below), output row $\mathbf{o}$, mask row $\boldsymbol{\ell}$
Ensure: z' with $d_1^{\mathsf{in}\prime}, d_1^{\mathsf{out}\prime}, d_L', \rho', \mathsf{pc}_z'$ updated; pc'

for $j \leftarrow 0$ **to** $W-1$ **do**
  BitDecompose$(v, 8)$ **for** $v \in \{a_j, p_j, b_j\}$
  assert $\ell_j \cdot (1-\ell_j) = 0$ % boolean flag
end for
assert $o_j = p_j$ **for** $j \in \{0, 1, W-2, W-1\}$ % border
for $j \leftarrow 2$ **to** $W-3$ **do**
  $S_j \leftarrow \sum_{\delta\in\{-2,0,2\}} (a_{j+\delta} + b_{j+\delta}) + p_{j-2} + p_{j+2}$
  $q_j \leftarrow \lfloor S_j/8 \rfloor$; $r_j \leftarrow S_j \bmod 8$ % hints
  assert $8q_j + r_j = S_j$
  BitDecompose$(r_j, 3)$; BitDecompose$(q_j, 8)$
  assert $\ell_j \cdot (q_j - p_j) = o_j - p_j$ % $o_j = \ell_j \,?\, q_j : p_j$
end for
$d_1^{\mathsf{in}\prime} \leftarrow$ AbsorbRow$(d_1^{\mathsf{in}}, \mathbf{a}, \mathbf{p}, \mathbf{b})$ % layout of d_1
$d_1^{\mathsf{out}\prime} \leftarrow$ AbsorbRow$(d_1^{\mathsf{out}}, \mathbf{o})$
$d_L' \leftarrow$ AbsorbBits$(d_L, \boldsymbol{\ell})$ % binds the mask row
$(\mathsf{pc}', \rho', last) \leftarrow$ AdvanceStage$(\rho, H, 1, 2)$
assert $\mathsf{pc}_z = 1$; $\mathsf{pc}_z' \leftarrow 1 + last$ % its turn
assert $last \cdot (d_L' - \eta_1) = 0$ % specified mask
return (z', pc')

absorbed into d_L, which the last substep compares with the digest η_1 of the specified mask, so that the prover cannot mark a healthy pixel as defective.

To convert Eq. 14 into R1CS, we constrain every flag ℓ_j to be boolean. The average is checked in circuit using the standard Euclidean division method, providing the quotient q_j and the remainder r_j of the neighbor sum S_j as hints to save constraints. To prevent a dishonest prover from wrapping around the field, r_j and q_j are also range checked. The two columns at each side of the row have no such neighbors, so there the output is constrained to equal the input; without this constraint, these outputs would be free advice. Finally, a single select constraint pins the output to q_j for a defective pixel and to p_j otherwise. Eq. 15 shows the DPC R1CS constraints.

$$\begin{cases} \ell_j \cdot (1-\ell_j) = 0, & 0 \leq j < W, \\ 8q_j + r_j = S_j, & 2 \leq j \leq W-3, \\ r_j < 2^3,\ q_j < 2^8, & 2 \leq j \leq W-3, \\ \ell_j \cdot (q_j - p_j) = o_j - p_j, & 2 \leq j \leq W-3, \\ o_j = p_j, & \text{otherwise}. \end{cases} \tag{15}$$

Digital Gain:

The digital gain is a saturating scalar multiplication. Given its input x_2 and the gain $\eta_2 = g$, the output is, pixel-wise,

$$x_3[t,j] = \min\left(g \cdot x_2[t,j],\, 255\right). \tag{16}$$

The step function F_2 takes as advice the input row $\mathbf{p}$ and the output row $\mathbf{o}$, and reads g from the state, which binds the gain to z_0 (Algorithm 2, Figure 4). To convert Eq. 16 into R1CS, we keep the bits $u_{j,0}, \ldots, u_{j,7}$ from the range check of every input pixel p_j, and range check the gain g in the same way. In the general case, the saturated

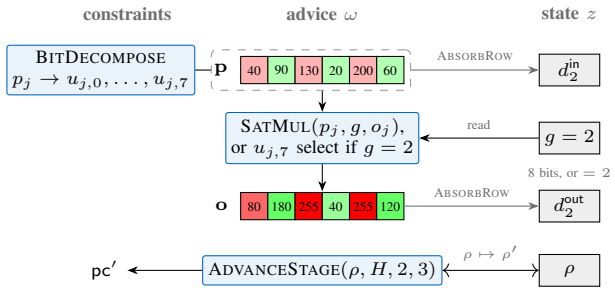


Figure 4. One digital gain substep (Algorithm 2) with $g = 2$: the select of Eq. 17 saturates 130 and 200 to 255.

Algorithm 2 Digital gain step circuit (one substep)

Require: State z; advice: input row $\mathbf{p}$, output row $\mathbf{o}$
Ensure: z' with $d_2^{\text{in}\prime}, d_2^{\text{out}\prime}, \rho', \text{pc}'_z$ updated; pc'

if $g = 2$ **then**
  assert $g = 2$ % binds the shape to z
else
  BitDecompose$(g, 8)$ % gain is a byte
end if
for $j \leftarrow 0$ **to** $W - 1$ **do**
  $(u_{j,0}, \ldots, u_{j,7}) \leftarrow$ BitDecompose$(p_j, 8)$
  if $g = 2$ **then**
    assert $u_{j,7} \cdot (255 - 2p_j) = o_j - 2p_j$
  else
    SatMul(p_j, g, o_j) % $o_j = \min(g\,p_j, 255)$
  end if
end for
$d_2^{\text{in}\prime} \leftarrow$ AbsorbRow$(d_2^{\text{in}}, \mathbf{p})$
$d_2^{\text{out}\prime} \leftarrow$ AbsorbRow$(d_2^{\text{out}}, \mathbf{o})$
$(\text{pc}', \rho', last) \leftarrow$ AdvanceStage$(\rho, H, 2, 3)$
assert $\text{pc}_z = 2$; $\text{pc}'_z \leftarrow 2 + last$ % its turn
assert $last \cdot (d_2^{\text{in}\prime} - d_1^{\text{out}}) = 0$ % link to DPC
return (z', pc')

product is computed by the saturating multiplication gadget (Algorithm 6). If the gain is 2, however, we avoid its additional constraints: $2p_j$ exceeds 255 exactly when the most significant bit $u_{j,7}$ of p_j is set, so this bit, already available from the decomposition of the input, serves as the overflow flag. The constant 2 is then part of the circuit, which constrains the g of the state to equal it instead of range checking it. In both cases, a single select constraint pins the output pixel:

$$\begin{cases} ov_j \cdot (255 - low_j) = o_j - low_j, & g \neq 2, \\ u_{j,7} \cdot (255 - 2p_j) = o_j - 2p_j, & g = 2, \end{cases} \tag{17}$$

where low_j is the low byte of the product $g \cdot p_j$ and ov_j its overflow flag, both computed by the gadget.

Demosaicing:

We implement the demosaicing stage by performing a nearest neighbor demosaicing assuming the *RGGB* Bayer pattern. For different patterns, such as *BGGR*, it is necessary to adjust the algorithm accordingly. Given its input x_3, the demosaicing algorithm creates the RGB triplet of every pixel. It has no parameter, so η_3 is empty.

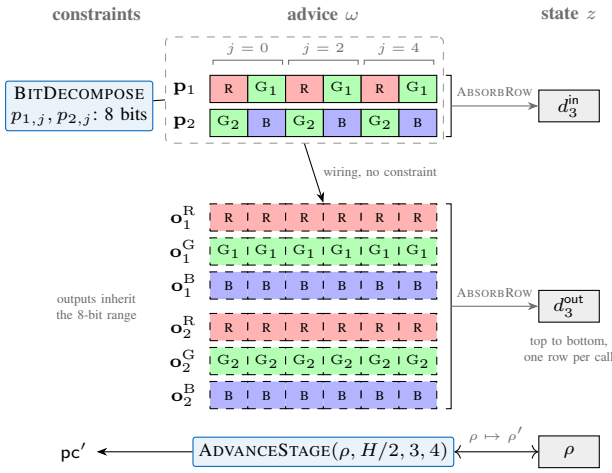


Figure 5. One demosaicing substep (Algorithm 3) on three super pixels. Output rows (dashed) are input samples wired as in Eq. 18.

Unlike the other stages, the output lives in a higher dimensional space than the input, since every Bayer sample becomes a triplet, $x_4[t,j] = \big(x_4^{\text{R}}[t,j], x_4^{\text{G}}[t,j], x_4^{\text{B}}[t,j]\big)$. The pattern defines a super pixel as a 2×2 grid of single-color samples, and every pixel of the super pixel receives the triplet built from it (Figure 5). For every even row t and every even column j, the super pixel covering the rows $t, t+1$ and the columns $j, j+1$ assigns the RGB triplet as in Eq. 18:

$$\begin{aligned} x_4[t,j'] &= \big(\text{R},\ \text{G}_1,\ \text{B}\big), \\ x_4[t+1,j'] &= \big(\text{R},\ \text{G}_2,\ \text{B}\big), \end{aligned} \qquad j' \in \{j, j+1\}, \tag{18}$$

with $\text{R} = x_3[t,j]$, $\text{G}_1 = x_3[t,j+1]$, $\text{G}_2 = x_3[t+1,j]$ and $\text{B} = x_3[t+1,j+1]$.

The step function F_3 takes as advice two consecutive Bayer rows, $\mathbf{p}_1$ and $\mathbf{p}_2$, i.e., the rows t and $t+1$ of x_3, so working on pairs of rows lets the circuit process whole super pixels while keeping the row as the unit of the digest (Algorithm 3, Figure 5). We write $p_{1,j}$ and $p_{2,j}$ for their pixels in column j, and $\mathbf{o}_1^C$ and $\mathbf{o}_2^C$, with $C \in \{\text{R}, \text{G}, \text{B}\}$, for the channel rows of the two output rows. As for the output, in the nearest neighbor demosaicing every output pixel is composed of elements of the corresponding super pixel (Eq. 18). The circuit therefore does not take the output rows as advice, as the other stages do, but assembles them itself from the input samples. Since the prover does not supply the output, it cannot alter it, and the stage needs no constraint to check it.

The two Bayer rows are absorbed into d_3^{in} in order, which is the layout of d_2^{out}, and the output into d_3^{out} row by row, the R, G and B rows of the first output row and then of the second, each channel row packed on its own.

White Balancing:

White balancing compensates the color cast of the illuminant by scaling the red and blue channels while leaving green untouched in an RGB image. It is a per-channel digital gain with the semantics of Eq. 16: given its input x_4 and

Algorithm 3 Demosaicing step circuit (one substep)

Require: State z; advice: input rows $\mathbf{p}_1, \mathbf{p}_2$; W even
Ensure: z' with $d_3^{\mathsf{in}\prime}, d_3^{\mathsf{out}\prime}, \rho', \mathsf{pc}'_z$ updated; pc'

for $j \leftarrow 0$ **to** $W-1$ **do**
  BitDecompose$(v, 8)$ **for** $v \in \{p_{1,j}, p_{2,j}\}$
end for
for $j \leftarrow 0, 2, \ldots, W-2$ **do** % wiring only
  for all $j' \in \{j, j+1\}$ **do**
    $(o^{\mathrm{R}}_{1,j'}, o^{\mathrm{G}}_{1,j'}, o^{\mathrm{B}}_{1,j'}) \leftarrow (p_{1,j}, p_{1,j+1}, p_{2,j+1})$
    $(o^{\mathrm{R}}_{2,j'}, o^{\mathrm{G}}_{2,j'}, o^{\mathrm{B}}_{2,j'}) \leftarrow (p_{1,j}, p_{2,j}, p_{2,j+1})$
  end for
end for
$d_3^{\mathsf{in}\prime} \leftarrow$ AbsorbRow$(d_3^{\mathsf{in}}, \mathbf{p}_1, \mathbf{p}_2)$
$d_3^{\mathsf{out}\prime} \leftarrow$ AbsorbRow$(d_3^{\mathsf{out}}, \mathbf{o}_1^{\mathrm{R}}, \mathbf{o}_1^{\mathrm{G}}, \mathbf{o}_1^{\mathrm{B}}, \mathbf{o}_2^{\mathrm{R}}, \mathbf{o}_2^{\mathrm{G}}, \mathbf{o}_2^{\mathrm{B}})$
$(\mathsf{pc}', \rho', last) \leftarrow$ AdvanceStage$(\rho, H/2, 3, 4)$
assert $\mathsf{pc}_z = 3$; $\mathsf{pc}'_z \leftarrow 3 + last$ % its turn
assert $last \cdot (d_3^{\mathsf{in}\prime} - d_2^{\mathsf{out}}) = 0$ % link to DGAIN
return (z', pc')

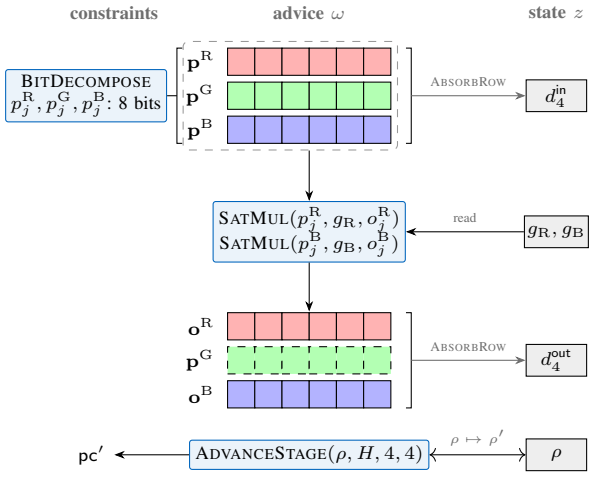


Figure 6. One white balancing substep (Algorithm 4). Green (dashed) passes through at no cost.

the gains $\eta_4 = (g_{\mathrm{R}}, g_{\mathrm{B}})$, the output $y = x_5$ of the pipeline is, pixel-wise,

$$\begin{aligned} y^{\mathrm{R}}[t,j] &= \min\left(g_{\mathrm{R}} \cdot x_4^{\mathrm{R}}[t,j],\, 255\right), \\ y^{\mathrm{G}}[t,j] &= x_4^{\mathrm{G}}[t,j], \\ y^{\mathrm{B}}[t,j] &= \min\left(g_{\mathrm{B}} \cdot x_4^{\mathrm{B}}[t,j],\, 255\right). \end{aligned} \tag{19}$$

The step function F_4 reads the gains $g_{\mathrm{R}}, g_{\mathrm{B}}$ from the state, and takes as advice the input rows $\mathbf{p}^{\mathrm{R}}, \mathbf{p}^{\mathrm{G}}, \mathbf{p}^{\mathrm{B}}$ and the output rows $\mathbf{o}^{\mathrm{R}}, \mathbf{o}^{\mathrm{B}}$ (Algorithm 4, Figure 6). We range check both gains to 8 bits and apply the saturating multiplication gadget (Algorithm 6) to every red and blue pixel, which gives, as for the digital gain, the constraints of Eq. 20:

$$\begin{cases} ov_j^{\mathrm{R}} \cdot (255 - low_j^{\mathrm{R}}) = o_j^{\mathrm{R}} - low_j^{\mathrm{R}}, \\ ov_j^{\mathrm{B}} \cdot (255 - low_j^{\mathrm{B}}) = o_j^{\mathrm{B}} - low_j^{\mathrm{B}}. \end{cases} \tag{20}$$

As in demosaicing (Algorithm 3), the green channel is directly wired and costs no additional constraint. The absorb order is $(\mathbf{p}^{\mathrm{R}}, \mathbf{p}^{\mathrm{G}}, \mathbf{p}^{\mathrm{B}})$ into d_4^{in}, which matches the output

Algorithm 4 White balancing step circuit (one substep)

Require: State z; advice: input rows $\mathbf{p}^{\mathrm{R}}, \mathbf{p}^{\mathrm{G}}, \mathbf{p}^{\mathrm{B}}$, output rows $\mathbf{o}^{\mathrm{R}}, \mathbf{o}^{\mathrm{B}}$
Ensure: z' with $d_4^{\mathsf{in}\prime}, d_4^{\mathsf{out}\prime}, \rho', \mathsf{pc}'_z$ updated; pc'

BitDecompose$(v, 8)$ **for** $v \in \{g_{\mathrm{R}}, g_{\mathrm{B}}\}$ % gains are bytes
for $j \leftarrow 0$ **to** $W-1$ **do**
  BitDecompose$(v, 8)$ **for** $v \in \{p_j^{\mathrm{R}}, p_j^{\mathrm{G}}, p_j^{\mathrm{B}}\}$
  SatMul$(p_j^{\mathrm{R}}, g_{\mathrm{R}}, o_j^{\mathrm{R}})$ % $o_j^{\mathrm{R}} = \min(g_{\mathrm{R}}\, p_j^{\mathrm{R}}, 255)$
  SatMul$(p_j^{\mathrm{B}}, g_{\mathrm{B}}, o_j^{\mathrm{B}})$ % $o_j^{\mathrm{B}} = \min(g_{\mathrm{B}}\, p_j^{\mathrm{B}}, 255)$
end for
$d_4^{\mathsf{in}\prime} \leftarrow$ AbsorbRow$(d_4^{\mathsf{in}}, \mathbf{p}^{\mathrm{R}}, \mathbf{p}^{\mathrm{G}}, \mathbf{p}^{\mathrm{B}})$
$d_4^{\mathsf{out}\prime} \leftarrow$ AbsorbRow$(d_4^{\mathsf{out}}, \mathbf{o}^{\mathrm{R}}, \mathbf{p}^{\mathrm{G}}, \mathbf{o}^{\mathrm{B}})$ % green as is
$(\mathsf{pc}', \rho', last) \leftarrow$ AdvanceStage$(\rho, H, 4, 4)$ % last stage
assert $\mathsf{pc}_z = 4$; $\mathsf{pc}'_z \leftarrow 4 + last$ % its turn
assert $last \cdot (d_4^{\mathsf{in}\prime} - d_3^{\mathsf{out}}) = 0$ % link to DEM
return (z', pc')

layout of demosaicing, and $(\mathbf{o}^{\mathrm{R}}, \mathbf{p}^{\mathrm{G}}, \mathbf{o}^{\mathrm{B}})$ into d_4^{out}, so that d_4^{out} is the digest of the delivered image y, which the verifier recomputes.

State Vector of SPECTRA-CAM: The implemented state vector holds sixteen field elements,

$$\begin{aligned} z = \big(&\mathsf{pc}_z,\ \eta_1,\ g,\ g_{\mathrm{R}},\ g_{\mathrm{B}},\ d_1,\ d_L,\ \rho, \\ &d_1^{\mathsf{in}}, d_1^{\mathsf{out}}, \ldots, d_4^{\mathsf{in}}, d_4^{\mathsf{out}}\big), \end{aligned} \tag{21}$$

which is the state of Eq. 7, with $\eta_d = (\eta_1, g, g_{\mathrm{R}}, g_{\mathrm{B}})$, d_1 as d_0^{out} and the digests $d_i^{\mathsf{in}}, d_i^{\mathsf{out}}$ as D_i, extended by d_L and ρ. One point is specific to the implementation: since the anchor is not implemented, d_1 comes unsigned in z_0, and the link of the first stage, $d_1^{\mathsf{in}} = d_1$, is checked by the verifier on the final state instead of by the last substep of F_1.

Row Batching: The model treats each stage as one step, but for large image resolutions, fitting an entire frame in a single circuit can require excessive memory, which is why a stage runs over substeps. With one substep per step, however, every row pays the fixed overhead of a folding step, and folding schemes assume a sufficiently large computation at each step to amortize it [18]. We therefore let a step consist of B step circuits of the same stage, one per substep, synthesized one after the other and chained through the state. A step then advances by B rows of the frame for DPC, digital gain and white balancing, and by B bands, i.e., $2B$ rows, for demosaicing, so that a frame requires

$$N_{\mathrm{steps}}(H, B) = \underbrace{\frac{H}{B}}_{\text{DPC}} + \underbrace{\frac{H}{B}}_{\text{DGAIN}} + \underbrace{\frac{H}{2B}}_{\text{DEM}} + \underbrace{\frac{H}{B}}_{\text{WBAL}} = \frac{7H}{2B} \tag{22}$$

steps in total. The batch size B is part of the circuits and is therefore fixed at setup. Since every step must contain whole demosaicing bands, the admissible batch sizes for a frame of height H are $\mathcal{B}(H) = \{B \in \mathbb{N}_{>0} : 2B \mid H\}$.

To estimate the effect of batching, we split the cost of a folding step into a fixed cost c_0, which models the overhead per fold that is independent of B, such as the recursion overhead and the startup latency of the GPU dispatch, and a cost c_1 per substep, averaged over the stages.

TABLE 2. EVALUATION HARDWARE (RAM AND VRAM IN GB).

Device	CPU	Cores	Thr.	RAM	GPU	VRAM
Jetson	Cortex-A78AE	6	6	8	Orin	shared
Server A	Xeon	4	8	126	RTX 5060	8
Server B	Xeon 5220R	24	48	128	A5000	24
Server C	EPYC	64	128	125	RTX Pro 6000	96

With $N_{\text{steps}}(H, B)$ steps of B substeps each, the proving time is approximately

$$T(H,B) \approx N_{\text{steps}}(H,B)\,(c_0 + c_1 B) = \frac{7H}{2B}\,c_0 + \frac{7H}{2}\,c_1, \tag{23}$$

which decreases in B, since the fixed cost c_0 is paid by fewer steps, while the work of the substeps, $\frac{7H}{2}c_1$, is unchanged. On the other hand, the size of a step grows linearly with B, and so does the memory held for the Multi Scalar Multiplication (MSM) that commits to its witness, which we approximate as

$$M(B) \approx m_0 + m_1 B, \tag{24}$$

where m_0 is the memory that is independent of B and m_1 the memory per substep, which grows with W. Given a memory budget $M_{\max}$, it is possible to select the batch size that minimizes the proving time within the budget,

$$\begin{aligned} B^\star(H) &= \arg\min_{B \in \mathcal{B}(H)} T(H,B) \quad \text{s.t.} \quad M(B) \le M_{\max} \\ &= \max\left\{ B \in \mathcal{B}(H) : B \le \frac{M_{\max} - m_0}{m_1} \right\}. \end{aligned} \tag{25}$$

The two optimization objectives are contrasting, which gives a Pareto front, shown in Figure 8(c).

5. Evaluation

We evaluate proving time and peak memory on all hardware configurations, providing the Pareto frontier based on the least capable device (Figure 8). We also investigate the benefits and costs of GPU acceleration for the MSMs, a known bottleneck [40]. Finally, we propose an attack against the camera pipeline integrity, to show limits of the state of the art, and the capabilities of SPECTRA-CAM under an actual threat (Section 5.3).

5.1. Setup

We implement the SPECTRA-CAM in Rust on the SuperNova implementation of arecibo [41], based on Spartan [42]. H is the 2-ary Poseidon hash [43]. The evaluation is carried out among different devices, with configurations ranging from embedded devices (Jetson) to server grade hardware (Table 2).

SPECTRA-CAM is evaluated on the representative camera pipeline of Eq. 13. For the frame sizes to be proven, we opt for two categories: utility frame sizes, from 480p to 4K, and common classifier sizes, from 16^2 to 224^2, based on GTSRB [44]. The reason behind the two categories resides in the wide range of applications of cameras, from photography, covered by the utility sizes, to autonomous driving, where classifiers require smaller images and embedded devices such as the Jetson are the target platform [45].

Methodology: For a fair benchmarking, all platforms run the same Docker Compose service, built from the `nvidia/cuda:12.8.2-devel-ubuntu24.04` base image, and the prover uses every hardware thread of the device (Table 2). The toolchain used for the Rust prover is 1.97.1 using Cargo's release profile.

Wall-clock time is measured with Rust's `std::time::Instant` around the complete proving loop for one frame. This interval includes constructing and proving every recursive step across all stages. Preparation, public-parameter setup, and final verification occur outside the timed interval. For memory, the benchmarks also record the process's peak resident set size (RSS) over the full run. Each sample runs in a fresh process, and for each device, frame size, and backend, we compare the median proving time at different row batch size $B = \{1, 2, 4, 6, 8, 10, 12, 15, 18\}$. Table 3 reports the lowest of these medians and the corresponding B.

5.2. Performance

Circuit size: Application constraints increase proportionally to the frame sizes (Table 4). In particular, the width W contributes to the number of constraints in an individual step, while height H plays no role in the per-step constraint count, but rather in the total count of the frame. Batching multiplies the count of a step by B and divides the number of steps by B, so the count per frame is unchanged, e.g., approximately 107.0 million white balancing constraints at 720p, but as mentioned in Section 4.1, a larger step can help amortize the overhead cost c_0.

Proving time: As expected, there is a direct dependency between the capabilities of the hardware and the performances. Unlike VIMz [4], whose per-operation circuits can be proven in parallel and linked at the verifier, SPECTRA-CAM proves its steps sequentially with NIVC, since a single proof attests all stages in their order. Proving time therefore grows linearly with the number of constraints, hence with the number of pixels $W \cdot H$. Table 3 reports, per device, the end-to-end proving time of the four step functions. In absolute terms the gap is most visible on the utility frame sizes, however on the classifier sizes it is much smaller, where the Jetson stays at most 32 s behind the fastest server, whereas at 4K the gap reaches 39.5 min.

GPU acceleration: On GPU, only the MSMs of the prover are accelerated, while everything else runs on CPU. On the utility frame sizes the gain shrinks with the CPU core count: from 720p upward it is more than 3× on the Jetson and Server A, 1.6–1.7× on Server B and none on Server C (Table 3). On the Jetson, for example, a 720p proof drops from 1,041.2 s to 315.6 s (3.30×), and a 4K proof, where both backends select $B = 4$, from 9,305.4 s to 2,821.7 s

TABLE 3. PROVING TIME (S) ON CPU (C) AND GPU (G), MEDIAN OF 3 RUNS AT THE FASTEST B (SUBSCRIPT).

	Jetson		Server A		Server B		Server C	
Size	C	G	C	G	C	G	C	G
Utility frame sizes								
480p	366.2_{15}	120.5_{12}	152.4_{12}	52.4_{12}	70.5_{15}	48.5_{12}	22.1_{15}	24.5_{12}
720p	1041.2_{12}	315.6_{12}	417.1_{18}	127.4_{18}	160.3_{18}	101.3_{18}	53.0_{18}	57.4_{18}
1080p	2399.5_{6}	742.4_{6}	856.9_{18}	273.0_{18}	345.8_{18}	205.8_{18}	115.2_{18}	121.7_{18}
2K	2570.7_{6}	792.3_{6}	921.0_{18}	292.7_{18}	370.0_{18}	223.5_{18}	125.8_{12}	129.0_{18}
4K	9305.4_{4}	2821.7_{4}	3368.0_{18}	1049.7_{18}	1280.6_{18}	758.9_{18}	474.0_{6}	450.9_{18}
classifier sizes								
16^2	1.4_{8}	1.4_{8}	0.8_{8}	0.8_{8}	0.9_{8}	0.9_{8}	0.3_{8}	0.3_{8}
32^2	3.8_{8}	3.8_{8}	2.0_{8}	2.0_{8}	2.4_{8}	2.3_{8}	0.7_{8}	0.6_{8}
48^2	5.2_{12}	3.9_{12}	2.5_{12}	3.2_{8}	2.7_{12}	2.2_{12}	0.7_{12}	1.0_{8}
64^2	11.0_{8}	8.4_{8}	5.1_{8}	5.6_{8}	5.6_{8}	4.4_{8}	1.5_{8}	2.4_{8}
80^2	13.5_{10}	8.3_{10}	6.0_{10}	5.4_{10}	6.3_{10}	4.2_{10}	1.7_{10}	2.6_{10}
96^2	17.3_{12}	11.4_{12}	8.1_{12}	6.0_{12}	6.8_{12}	4.7_{12}	2.0_{12}	2.8_{12}
112^2	24.9_{8}	14.8_{8}	11.1_{8}	8.4_{8}	11.5_{8}	7.7_{8}	3.3_{8}	3.9_{8}
128^2	31.9_{8}	18.3_{8}	14.8_{8}	10.0_{8}	12.4_{8}	9.1_{8}	3.9_{8}	4.6_{8}
144^2	28.7_{18}	16.0_{18}	13.3_{18}	6.8_{18}	7.9_{18}	5.4_{18}	2.7_{18}	3.4_{18}
160^2	40.9_{10}	22.2_{10}	19.8_{10}	11.7_{10}	13.0_{10}	10.9_{10}	4.3_{10}	5.4_{10}
176^2	49.5_{8}	31.2_{8}	22.8_{8}	14.3_{8}	19.3_{8}	14.0_{8}	6.0_{8}	6.6_{8}
192^2	55.7_{12}	29.2_{12}	25.8_{12}	11.6_{12}	14.9_{12}	10.3_{12}	5.7_{12}	5.5_{12}
208^2	64.9_{8}	36.2_{8}	31.3_{8}	17.8_{8}	21.8_{8}	17.7_{8}	7.4_{8}	8.0_{8}
224^2	76.1_{8}	40.4_{8}	36.7_{8}	20.0_{8}	24.2_{8}	20.0_{8}	8.5_{8}	8.9_{8}

TABLE 4. CONSTRAINTS PER STEP ($B = 1$, $B = 4$) AND PER FRAME (MILLIONS, $B = 1$).

	Size	DPC	DGAIN	DEM	WBAL
$B = 1$	480p	48,265	16,447	51,680	74,300
	720p	96,578	32,885	103,352	148,574
	1080p	143,696	48,845	153,112	221,414
	2K	154,219	52,515	164,976	237,416
	4K	287,440	97,681	306,216	442,802
$B = 4$	480p	193,063	65,791	206,723	297,203
	720p	386,315	131,543	413,411	594,299
	1080p	574,787	195,383	612,451	885,659
	2K	616,879	210,063	659,907	949,667
	4K	1,149,763	390,727	1,224,867	1,771,211
Total (M)	480p	23.2	7.9	12.4	35.7
	720p	69.5	23.7	37.2	107.0
	1080p	155.2	52.8	82.7	239.1
	2K	166.6	56.7	89.1	256.4
	4K	620.9	211.0	330.7	956.5

TABLE 5. JETSON PEAK RSS (GIB), GPU SPEEDUP AND VERIFICATION TIME (S); B AS IN TABLE 3.

Size	480p	720p	1080p	2K	4K
CPU RSS	3.69	5.71	4.21	4.44	5.75
GPU RSS	3.12	5.92	4.55	4.71	5.99
Speedup	3.04×	3.30×	3.23×	3.24×	3.30×
Verify (s)	5.72	11.18	8.69	9.26	10.58

(3.30×), which is still approximately 47 minutes. On the classifier sizes, however, the GPU-enabled prover does not always perform better than its CPU-only counterpart (Figure 7). Up to a frame size that depends on the device, the pure CPU runs are as fast or faster, and the GPU consistently outperforms only from 48×48 on the Jetson and Server B, 80×80 on Server A, and never on Server C. This may come down to the fact that the steps of such frames are small, so that the overhead of dispatching an MSM to the GPU and of the host-to-device transfer can outweigh the cost of executing everything on the CPU. GPU implementations of the MSM target far larger instances: cuZK [40], for example, is evaluated from 2^{19} points upward and reports the transfer as a cost of its own. The cutoff comes early on the Jetson, whose CPU cores are less powerful. Sub-second proving is reached only at the smallest classifier sizes and only on the servers.

Batch size: The batch size B, hence the number of pixels processed per step, determines the peak memory usage (Eq. 24), which means that it is possible to tune the prover based on the memory budget at setup time. We evaluate this trade-off on the Jetson for a 720p image (Table 6, Figure 8). The number of steps decreases with increasing B, however the peak memory increases accordingly: increasing the GPU batch size from 1 to 12 reduces the number of recursive steps from 2,520 to 210 and proving time from 988.7 s to 315.6 s, while peak RSS rises from approximately 0.68 to 5.92 GiB. Peak memory grows linearly with B, as in Eq. 24, by about 0.47 GiB per substep (Figure 8(b)). The proving time follows Eq. 23: the contribution of the fixed cost c_0 shrinks as $1/B$, and the runtime approaches the term $\frac{7H}{2}c_1$, which batching does not change (Figure 8(a)).

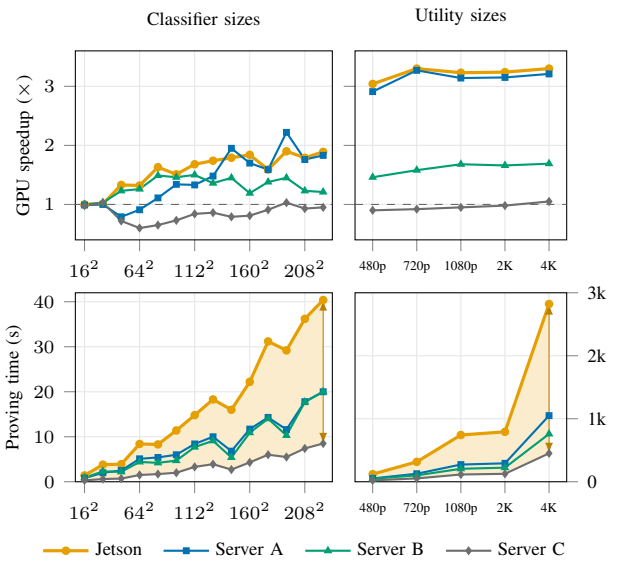


Figure 7. Top: GPU speedup over CPU (CPU faster below the dashed line). Bottom: proving time on each device's best backend; band: Jetson's gap to the fastest, at most 32 s on classifier and 39.5 min on utility sizes.

Memory: Table 5 reports the median process peak resident set size (Linux `VmHWM`) at each selected batch size, i.e., the fastest one that was tested (Table 3), and in its last row the verification time of the GPU runs. The selected batch

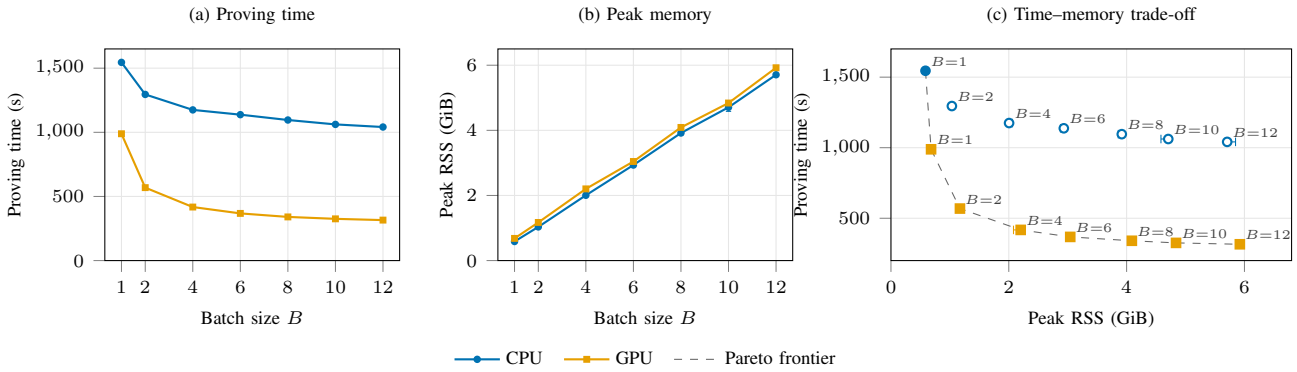


Figure 8. Jetson, 720p: (a) time and (b) peak memory vs. B; (c) time vs. memory, Pareto-optimal filled. Medians of three runs, whiskers show the range.

TABLE 6. JETSON ORIN NANO, 720P, BY BATCH SIZE B (MEDIANS OF THREE RUNS).

		Proving time (s)		Peak RSS (GiB)	
B	Steps	CPU	GPU	CPU	GPU
1	2,520	1545.2	988.7	0.59	0.68
2	1,260	1295.2	568.7	1.03	1.17
4	630	1174.7	417.4	2.01	2.20
6	420	1137.5	367.8	2.93	3.04
8	315	1095.9	340.4	3.92	4.09
10	252	1061.6	325.4	4.71	4.84
12	210	1041.2	315.6	5.71	5.92

size shrinks as the frame grows, from $B = 12$ or more at 480p to $B = 4$ at 4K. In SPECTRA-CAM we maintain a low memory profile, with a maximum memory usage for 4K images of about 6 GiB on the Jetson. This is specifically important when developing ZKPs for embedded devices, where the high memory requirements [6], [19], [30] often make the deployment in constrained devices not suitable.

5.3. Pipeline Ordering Shuffling Attacks

The mechanisms of Table 1 place the trust anchor too late for an attacker inside the sensor to be detected. We show this with Pipeline Ordering Shuffling Attacks (POSA), inspired by execution flow hijacking [17], where an attacker creates adversarial effects solely by diverting the order of execution of the stages (Figure 9).

We demonstrate POSA on the SPECTRA-CAM pipeline, where the inputs are MNIST digits [46] turned into raw Bayer captures with 4, 15 or 45 defective pixels out of 784, the application is one of three image classifiers, ResNet50 [47], VGG16 [48] and ConvNeXt [49], and the gains are swept over the operating range of the sensor; the full setup and results are in Appendix B. We identify 11 admissible reorderings; the strongest ones are those that run demosaicing before DPC, since demosaicing copies the stuck value of each defective pixel into its 2×2 superpixel, and DPC then corrects only the pixel listed in its mask, so the output differs from the benign one at a fixed footprint around every defective pixel.

TABLE 7. POSA ON RAW MNIST: ASR (%) OF THE STRONGEST REORDERING AND SHARE OF CHANGED PIXELS.

		nominal			most favorable		
Defective px	Δ_{px}	ResNet	VGG	ConvNeXt	ResNet	VGG	ConvNeXt
4 (0.5 %)	1.3	1.0	0.1	0.2	2.2	26.4	2.5
15 (1.9 %)	5.9	6.7	1.1	2.1	11.8	73.2	13.0
45 (5.7 %)	18.6	29.1	5.2	8.3	44.3	75.9	51.2

Table 7 reports the attack success rate, the share of correctly classified inputs that the reordered pipeline misclassifies, at the nominal setting and at the most favorable setting of the operating range.

At the nominal setting the effect is small except with many defective pixels; it grows in low light, where the defective pixels dominate a dark digit, and with the number of defective pixels, and the same orders are the strongest for all three classifiers, so the attacker needs no access to the model. How strong the effect is on a real sensor depends on its defect density and illumination. The point of the demonstration is not the magnitude, however, but that the manipulation is invisible to every mechanism of Table 1 except SPECTRA (Corollary 2).

6. Discussion

Residual Attacks: Three limits remain outside the guarantees of Section 3.2. First, the verifier checks z_0 against the specified parameters, so no stage runs with other values; parameters chosen at runtime, e.g., by an auto white balance loop in the untrusted firmware, must therefore be exposed to the verifier, and SPECTRA makes such a choice visible, not impossible, unless the control loop is attested as a further stage. Second, the digest d_1 carries no freshness, so an attacker can replay an earlier readout through the honest pipeline; a timestamp or frame counter signed by the anchor together with d_1 binds every proof to one capture without interaction. Third, the analog stages and the anchor are trusted: $b = 1$ is the lowest boundary a proof can reach, and

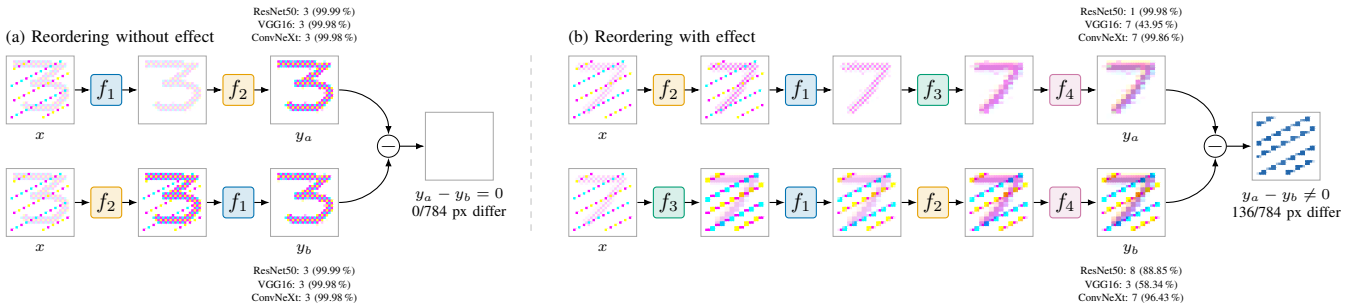


Figure 9. Same frame x, different order of f_1 DPC, f_2 gain, f_3 demosaicing, f_4 white balance: (a) bit-identical outputs; (b) two of three classifiers flip.

attacks on the analog front end, such as optical injection, are out of scope.

Anchor Signature: The verifier checks the signature of the anchor natively, outside every circuit: d_1 is part of the public state z_0, so, as Lemma 2 shows, unforgeability alone binds the proof to the readout of the anchor. A signature is verified inside a proof only when the signer must stay hidden, as in VIMz [4]; if the anonymity of the sensor were required, such a statement could be added next to SPECTRA without changing any step function, since it refers only to the public d_1. SPECTRA-CAM does not implement the anchor and takes d_1 as given (Section 4).

Practicality: The evaluation of SPECTRA-CAM shows that proving at the input resolutions of common vision classifiers takes tens of seconds on the Jetson. This is already practical when frames need not be attested one by one, e.g., by proving periodic key frames and checking consistency across the frames between them. Full-resolution frames remain out of real-time reach and, for now, suit post-hoc verification; dedicated hardware accelerators [50], [51] narrow this gap.

Other Sensors: The SPECTRA design is general: every digital processing stage can be integrated, provided that it is arithmetized as a step circuit, which enables a wide range of SPECTRA sensors.

Privacy: The verifier never sees the intermediate data, which enter the proof only as advice. The recursive proof we evaluate is not ZK by itself. ZK can be obtained by compressing the final proof with a ZK SNARK [18], which we do not evaluate.

7. Conclusion

Existing integrity measures for sensor data trust the sensor up to the point where its output is signed, so a manipulation of any processing stage before the signature goes unnoticed. We close this gap with a sensor model in which the trust boundary moves from the output to the conversion stage, and in which a stage counts as attested only when the correctness of its execution and the provenance of its input can be verified without trusting the component that runs it. Under this definition no prior approach attests a single stage between conversion and signature, and we show that reordering correct stages, without injecting any content, suffices to mislead downstream classifiers while the output still carries a valid signature.

SPECTRA realizes the model with NIVC, where every stage is defined as a step circuit, the proof is anchored at the signed readout, and acceptance implies that the output results from the prescribed stages in the prescribed order. SPECTRA-CAM instantiates it for the camera pipeline on top of SuperNova [18] and is, to the best of our knowledge, the first image-based, optionally ZK, proof system evaluated directly on embedded hardware. On the Jetson, classifier-sized frames prove within tens of seconds of a server and a 4K frame fits in under 6 GiB, with the batch size as the knob between memory and proving time.

Attesting the control loops that choose parameters at runtime, such as auto white balance, as further SPECTRA stages would extend the guarantee from how the data was processed to the choice of the processing itself, and is the natural next step.

References

[1] M. Calipari, L.-C. Chen, M. Kühr, M. Hamad, and S. Steinhorst, "PRISM-D: PRNU-based runtime image-tampering detection for automotive cameras," in *4th EAI International Conference on Security and Privacy in Cyber-Physical Systems and Smart Vehicles (SmartSP '26)*, 2026, in press.

[2] Coalition for Content Provenance and Authenticity, "C2PA technical specification, version 2.2," https://spec.c2pa.org/specifications/specifications/2.2/specs/C2PA_Specification.html, 2025.

[3] MIPI Alliance, "A Guide to the MIPI Camera Security Framework for Automotive Applications," Oct. 2024. [Online]. Available: https://www.mipi.org/download-the-mipi-camera-security-white-paper

[4] S. Dziembowski, S. Ebrahimi, and P. Hassanizadeh, "VIMz: Private proofs of image manipulation using folding-based zkSNARKs," *Proceedings on Privacy Enhancing Technologies*, 2025.

[5] P. Della Monica, I. Visconti, A. Vitaletti, and M. Zecchini, "Trust nobody: privacy-preserving proofs for edited photos with your laptop," in *2025 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2025, pp. 4624–4642.

[6] T. Datta, B. Chen, and D. Boneh, "VerITAS: verifying image transformations at scale," in *2025 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2025, pp. 4606–4623.

[7] M. Kühr, M. Hamad, P. MohajerAnsari, M. D. Pesé, and S. Steinhorst, "SoK: Security of the image processing pipeline for camera-based sensing in autonomous vehicles," in *Proceedings of the ACM Asia Conference on Computer and Communications Security*, 2026, pp. 1738–1754.

[8] C. Yan, H. Shin, C. Bolton, W. Xu, Y. Kim, and K. Fu, "SoK: A minimalist approach to formalizing analog sensor security," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 233–248.

[9] Z. Jin, Q. Jiang, X. Lu, C. Yan, X. Ji, and W. Xu, "PhantomLiDAR: Cross-modality signal injection attacks against LiDAR," in *Network and Distributed System Security (NDSS) Symposium*, San Diego, CA, USA, 2025. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2025-997-paper.pdf

[10] M. Buckler, S. Jayasuriya, and A. Sampson, "Reconfiguring the imaging pipeline for computer vision," in *2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2017, pp. 975–984.

[11] B. Phan, F. Mannan, and F. Heide, "Adversarial imaging pipelines," *arXiv preprint arXiv:2102.03728*, 2021.

[12] M. Calipari, M. Kühr, D. Kulmer, M. Luedecke, M. Hamad, and S. Steinhorst, "Perspective-Shift attacks against optical perception sensors: A novel attack vector on LiDAR and camera," in *4th USENIX Symposium on Vehicle Security and Privacy (VehicleSec 26)*, 2026, pp. 283–299.

[13] T. Spring, "DEF CON 2019: Picture perfect hack of a Canon EOS 80D DSLR," Threatpost, Aug. 2019, accessed: 2026-11-03. [Online]. Available: https://threatpost.com/hack-of-a-canon-eos-80d-dslr/147214/

[14] National Institute of Standards and Technology, "CVE-2021-36260: Command injection vulnerability in the web server of some Hikvision products," National Vulnerability Database, 2021, cVSS v3.1 base score 9.8 (Critical). Accessed: 2026-11-03. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2021-36260

[15] M. Calipari, F. M. Schmidt, M. Hamad, and S. Steinhorst, "CIAK-CP: Camera feed injection attack in collaborative perception," in *The 41st ACM/SIGAPP Symposium On Applied Computing (SAC2026)*, Mar. 2026, in press.

[16] Q. Zhang, S. Jin, R. Zhu, J. Sun, X. Zhang, Q. A. Chen, and Z. M. Mao, "On data fabrication in collaborative vehicular perception: Attacks and countermeasures," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024, pp. 6309–6326.

[17] T. Abera, N. Asokan, L. Davi, J.-E. Ekberg, T. Nyman, A. Paverd, A.-R. Sadeghi, and G. Tsudik, "C-flat: control-flow attestation for embedded systems software," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 743–754.

[18] A. Kothapalli and S. Setty, "Supernova: Proving universal machine executions without universal circuits," *Cryptology ePrint Archive*, 2022.

[19] D. Kang, T. Hashimoto, I. Stoica, and Y. Sun, "ZK-IMG: Attested images via zero-knowledge proofs to fight disinformation," *arXiv preprint arXiv:2211.04775*, 2022.

[20] A. Koyama, K. Toyoda, M. Fujimoto, and T. H. Tran, "zk-REAL: A zero-knowledge-based protocol for repeated image edit authenticity proof with lattice hashing," *IEEE Open Journal of the Computer Society*, 2026.

[21] C.-Y. Lin and S.-F. Chang, "A robust image authentication method distinguishing JPEG compression from malicious manipulation," *IEEE transactions on circuits and systems for video technology*, vol. 11, no. 2, pp. 153–168, 2001.

[22] J. Apostolopoulos, S. Wee, F. Dufaux, T. Ebrahimi, Q. Sun, and Z. Zhang, "The emerging JPEG-2000 security (JPSEC) standard," in *2006 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2006.

[23] P. Gilbert, J. Jung, K. Lee, H. Qin, D. Sharkey, A. Sheth, and L. P. Cox, "Youprove: authenticity and fidelity in mobile sensing," in *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, 2011, pp. 176–189.

[24] T. Winkler, A. Erdélyi, and B. Rinner, "TrustEYE.M4: protecting the sensor—not the camera," in *2014 11th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*. IEEE, 2014, pp. 159–164.

[25] P. W. Khan, Y.-C. Byun, and N. Park, "A data verification system for cctv surveillance cameras using blockchain technology in smart cities," *Electronics*, vol. 9, no. 3, p. 484, 2020.

[26] H. Ko, I. Lee, S. Lee, J. Kim, and H. Oh, "Efficient verifiable image redacting based on zk-SNARKs," in *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, 2021, pp. 213–226.

[27] Y. Liu, Y. Nakatsuka, A. A. Sani, S. Agarwal, and G. Tsudik, "Vronicle: verifiable provenance for videos from mobile devices," in *Proceedings of the 20th annual international conference on mobile systems, applications and services*, 2022, pp. 196–208.

[28] Y. Liu, Z. Yao, M. Chen, A. Amiri Sani, S. Agarwal, and G. Tsudik, "Provcam: A camera module with self-contained tcb for producing verifiable videos," in *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, 2024, pp. 588–602.

[29] Y. Jang, "Signing right away," *arXiv preprint arXiv:2510.09656*, 2025.

[30] C. Zhang, X. Yang, D. Oswald, M. Ryan, and P. Jovanovic, "Eva: Efficient privacy-preserving proof of authenticity for lossily encoded videos," in *2025 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2025, pp. 4643–4662.

[31] P. Perazzo, M. Mattei, G. Anastasi, M. Avvenuti, G. Dini, G. Lettieri, and C. Vallati, "JPEGs just got snipped: Croppable signatures against deepfake images," in *2025 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2025, pp. 1–6.

[32] J. Marinho, E. Schiller, A. Debauge, and N. Foukia, "ZK-Disclosure: Privacy-preserving information disclosure for digital evidence with C2PA and zk-SNARKs," in *2025 Annual Computer Security Applications Conference Workshops (ACSAC Workshops)*. IEEE, 2025, pp. 309–315.

[33] A. Frolov, J. Guo, X. Zhao, T. Datta, D. Boneh, and I. Miers, "zk-Cinema: Proving video provenance in zero knowledge," *Cryptology ePrint Archive*, 2026.

[34] S. Dittmer, S. Lu, K. Model, and J. Near, "ZK-JPEG: Zero-knowledge image editing and compression," *Cryptology ePrint Archive*, 2026.

[35] S. Goldwasser, S. Micali, and R. L. Rivest, "A digital signature scheme secure against adaptive chosen-message attacks," *SIAM Journal on Computing*, vol. 17, no. 2, pp. 281–308, 1988.

[36] A. Agarwal, N. Ratha, M. Vatsa, and R. Singh, "Crafting adversarial perturbations via transformed image component swapping," *IEEE Transactions on Image Processing*, vol. 31, pp. 7338–7349, 2022.

[37] A. Arun and S. Setty, "Nebula: Proving machine executions via folding schemes," in *2026 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2026, pp. 3223–3242.

[38] T. Bilal, B. Amjad, T. Iqbal, B. Zafar, K. Usman, S. I. Bhatti, A. R. Qureshi, M. Rafiq, M. Nadeem, J. Amjad *et al.*, "Infinite-isp: An open source hardware image signal processor platform for all imaging needs," *Electronic Imaging*, vol. 37, pp. 1–6, 2025.

[39] B. Jähne, "What is new about release 4.1 of the standard EMVA 1288? – adaption to modern image sensors," Presentation at the Scientific VISION Days, VISION 2024, Messe Stuttgart, Stuttgart, Germany, Oct. 2024, heidelberg Collaboratory for Image Processing (HCI), Heidelberg University & EMVA. [Online]. Available: https://www.messe-stuttgart.de/vision/fileadmin/media/02-programm/Scientific_VISION_Days_2024/08.10.2024_PDFs_Praesentationen/05_SVD_EMVA_Bernd_Jaehne.pdf

[40] T. Lu, C. Wei, R. Yu, C. Chen, W. Fang, L. Wang, Z. Wang, and W. Chen, "cuZK: Accelerating zero-knowledge proof with a faster parallel multi-scalar multiplication algorithm on GPUs," *Cryptology ePrint Archive*, 2022.

[41] Lurk-Lang, "Arecibo: Supernova implementation," https://github.com/lurk-lang/arecibo/tree/8997d733aa69c3b6db5a44e06f7c0f74ad315d6f/src/supernova, 2025, source code repository, commit 8997d733aa69c3b6db5a44e06f7c0f74ad315d6f.

[42] S. Setty, "Spartan: Efficient and general-purpose zkSNARKs without trusted setup," in *Annual International Cryptology Conference*. Springer, 2020, pp. 704–737.

[43] L. Grassi, D. Khovratovich, C. Rechberger, A. Roy, and M. Schofnegger, "Poseidon: A new hash function for zero-knowledge proof systems," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 519–535.

[44] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition," *Neural networks*, vol. 32, pp. 323–332, 2012.

[45] R. Muller, R. Song, C. Wang, Y. Zhan, J.-P. Monteuuis, Y. Man, M. Li, R. Gerdes, J. Petit, and Z. B. Celik, "Investigating physical latency attacks against camera-based perception," in *2025 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2025, pp. 4588–4605.

[46] Y. LeCun, "The mnist database of handwritten digits," *http://yann.lecun. com/exdb/mnist/*, 1998.

[47] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[48] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[49] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, "A ConvNet for the 2020s," in *2022 IEEE/CVF conference on computer vision and pattern recognition (CVPR)*. IEEE, 2022, pp. 11 966–11 976.

[50] Y. Yang, Z. Lu, J. Zeng, X. Liu, X. Qian, and Z. Yu, "FALIC: An FPGA-based multi-scalar multiplication accelerator for zero-knowledge proof," *IEEE Transactions on Computers*, vol. 73, no. 12, pp. 2791–2804, 2024.

[51] J. Tong, J. Dang, S. Langowski, T. Huang, A. Ali, J. Kun, J. Jiang, S. Devadas, and T. Krishna, "Enabling AI ASICs for zero knowledge proof," in *Proceedings of the 63rd ACM/IEEE Design Automation Conference*, ser. DAC '26. New York, NY, USA: Association for Computing Machinery, 2026. [Online]. Available: https://doi.org/10.1145/3770743.3804402

Appendix A. Gadgets

The five gadgets shared by the step circuits of Section 4, with the notation of Section 4.1: **assert** adds an R1CS constraint, and a *hint* is computed natively by the prover and bound by the constraints that follow.

Range Checks: Every range check decomposes a value into bits (Algorithm 5, Figure 10): each bit is constrained to be boolean, and the bits must recompose the value, so any value of 2^n or more leaves the system unsatisfiable, as long as 2^n does not exceed the field modulus.

Saturating Multiplication: The saturating multiplication of Eq. 16 and Eq. 19 multiplies two bytes and clamps the product to 255 (Algorithm 6). The product of two bytes fits in 16 bits, and its high byte is non-zero exactly when the product exceeds 255, so the overflow flag follows from the inverse trick, $ov = [high \neq 0]$, and a single select constraint pins the output to 255 or to the low byte.

Algorithm 5 Bit decomposition gadget

Require: Field element v, bit count n
Ensure: Bits $(u_0, \ldots, u_{n-1})$ with $v = \sum_s 2^s u_s$
1: **for** $s \leftarrow 0$ **to** $n-1$ **do**
2: $\quad u_s \leftarrow (v \gg s) \wedge 1$ % hint
3: $\quad$ **assert** $u_s \cdot (1 - u_s) = 0$ % u_s is a bit
4: **end for**
5: **assert** $\sum_{s=0}^{n-1} 2^s u_s = v$ % bits recompose v
6: **return** $(u_0, \ldots, u_{n-1})$

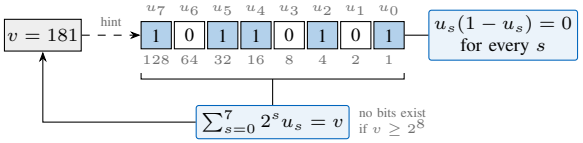


Figure 10. Bit decomposition (Algorithm 5) of $v = 181$ into $n = 8$ bits. The prover supplies the bits as hints, n constraints make each a bit, and one constraint recomposes v from them, so the system is satisfiable only if $v < 2^n$.

Algorithm 6 Saturating multiplication gadget

Require: Pixel p and gain g, each already constrained to 8 bits; claimed output o
Ensure: $o = \min(g \cdot p, 255)$
1: **assert** $g \cdot p = m$ % fresh product variable
2: $(u_0, \ldots, u_{15}) \leftarrow \textsc{BitDecompose}(m, 16)$
3: $low \leftarrow \sum_{s=0}^{7} 2^s u_s$; $high \leftarrow \sum_{s=8}^{15} 2^{s-8} u_s$ % linear
4: $inv \leftarrow high^{-1}$ if $high \neq 0$, else 0 % hint
5: **assert** $high \cdot inv = ov$; $high \cdot (1 - ov) = 0$
% $ov = [high \neq 0]$
6: **assert** $ov \cdot (255 - low) = o - low$ % $o = ov\ ?\ 255 : low$

Pixel Packing and Digest Absorption: Every digest is a hash chain, $d' = \mathsf{H}(d, e)$, which absorbs one field element e per link (Algorithm 8, Figure 11); every step extends the chain by its own rows, so the final digest commits to the full sequence of rows in order. Before absorption, following the lossless pixel compression of VIMz [4], Algorithm 7 packs 31 pixels into one field element in base 256, or 254 flags in base 2, with one linear constraint per element and no range check. The packing is injective only if every pixel is a byte, which is what the range checks of every stage enforce on its inputs, and what its outputs inherit. The chunk boundaries are part of the digest layout, so both sides of every link of Eq. 9 pack one row per call. We write $\textsc{AbsorbRow}(d, \mathbf{p}_1, \mathbf{p}_2, \ldots)$ for packing and absorbing the rows $\mathbf{p}_1$, then $\mathbf{p}_2$, and so on, into d, and $\textsc{AbsorbBits}$ for the analogue in base 2.

Stage Handover: Each substep ends with the same bookkeeping (Algorithm 9, Figure 12): it advances the substep counter ρ and, once the stage has consumed its $N_{\text{sub}}(i)$ substeps, it hands the program counter to the next step circuit and resets ρ to 0, so that the next stage counts its own substeps from 0. The intermediate program counters of a step are constrained to remain at the current circuit, so only the final substep may hand over, and since $N_{\text{sub}}(i)$

Algorithm 7 Pixel packing gadget

Require: Pixels $\mathbf{p}$, each already constrained to 8 bits; $c = 31$ pixels per field element
Ensure: Packed field elements $(P_0, \dots, P_{n-1})$, $n = \lceil W/c \rceil$
1: **for** $s \leftarrow 0$ **to** $n-1$ **do**
2: $\quad P_s \leftarrow \sum_{j=sc}^{sc+c-1} 256^{j-sc} p_j$ % hint
3: $\quad$ **assert** $\sum_{j=sc}^{sc+c-1} 256^{j-sc} p_j = P_s$ % binds P_s
4: **end for**
5: **return** $(P_0, \dots, P_{n-1})$

Algorithm 8 Digest absorption gadget

Require: Running digest d, field elements $(e_0, \dots, e_{n-1})$
Ensure: Digest committing to the input d and to all e_s in order
1: **for** $s \leftarrow 0$ **to** $n-1$ **do**
2: $\quad d \leftarrow \mathsf{H}(d, e_s)$ % 2-ary Poseidon
3: **end for**
4: **return** d

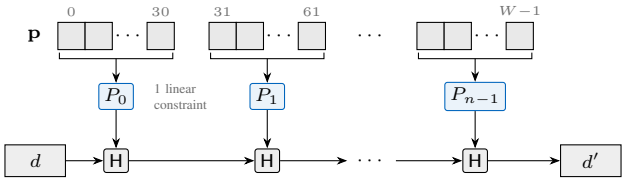


Figure 11. Packing and absorbing one row, AbsorbRow$(d, \mathbf{p})$. Algorithm 7 packs every chunk of 31 pixels into one field element $P_s = \sum_{j=31s}^{31s+30} 256^{j-31s} p_j$, and Algorithm 8 chains them into the running digest, $d \leftarrow \mathsf{H}(d, P_s)$. AbsorbBits is the same with chunks of 254 flags in base 2.

is a constant of the shape, a stage cannot hand over early. The last stage passes its own index as the next one, so the program counter stays while ρ still wraps to 0. The gadget returns the flag *last* to the step circuit, which uses it to move the program counter pc_z of the state from i to $i+1$, also after the last stage, and to enforce the checks on the complete digests.

Algorithm 9 Stage handover gadget

Require: Substep counter ρ; substeps in this stage $N_{\text{sub}}(i)$; own index i; next index i_{next}
Ensure: Next program counter pc', substep counter ρ' and flag *last*
1: $inv \leftarrow (\rho+1-N_{\text{sub}}(i))^{-1}$ if $\rho+1 \neq N_{\text{sub}}(i)$, else 0 % hint
2: **assert** $(\rho+1-N_{\text{sub}}(i)) \cdot inv = 1 - last$; $\quad (\rho+1-N_{\text{sub}}(i)) \cdot last = 0$
% $last = [\rho+1 = N_{\text{sub}}(i)]$
3: **assert** $\rho' = \rho + 1 - N_{\text{sub}}(i) \cdot last$ % wraps to 0 on handover
4: **assert** $\mathsf{pc}' = i + (i_{\text{next}} - i) \cdot last$
5: **return** $(\mathsf{pc}', \rho', last)$

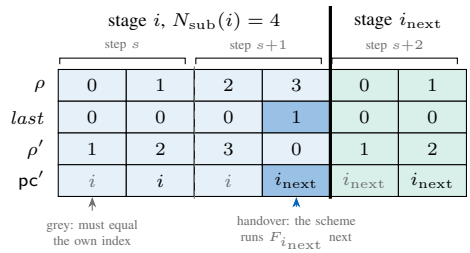


Figure 12. The stage handover (Algorithm 9) on a toy run with $N_{\text{sub}}(i) = 4$ substeps for stage i and $B = 2$ substeps per step. The counter ρ counts the substeps of the stage, and *last* is set only on the final substep, which wraps ρ to 0 and hands the program counter to i_{next}. Only the program counter of the last substep of a step leaves the circuit; the others are constrained to the own index, so a stage cannot hand over in the middle of a step.

Appendix B. POSA on MNIST

This appendix gives the setup and the full results of the reordering attack summarised in Table 7. We use the camera as case study, since mature standards and open-source implementations of its stages exist [38]; the model applies to any cascading pipeline, e.g., Light Detection and Ranging (LiDAR) [12], which we leave to future work. We evaluate on MNIST [46]: a 28×28 px grayscale grid allows a controlled analysis of what the attack needs, without noise or other variables.

Pipeline and Reorderings: The pipeline is the four digital stages of SPECTRA-CAM in the order of Eq. 13, DPC, digital gain (DGAIN), demosaicing (DEM) and white balancing (WBAL), with the conversion stage replaced by the simulated raw capture below. A *reordering* ϑ applies the same four stages, each with its own parameters, in a different order, and is *admissible* if consecutive stages fit together as the pipeline definition requires (Eq. 2). DEM maps the single-channel mosaic to three channels and WBAL accepts only three-channel input, so every order that places WBAL before DEM is discarded: of the $4! = 24$ orders, 12 are admissible (Table 8), 11 of them different from the benign order ϑ_{id}. We call a reordering *effective* on an input if its output differs from the benign output, and *successful* if the label assigned by the classifier differs as well. Orders that skip or repeat stages are rejected by Corollary 2 in the same way and are not evaluated.

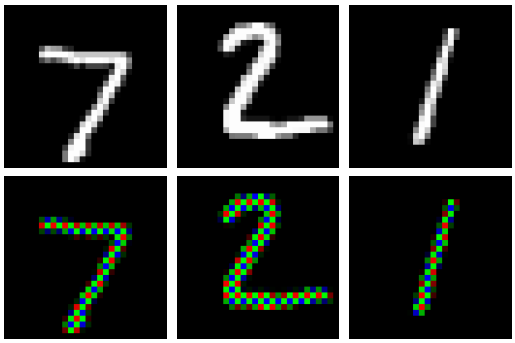
Raw Simulation: MNIST is grayscale, whereas a raw pipeline expects a single-channel mosaic [39]. We simulate the capture by placing an RGGB Color Filter Array (CFA) over each digit and treating its intensity as the irradiance incident on the sensor. Let $\mathcal{I}_{\text{MNIST}} \in [0,1]^{H \times W}$, with $H = W = 28$, and let the Bayer pattern $\mathcal{C} = \left(\begin{smallmatrix} R & G \\ G & B \end{smallmatrix}\right)$ be tiled over the image, $\text{CFA} = \mathbf{1}_{H/2 \times W/2} \otimes \mathcal{C}$, with $\otimes$ the Kronecker product and $\mathbf{1}$ the all-ones matrix. The raw image is

$$\mathcal{I}_{\text{MNIST,raw}} = \mathcal{I}_{\text{MNIST}} \odot \text{CFA} \in [0,1]^{H \times W}, \qquad (26)$$

where $\odot$ is the element-wise product (Figure 13).

TABLE 8. ADMISSIBLE ORDERS OF THE FOUR STAGES.

$O_1 = \vartheta_{\mathrm{id}}$	DPC → DGAIN → DEM → WBAL
O_2	DPC → DEM → DGAIN → WBAL
O_3	DPC → DEM → WBAL → DGAIN
O_4	DGAIN → DPC → DEM → WBAL
O_5	DGAIN → DEM → DPC → WBAL
O_6	DGAIN → DEM → WBAL → DPC
O_7	DEM → DPC → DGAIN → WBAL
O_8	DEM → DPC → WBAL → DGAIN
O_9	DEM → DGAIN → DPC → WBAL
O_{10}	DEM → DGAIN → WBAL → DPC
O_{11}	DEM → WBAL → DPC → DGAIN
O_{12}	DEM → WBAL → DGAIN → DPC



label 7 label 2 label 1

Figure 13. Three samples from the MNIST test set: $\mathcal{I}_{\mathrm{MNIST}}$ (top) and $\mathcal{I}_{\mathrm{MNIST,raw}}$ (bottom), each photosite shown in its RGGB filter color.

Setup: The sensor is simulated as the inverse of the benign pipeline: each raw image is captured at an exposure of $1/4$, with the R and B photosites attenuated by the gains the nominal white balance undoes, so that the nominal setting, digital gain $g = 4$ and white balance gains $g_{\mathrm{R}} = g_{\mathrm{B}} = 4$, restores $\mathcal{I}_{\mathrm{MNIST}}$. Defective pixels read as stuck at full scale. We draw three defect masks at random positions, nested so that each contains the previous one, with 4, 15 and 45 defective pixels out of 784 (0.5 %, 1.9 % and 5.7 %), which we call the hard, medium and easy set from the attacker's point of view. The application is one of three MNIST-trained classifiers, ResNet50 [47], VGG16 [48] and ConvNeXt [49], evaluated on the 10,000 test images. The classifiers are trained on the standard MNIST dataset, so their accuracy on the benign output measures the fidelity of the pipeline: at the nominal setting it is 88 % for ResNet50 and 98–99 % for VGG16 and ConvNeXt, i.e., the pipeline reproduces the digit. Each capture is processed at every setting of the sweep $g \in \{2, 4, 8, 16\}$, $g_{\mathrm{R}}, g_{\mathrm{B}} \in \{1, 2, 4, 8\}$, which spans the operating points of the sensor around the nominal one, from a dark to a saturated digit. A setting belongs to the *operating range* of a classifier if its benign accuracy there is at least 80 %; VGG16 and ConvNeXt are in range at all 64 settings, ResNet50 at 38 to 43 depending on the set. All attack numbers are reported inside the operating range, since outside it the benign pipeline itself fails and a misclassification cannot be attributed to the reordering. Parameters are fixed within a capture and swept across captures, in line with the threat model of Section 3: the sweep explores operating points, not parameters chosen by the attacker.

Attack Success Rate: Let $\mathcal{F}$ be the classifier, $y(x)$ the benign output for the raw input x and $y_\vartheta(x)$ the output of the reordering ϑ. For a labelled set $\mathcal{D} = \{(x^{(n)}, o^{(n)})\}_n$, let $\mathcal{D}_{\mathrm{corr}} = \{\, n \mid \mathcal{F}(y(x^{(n)})) = o^{(n)} \,\}$ be the samples the benign pipeline classifies correctly. The Attack Success Rate (ASR) of ϑ is the fraction of these that the reordered pipeline misclassifies,

$$\mathrm{ASR}(\vartheta) = \frac{\left|\{\, n \in \mathcal{D}_{\mathrm{corr}} \mid \mathcal{F}(y_\vartheta(x^{(n)})) \neq o^{(n)} \,\}\right|}{|\mathcal{D}_{\mathrm{corr}}|}, \quad (27)$$

so that only errors caused by the reordering are attributed to ϑ.

Results: Table 9 reports, for every admissible reordering, the highest ASR over the operating range and the share Δ_{px} of output pixels that differ from the benign output. O_2 and O_3 are not effective: they run DPC first and reproduce the benign output exactly. O_4 is effective but not successful, with an ASR below 2 % for every classifier. The remaining orders all run demosaicing before DPC: demosaicing copies the stuck value of each defective pixel, full scale, into its 2×2 superpixel, and DPC then corrects only the pixel listed in the mask, from neighbors that are already contaminated. The output therefore differs from the benign one at a fixed set of positions, the footprint of the defect mask, identical for every image and every setting, which is why Δ_{px} is constant per set and why the ASR grows with the number of defective pixels (Figure 9b). With the hard set, the ASR stays below 3 % for ResNet50 and ConvNeXt, while VGG16 is the most sensitive, about $10\times$ higher but still limited (≈ 26 % in the worst case). With the medium set, where ≈ 6 % of the pixels differ, the ASR of VGG16 rises to ≈ 73 %, that of ResNet50 to ≈ 12 % and that of ConvNeXt to ≈ 13 %. With the easy set, ≈ 19 % of the pixels differ and the attack succeeds on all three classifiers: ≈ 76 % for VGG16, ≈ 44 % for ResNet50 and ≈ 51 % for ConvNeXt. The highest ASR is reached by O_7 and O_8, which produce the same output, for VGG16 and ConvNeXt, and by O_{11} for ResNet50 on the easy and medium sets, within 0.2 points of O_7. The ranking of the orders is the same for the three classifiers, so the attacker selects ϑ once, without access to $\mathcal{F}$, as the threat model assumes; the sweep over all orders serves the evaluation, not the attacker.

All Settings: For VGG16 and ConvNeXt the attack is most effective at the lowest digital gain ($g = 2$) and white balance gains ($g_{\mathrm{R}} = g_{\mathrm{B}} = 1$), where the defective pixels dominate a dark digit, and its effectiveness decreases as the gains grow, whereas for ResNet50 the ASR depends little on the gains. All settings in the operating range have a benign accuracy of at least 80 %, and of at least 94 % for VGG16 and ConvNeXt.

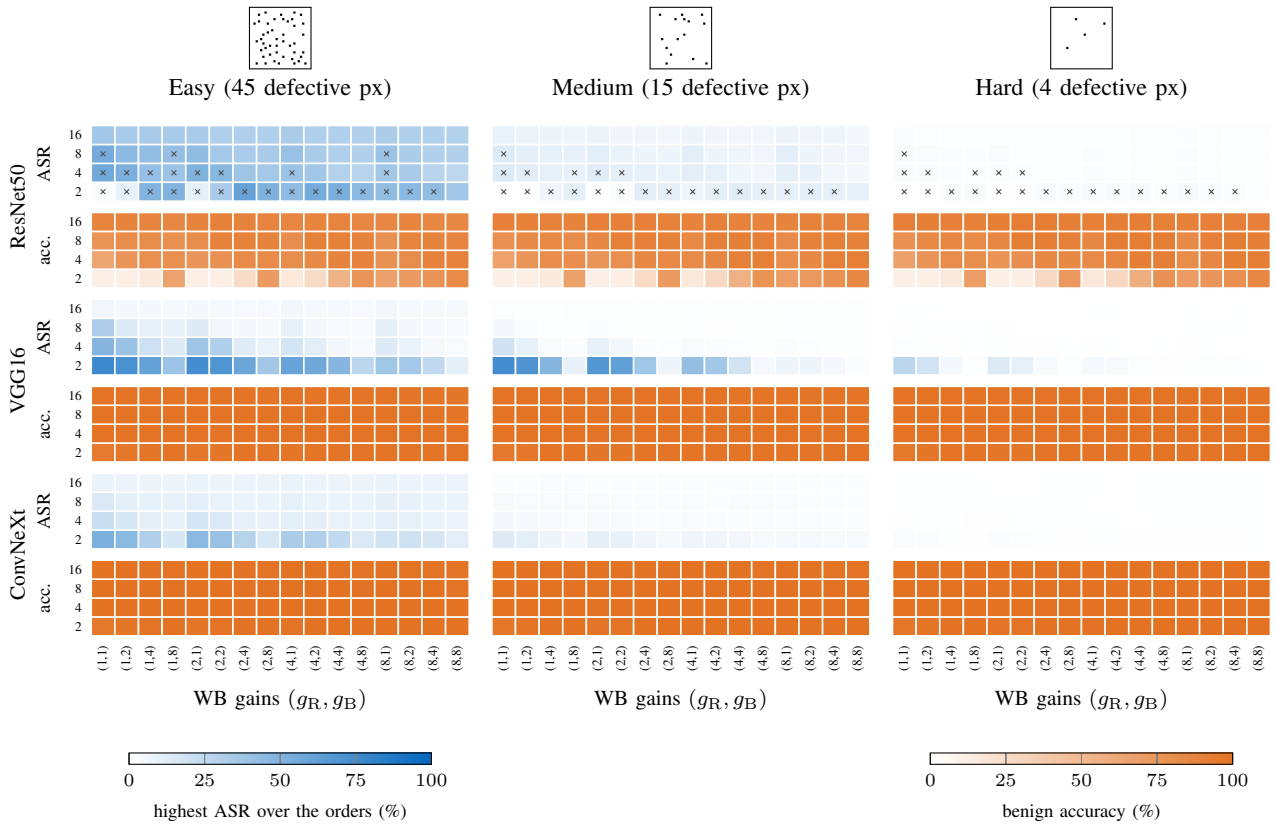


Figure 14. POSA on MNIST at every setting of the sweep, for the three defective-pixel sets and the three classifiers, on the 10,000 test images. Per classifier, top: the highest ASR over the adversarial orders at each setting (digital gain g, rows; white balance gains g_R, g_B, columns), white 0 % to blue 100 %; bottom: the benign accuracy at that setting, white 0 % to orange 100 %, so pale cells are less reliable; ×: benign accuracy below 80 %, the ASR there is measured on few images.

TABLE 9. POSA ON MNIST: HIGHEST ASR (%, EQ. 27) OF EVERY ADVERSARIAL PERMUTATION (TABLE 8) OVER THE SWEEP SETTINGS IN EACH CLASSIFIER'S OPERATING RANGE (BENIGN ACCURACY $\geq$ 80 %), PER DEFECTIVE-PIXEL SET; PERMUTATIONS WITH IDENTICAL OUTPUTS SHARE A ROW, THE HIGHEST PER CLASSIFIER AND SET IS BOLD. Δ_{px}: SHARE OF PIXELS DIFFERING FROM ϑ_{id} AT THAT SETTING (A RANGE WHEN THE CLASSIFIERS' SETTINGS DIFFER). ALL SETTINGS: FIGURE 14.

	Easy (45 defective px)				Medium (15 defective px)				Hard (4 defective px)			
Order	Δ_{px}	ResNet50	VGG16	ConvNeXt	Δ_{px}	ResNet50	VGG16	ConvNeXt	Δ_{px}	ResNet50	VGG16	ConvNeXt
O_2,O_3	0.00	0.0	0.0	0.0	0.00	0.0	0.0	0.0	0.00	0.0	0.0	0.0
O_4	4.09–8.47	1.6	0.1	0.4	1.88–2.93	0.9	0.1	0.2	0.58–0.70	0.7	0.1	0.1
O_5,O_9	18.62	34.3	73.4	42.1	5.87	9.4	71.3	9.2	1.22–1.28	1.7	22.5	2.2
O_6,O_{10},O_{12}	18.62	21.0	73.4	42.1	5.87	6.8	71.3	9.2	1.22–1.28	1.6	22.5	2.2
O_7,O_8	18.44–18.62	44.1	**75.9**	**51.2**	5.79–5.87	11.7	**73.2**	**13.0**	1.22–1.28	2.2	**26.4**	**2.5**
O_{11}	18.44–18.62	**44.3**	**75.9**	**51.2**	5.79–5.87	**11.8**	**73.2**	**13.0**	1.22–1.28	**2.2**	**26.4**	**2.5**

ASR (%) 100 75 50 25 0

Appendix C. Ethical Consideration

This work develops a defensive mechanism for sensor data attestation using NIVC. It involves no human subjects, no personal data, and no interaction with live systems. The POSA offensive attack has been evaluated on synthetic data generated ad-hoc for the evaluation.

Appendix D. Generative AI Disclosure

Generative AI was used for editorial purposes in this manuscript, and all outputs were inspected by the authors to ensure accuracy and originality. These edits cover grammar, spelling, wording, and presentation in all sections, as well as in assistance for debugging and code reviews.