Modular Sumcheck Proofs with Applications to Machine Learning and Image Processing

David Balbás
IMDEA Software Institute &
Universidad Politécnica de Madrid
Madrid, Spain

Dario Fiore
IMDEA Software Institute
Madrid, Spain

Maria Isabel González Vasco
Universidad Carlos III de Madrid
Madrid, Spain

Damien Robissout
IMDEA Software Institute
Madrid, Spain

Claudio Soriente
NEC Laboratories Europe
Madrid, Spain

ABSTRACT

Cryptographic proof systems provide integrity, fairness, and privacy in applications that outsource data processing tasks. However, general-purpose proof systems do not scale well to large inputs. At the same time, ad-hoc solutions for concrete applications—e.g., machine learning or image processing—are more efficient but lack modularity, hence they are hard to extend or to compose with other tools of a data-processing pipeline.

In this paper, we combine the performance of tailored solutions with the versatility of general-purpose proof systems. We do so by introducing a modular framework for verifiable computation of sequential operations. The main tool of our framework is a new information-theoretic primitive called Verifiable Evaluation Scheme on Fingerprinted Data (VE) that captures the properties of diverse sumcheck-based interactive proofs, including the well-established GKR protocol. Thus, we show how to compose VEs for specific functions to obtain verifiability of a data-processing pipeline.

We propose a novel VE for convolution operations that can handle multiple input-output channels and batching, and we use it in our framework to build proofs for (convolutional) neural networks and image processing. We realize a prototype implementation of our proof systems, and show that we achieve up to 5× faster proving time and 10× shorter proofs compared to the state-of-the-art, in addition to asymptotic improvements.

CCS CONCEPTS

• **Security and privacy** → **Cryptography**; *Information-theoretic techniques*; *Privacy-preserving protocols*.

KEYWORDS

Proof Systems; Verifiable Computation; Zero-Knowledge Proofs; Machine Learning; Convolutional Neural Networks; Image Processing.

ACM Reference Format:
David Balbás, Dario Fiore, Maria Isabel González Vasco, Damien Robissout, and Claudio Soriente. 2023. Modular Sumcheck Proofs with Applications to Machine Learning and Image Processing. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23), November 26–30, 2023, Copenhagen, Denmark.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3576915.3623160

1 INTRODUCTION

Cryptographic proof systems can be used in distributed data-processing applications to provide both security and privacy guarantees. This is especially relevant when clients outsource the data processing task to a potentially untrusted server that (i) has enough resources to carry out the computation and optionally (ii) may hold additional data that is required to complete the task but that cannot be shared with clients.

As an example, consider the scenario where a bank owns a machine learning model F that decides credit worthiness $Y = F(X, W)$, given some customer data X and model parameters W. A proof system for this scenario should provide *publicly verifiable* (hence auditable) proofs with strong guarantees for:

- **Integrity**: the prediction is indeed generated by the model, given solely the data provided by the customer and the model parameters. Integrity also guarantees that no bias or unauthorized data—such as gender or race—were used in the computation. This is relevant as the bank (or similar stakeholders) must abide to legal directives that forbid discrimination when providing goods or services [13, 14].
- **Fairness**: if the model is certified by a third-party auditor, customer may obtain guarantees of fair treatment, i.e., the decision process has been the same across all customers. We note that Supreme Audit Institutions have recently defined best-practices to audit ML models and certified ML may be soon available in real-world applications [17, 36].
- **Privacy**: if the model parameters W are proprietary, the bank may publish a (certified) commitment to W while proving that $Y = F(X, W)$ in zero knowledge [20]; this allows the customer to verify that computation was carried out correctly, while W is kept private and nothing is leaked other than what can be inferred by the prediction itself.

Despite the rapid progress in the last decade, general-purpose cryptographic proof systems do not scale well to large inputs. The main bottleneck appears at the prover side, both on running time and memory usage. Among the many families of cryptographic proof systems in the literature, sumcheck-based proof systems [4, 21, 33, 41, 42] achieve the best prover performance (linear on

CCS '23, November 26–30, 2023, Copenhagen, Denmark
© 2023 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0050-7/23/11.
https://doi.org/10.1145/3576915.3623160

the circuit size). Nevertheless, modeling computation as a circuit introduces high overheads that make even these systems impractical when executed on computations that process large amounts of data.

Dedicated proof systems trade-off generality for performance. In particular, they avoid the general circuit encodings of their general-purpose counterparts, and achieve better performance, albeit for restricted classes of functions. For example, previous work has shown how to exploit the sequentiality and low multiplicative depth of some classes of functions to achieve low overhead both for provers and verifiers. For example, vCNN [26] and zkCNN [27] enable verifiable ML applications by exploiting the sequential composition of ML functions where data is processed one layer (i.e, function) at a time and the output of the current layer is fed as input to the next one. The same principle is used by PhotoProof [30] and ZK-IMG [24] that exploit the sequential composition of image processing tasks and provide proof systems tailored to verifiable image processing. Dedicated protocols as described above, however, come at the price of poor composability and leave little room for modification and improvement.

1.1 Contributions

In this work, we aim at solutions combining the best of both worlds: the efficiency of dedicated protocols and the versatility of general-purpose schemes. With this goal in mind, we introduce a new framework for the modular design of sumcheck-based proof systems, and we use it to develop new efficient protocols for verifiable machine learning and image processing. More specifically, our contributions are the following:

A modular framework for sumcheck-based proofs. We develop our framework by identifying and abstracting away the key properties of a variety of proof systems based on the sumcheck protocol, including the well-established GKR protocol [19]. Briefly speaking, these protocols proceed in a layer-by-layer fashion so that at each layer the prover starts by making a "promise" about the output, and later the verifier ends with a "promise" about the input. Their security guarantee is that if the input's "promise" is correct then the initial output's "promise" must be correct too.

We define our framework abstracting these protocols as follows:

- We introduce the notions of *fingerprinting scheme* and *verifiable evaluation scheme on fingerprinted data* (VE). Fingerprinting schemes characterize the aforementioned notion of "promise" and are essentially a mechanism that allows prover and verifier to succinctly represent vectors of inputs/outputs. VEs are interactive protocols in which the verifier works by only knowing fingerprints of inputs and outputs (and thus can run sublinear in the input/output size).
- We show a *generic composition theorem*: given two VEs for functions f_1 and f_2 and compatible fingerprints, one can build a VE for their (partial or total) composition $f(x, y) = f_2(f_1(x), y)$.
- We show that a *VE can be lifted to become an interactive proof* if the verifier computes the fingerprints of the inputs and outputs of the computation (but not of intermediate steps). We also show that a *VE can be compiled into a succinct argument* by using commit-and-prove arguments for the evaluation of fingerprints (instantiatable with polynomial commitments [25]).
- We instantiate our fingerprints as evaluations of multilinear polynomials, and then we show how to capture a large class of existing protocols—such as the multilinear sumcheck protocol of [41], GKR, and the efficient matrix multiplication from [35]—under our framework.

By combining these results, we obtain a way to easily design sumcheck-based proof systems in a modular way. Following the principle of modularity, one needs only to focus on designing VE schemes for specific functionalities, a task that likely results in more lightweight solutions (as we confirm below). In particular, we may take advantage of many years of great research in the field, as our modular design allows us to nicely integrate previous tools and gadgets. Furthermore, the practicality of modular VEs is not only evident at design time, but also at implementation time, since the code can be designed in blocks, in a "Lego" manner.

Applications to verifiable machine learning and image processing. We apply our approach to construct efficient proofs of computation for (convolutional) neural networks and image processing. Both processes have a layered structure that is amenable to our modular framework. Therefore, we build a VE protocol for the full computation by composing several "gadgets" VEs for each layer (including existing and new VEs that we develop – see below), and then we use a multilinear polynomial commitment to compile it into an argument of knowledge. Following the modularity principle, then we focus on designing efficient VEs for the main subroutines needed by these applications.

In this application context, our main contribution is a new VE scheme for convolution operations which is amenable to multiple input-output channels and also to prediction batching. Convolution is a challenging operation in proof systems, as it is represented by arithmetic circuits with complex wiring (and up to $O(n^3)$ size for convolutions over a $n \times n$ matrix) which is expensive for general purpose solutions. The most efficient dedicated protocol in the literature appears in zkCNN [27], which proposes a fast proving technique for Fast Fourier Transform (FFT), achieving asymptotically optimal $O(n^2)$ proving time. Nevertheless, their approach requires proving an FFT, a Hadamard product and an inverse FFT, which increase concrete proof size and prover time. Moreover, in their case the convolution kernel, which is often small in applications, needs to be padded to the input size.

We overcome these limitations by designing a compact matrix encoding of the convolution operation to which we apply the efficient matrix multiplication prover in [35]. Crucially, we optimize our technique to efficiently support multiple channels (both input and output), which is when our solutions improve even more over the zkCNN's approach. Notably, our convolution VE achieves proof size and verifier time that are independent from the input size and the number of output channels.

We obtain further improvements by designing VE gadgets that extend techniques originally proposed in the context of the GKR protocol for arithmetic circuits. Notably, we propose a VE for "many-to-one reductions" for input fingerprints that extends the GKR-specific technique of [47], and we generalize the blueprint from

Hyrax [40] in order to efficiently batch the executions of the same VE on different inputs, e.g., $Y_i = F(X_i)$ for $i = 1$ to N.

Finally, we leverage our framework to construct the first dedicated proof system for recurrent neural networks.

Implementation and evaluation. We implement and benchmark our efficient convolution prover in Rust and confirm the concrete improvements (in overall efficiency and proof size) over the state-of-the-art [27] for common sets of parameters. Even for a single-channel convolution, our VE improves over previous solutions by a factor of 5-10× in proof size, and by a similar factor in prover time for small kernel sizes.

1.2 Additional Related Work

Sumcheck-based proofs. The seminal paper of Goldwasser, Kalai and Rothblum [19] showed how to use the sumcheck protocol [29] to construct a doubly-efficient interactive proof (known as GKR in the literature) for layered arithmetic circuits. Several papers improved the proving time of GKR either in general [10], for circuits with specific structure [35, 39, 50] or through variants of the original protocol [41, 47]. Thaler was the first to show sumcheck-based protocols for specialized computations, such as matrix multiplications, with optimal prover time [35]. Another line of works, started by Zhang et al. [49], showed how to use GKR in combination with polynomial commitments to build (zero-knowledge) argument systems [33, 40, 41, 48]. Arguments based on this approach are among the most efficient ones for proving time, as most of their computational efford relates to an information-theoretic-secure protocol involving only finite field operations. Recent works show how to combine the sumcheck protocol with multilinear polynomial commitments to build succinct non-interactive arguments [4, 21, 42].

Our modular framework is close in the spirit to that of Campanelli, Fiore and Querol [3] who build zk-SNARKs modularly via the efficient composition of specialized commit-and-prove SNARKs. Our techniques work at the information-theoretic level and are based on fingerprints and VE schemes, as opposed to commitments and SNARKs, allowing for a less demanding security notion than computational binding.

Verifiable machine learning. The closest work to this contribution is zkCNN [27] which shows how to exploit the sequential nature of neural networks to build an argument system for their verifiability. Compared to zkCNN, our work improves prover time by showing a faster protocol for convolutions and proposes a general framework that makes it easier to reuse, implement, and improve the components of these protocols. vCNN [26] and ZEN [18] also tackle the problem of zero-knowledge neural network predictions. vCNN combines different commit-and-prove SNARKs to efficiently prove the CNN layers, notably they use quadratic polynomial programs for convolution layers and quadratic arithmetic programs for ReLU and Pooling layers. ZEN presents a quantisation mechanism (based on [23]) for R1CS-based proof systems that achieves significantly less constraints and hence a faster proving time and smaller public parameters. Although we do not directly compare to vCNN and ZEN, we observe that [27] shows that zkCNN is orders of magnitude faster than vCNN and ZEN, and thus we achieve the same improvements. Another related work about zero-knowledge proofs for ML-based predictions is that of Zhang et al. [46], whose techniques are however specialized to decision trees.

Verifiable Image Processing. Besides solutions based on general-purpose zkSNARKs, there are a few works that build specialized proof systems for image processing transformations, notably PhotoProof [30], ZK-IMG [24], and VILS [5].

PhotoProof [30] presents an image authentication framework where images are output by "secure" cameras (i.e., cameras capable of signing images) and Proof-Carrying Data [9] is used to define a set of admissible transformations. The PhotoProof prototype is based on `libsnark` [32] and experiments show that proving one transformation of a 128×128 image takes more than 300 seconds and a public key of a few GBs. ZK-IMG [24] improves over PhotoProof by using `halo2` [45] as the underlying ZK-SNARK system and by showing how to chain proofs of sequential transformations without revealing the intermediate outputs—a feature that may be desirable in scenarios where the input image is private. Performance reported in [24] show that convolution operations can take more than 80 seconds to generate a proof for images of 1280×720 pixels. Finally, VILS [5] takes an alternative approach to authenticated image editing by computing all possible image transformation at the source (i.e., by the secure camera) and accumulating them in a cryptographic accumulator.

Our techniques allow us to obtain a 20× smaller proof size than [24] (albeit not taking into account the opening size of a polynomial commitment, since these are scheme-dependent) and faster prover and verifier times even while running on less powerful hardware.

2 PRELIMINARIES

2.1 Notation

The definitions, games, and constructions that we introduce in our work use standard notation. Algorithms, oracle names, and cryptographic parameters are denoted in sans-serif font. To assign the output of an algorithm Alg on input x to a variable a, we write $a \leftarrow \mathsf{Alg}(x)$. To remark that an algorithm is randomized, we write $a \leftarrow\!\!\$\ \mathsf{Alg}(x)$. An algorithm can input or return blank values, represented by $\perp$. The security parameter is denoted by λ, and its unary representation as 1^λ. In interactive algorithms, we underline steps that involve interaction, such as $\underline{\mathsf{Send}}$ or $\underline{\mathsf{Get}}$.

2.2 Cryptographic Primitives

We define informally the main cryptographic primitives used in our constructions – commitments and (commit-and-prove) arguments of knowledge – and refer to appendix A for more formal definitions.

Commitment schemes allow one to commit to a value (e.g., a scalar, a vector, a polynomial) in a way that is binding and hiding. Binding informally means that a commitment cannot be opened to two distinct values, while hiding guarantees that the commitment reveals no information about the underlying value. In our work, we denote a commitment scheme Com with a tuple of algorithms $(\mathsf{Setup}, \mathsf{Com}, \mathsf{Vf})$ such that: $\mathsf{Setup}(1^\lambda)$ generates the commitment key ck; $\mathsf{Com}(\mathsf{ck}, x)$ outputs a commitment com and an opening o for input value x; $\mathsf{Vf}(\mathsf{ck}, \mathsf{com}, x, o)$ returns a bit b to indicate if o is a valid opening of commitment com to x.

An *argument of knowledge* AoK for an NP relation $\mathcal{R}$ is a tuple of algorithms (Setup, Prove, Vf) such that: $\mathsf{Setup}(1^\lambda, \mathcal{R})$ outputs a common reference string crs; $\mathsf{Prove}(\mathsf{crs}, x, w) \rightarrow \pi$ returns a proof π for $(x, w) \in \mathcal{R}$; $\mathsf{Vf}(\mathsf{crs}, x, \pi)$ accepts or rejects π. An AoK should be *complete* and *knowledge-sound*. The former informally means that honestly generated proofs are accepted by Vf. The latter informally guarantees that any prover producing a valid proof for a statement x must know a valid witness w for it. An AoK is said *succinct* if the total communication between prover and verifier is polylogarithmic in the witness size. AoK satisfies zero-knowledge if proofs leak no information about the witness beyond the truth of the statement (this is modeled through a simulator that can generate valid proofs for a valid statements without knowing the witness).

In our work we use the notion of *commit-and-prove AoKs* for relation $\mathcal{R}$ and commitment scheme Com, which is an AoK for the NP relation $\mathcal{R}_{\mathsf{Com}}$ such that $((x, \mathsf{com}); (u, o, w)) \in \mathcal{R}_{\mathsf{Com}}$ iff $(x, (u, w)) \in \mathcal{R}$ and $\mathsf{Com.Vf}(\mathsf{ck}, \mathsf{com}, u, o) = 1$.

2.3 Proof Systems

We include standard background and definitions on proof systems. In the sequel, let $\mathbb{F}$ be a finite field and ℓ a natural number.

Definition 2.1 (Multilinear extension). Let $f : \{0,1\}^\ell \rightarrow \mathbb{F}$ be a function. The multilinear extension (MLE) $\tilde{f}$ of f is the unique multilinear polynomial $\tilde{f} : \mathbb{F}^\ell \rightarrow \mathbb{F}$ such that $f(\boldsymbol{x}) = \tilde{f}(\boldsymbol{x})$ for all $x \in \{0,1\}^\ell$. It has the following closed form:

$$\tilde{f}(\boldsymbol{x}) = \sum_{\boldsymbol{b} \in \{0,1\}^\ell} \tilde{I}(\boldsymbol{x}, \boldsymbol{b}) \cdot f(\boldsymbol{b})$$

Where $\tilde{I}(\boldsymbol{x}, \boldsymbol{b}) = \prod_{i=1}^{\ell} ((1 - x_i)(1 - b_i) + x_i b_i)$ is the MLE of the *indicator function* $I : \{0,1\}^\ell \times \{0,1\}^\ell \rightarrow \{0,1\}$ such that $I(\boldsymbol{x}, \boldsymbol{b}) = 1$ if $\boldsymbol{x} = \boldsymbol{b}$ and $I(\boldsymbol{x}, \boldsymbol{b}) = 0$ elsewhere.

Lemma 2.2 ([38]). *Given $f(\boldsymbol{x})$ for all $\boldsymbol{x} \in \{0,1\}^\ell$ and a vector $\boldsymbol{r} \in \mathbb{F}^\ell$, the value $\tilde{f}(\boldsymbol{r})$ can be computed in $O(2^\ell)$ time and $O(\ell)$ space.*

For $n \in \mathbb{N}$ and a vector $\boldsymbol{x} \in \mathbb{F}^n$ and $\ell = \lceil \log n \rceil$, there exists a (unique) indexing function $f_{\boldsymbol{x}} : \{0,1\}^\ell \rightarrow \mathbb{F}$ given by $f_{\boldsymbol{x}}(\boldsymbol{b}) = x_i$ where $\boldsymbol{b} = (b_1, \ldots, b_\ell)$ is the binary representation of i. Then, we define the MLE of $\boldsymbol{x}$, that we denote by $\tilde{\boldsymbol{x}} : \mathbb{F}^\ell \rightarrow \mathbb{F}$, as the MLE of the indexing function $f_{\boldsymbol{x}}$.

Definition 2.3 (Interactive Proof). Let $\mathcal{F}$ be a family of functions, and let $\mathcal{L}_{\mathcal{F}} = \{(f, x, y) : f \in \mathcal{F} \wedge f(x) = y\}$ the corresponding language. An interactive proof is a pair of algorithms $b \leftarrow \langle \mathsf{P}, \mathsf{V} \rangle (f, x, y)$ such that the following properties hold:

Completeness: For any $(f, x, y) \in \mathcal{L}_{\mathcal{F}}$,
$\Pr[\langle \mathsf{P}, \mathsf{V} \rangle (f, x, y) \rightarrow 1] = 1$.

ϵ-Soundness: For any algorithm P^* and $(f, x, y) \notin \mathcal{L}_{\mathcal{F}}$,
$\Pr[\langle \mathsf{P}^*, \mathsf{V} \rangle (f, x, y) \rightarrow 1] \leq \epsilon$.

The probabilities are over the random coins of the verifier.

3 COMPOSITION FRAMEWORK FOR INTERACTIVE PROOFS

Our goal in this section is to introduce a framework for building interactive proofs from the composition of function-specific protocols. Our framework consists of three main components: (1) fingerprinting schemes, that are a mechanism with which prover and verifier can succinctly represent inputs and outputs of the computation; (2) verifiable evaluation schemes on fingerprinted data (VE), that are the function-specific protocols in which the verifier works by only knowing fingerprints of inputs and outputs; (3) a composition theorem which shows how to compose VEs, in such a way that the verifier only needs to compute fingerprints for the main input and output of the computation, but not for the intermediate inputs of the sequential steps.

In this section, we define the syntax and the security property of these objects, state and prove the composition and finally also show how to compile a VE scheme into succinct arguments.

Definition 3.1 (Fingerprint). Let $\mathcal{X}$ be a data space, $\mathcal{D}_{\mathcal{X}}$ a distribution over a randomness space $\mathcal{R}_{\mathcal{X}}$, and C a finite set. A randomized fingerprint (with fingerprint space C) is a function $\mathsf{H} : \mathcal{X} \times \mathcal{R}_{\mathcal{X}} \rightarrow C$. Given $x \in \mathcal{X}, r \in \mathcal{R}_{\mathcal{X}}$, we call $c_x \leftarrow \mathsf{H}(x, r)$ the fingerprint of x on r. Furthermore, we say that a fingerprint H is (statistically) sound for $\mathcal{D}_{\mathcal{X}}$ if for any pair $x, x^* \in \mathcal{X}$ such that $x \neq x^*$, we have

$$\Pr_{r \leftarrow\$ \mathcal{R}_{\mathcal{X}}} [\mathsf{H}(x, r) = \mathsf{H}(x^*, r)] = \mathsf{negl}(\lambda).$$

For vectors of inputs $\boldsymbol{x} \in \prod_{i=1}^{M} \mathcal{X}_i$ and randomness $\boldsymbol{r} \in \prod_{i=1}^{M} \mathcal{R}_{\mathcal{X}_i}$, we use the compact notation $\mathsf{H}(\boldsymbol{x}, \boldsymbol{r}) := (\mathsf{H}(x_1, r_1), \ldots, \mathsf{H}(x_M, r_M))$.

The distribution $\mathcal{D}_{\mathcal{X}}$ is an abstraction that allows us to capture sampling (e.g. via a uniform distribution) from a space which is yet undefined. The randomness space $\mathcal{R}_{\mathcal{X}}$ may depend on the data space $\mathcal{X}$ and on the security parameter λ of the scheme, that will generally be implicit. For instance, large domains may require large randomness spaces[1].

Fingerprints and CRHFs. Even though their syntax presents similarities, fingerprints are strictly weaker objects than collision-resistant hash functions (CRHFs). Fingerprints are only guaranteed to be sound if the randomness r is randomly sampled, as opposed to controlled by the adversary. Also, the input x has to be chosen by the adversary before seeing r. The closest notion to our fingerprints are universal hash functions (when instantiated over an exponentially large output space).

3.1 Verifiable Evaluation Schemes on Fingerprinted Data

For our framework we consider a class of interactive proofs for the language $\mathcal{L}_{\mathcal{F}} = \{(f, x, y) : f \in \mathcal{F} \wedge f(x) = y\}$, which have the following structure (cf. Figure 1):

1. Prover and verifier agree on a common fingerprint $c_y = \mathsf{H}(y, r_y)$. As an example, the verifier samples and sends randomness $r_y \leftarrow\$ \mathcal{D}_{\mathcal{Y}}$ to the prover, and both parties compute c_y independently.
2. Prover and verifier interact on common input (f, c_y, r_y) through subroutines $\mathsf{VE.P}(x)$ and $\mathsf{VE.V}(r_x)$ respectively. Notably, neither x nor y are used by the verifier in this part of the interaction. At the end of a successful interaction,

[1] We write $\mathcal{D}_{\mathcal{F}}$ to refer to $\mathcal{D}_{\mathcal{X}}$ when the domain $\mathcal{X}$ is defined by a family of functions $\mathcal{F}$.

both parties agree on a common fingerprint c_x and randomness r_x.

(3) The verifier checks that $c_x = \mathsf{H}(x, r_x)$ and rejects otherwise.

$\langle \mathsf{P}, \mathsf{V}\rangle(f, x, y)$

Prover		**Verifier**
	$\xleftarrow{r_y \leftarrow\!\!\$\ \mathcal{D}_Y}$	
$\mathsf{c}_y \leftarrow \mathsf{H}(y, r_y)$		$\mathsf{c}_y \leftarrow \mathsf{H}(y, r_y)$ $r_x \leftarrow\!\!\$\ \mathcal{D}_X$
c_x, r_x	$\xleftrightarrow{\langle \mathsf{VE.P}(x), \mathsf{VE.V}(r_x)\rangle(f, \mathsf{c}_y, r_y)}$	c_x, b
		$b' \leftarrow [\mathsf{c}_x = \mathsf{H}(x, r_x)]$
		return $b \wedge b'$

Figure 1: Interactive proof constructed from a verifiable evaluation scheme on fingerprinted data VE **and a fingerprinting scheme** H**.**

In other words, these are interactive proofs that manage to reduce the check $f(x) = y$ into a simpler verification that only involves the fingerprints of the output (computed in step (1)) and of the input (computed in step (3)). In this work, we formalize the primitive that takes place in step (2), that we call (interactive) *verifiable evaluation scheme on fingerprinted data* (VE). The goal of a VE scheme is to prove that, given an admissible function f and fingerprints c_x, c_y, then c_x is a valid fingerprint to the input x and c_y is a valid fingerprint to $f(x)$. Contrary to the intuitive setting where the interaction starts with both parties having a common input x (or fingerprint c_x) and finishes on $f(x)$ (or c_y), VE interactions start at a common output fingerprint c_y and finish with both parties agreeing on an input fingerprint c_x.

Definition 3.2. A *verifiable evaluation scheme on fingerprinted data* VE for a family of functions $\mathcal{F}$ is a pair of interactive algorithms (VE.P, VE.V) that, given as prover input $\boldsymbol{x}$; as verifier input randomness $\mathbf{r}_{\boldsymbol{x}}$; and as common input fingerprints $\mathbf{c}_{\boldsymbol{y}}$, randomness $\mathbf{r}_{\boldsymbol{y}}$, and a function $f \in \mathcal{F}$, the interaction outputs

$$(\mathbf{c}_{\boldsymbol{x}}; \mathbf{r}_{\boldsymbol{x}}; b) \leftarrow\!\!\$\ \langle \mathbf{VE.P}(\boldsymbol{x}), \mathbf{VE.V}(\mathbf{r}_{\boldsymbol{x}})\rangle\, (\mathbf{c}_{\boldsymbol{y}}, \mathbf{r}_{\boldsymbol{y}}, f)$$

Where $\mathbf{c}_{\boldsymbol{x}}$ is a common output, $\mathbf{r}_{\boldsymbol{x}}$ a prover output, and b a verifier output. Furthermore, the verifier VE.V is public-coin.

The scheme VE is **correct** if for any valid pair $(f, \boldsymbol{x})$ and randomness $\mathbf{r}_{\boldsymbol{x}}, \mathbf{r}_{\boldsymbol{y}}$, we have that

$$\Pr\left[\begin{array}{l} \mathbf{c}_{\boldsymbol{x}} = \mathbf{H}(\boldsymbol{x}, \mathbf{r}_{\boldsymbol{x}}) \\ \wedge\, b \end{array} \,\middle|\, \begin{array}{l} \mathbf{c}_{\boldsymbol{y}} \leftarrow \mathbf{H}(f(\boldsymbol{x}), \mathbf{r}_{\boldsymbol{y}}) \\ (\mathbf{c}_{\boldsymbol{x}}; \mathbf{r}_{\boldsymbol{x}}; b) \leftarrow\!\!\$\ \langle \mathbf{VE.P}(\boldsymbol{x}), \mathbf{VE.V}(\mathbf{r}_{\boldsymbol{x}})\rangle \\ \hfill (\mathbf{c}_{\boldsymbol{y}}, \mathbf{r}_{\boldsymbol{y}}, f) \end{array}\right] = 1$$

Our definition considers families of functions with multiple inputs and outputs, and also with multiple input-output fingerprints. Inputs and outputs may correspond one-to-one with fingerprints, but it is also possible that several fingerprints (computed on different randomness) correspond to a single input or output. For compactness, we write vectors $\mathbf{c}_{\boldsymbol{x}}, \mathbf{r}_{\boldsymbol{x}}$ (respectively $\mathbf{c}_{\boldsymbol{y}}, \mathbf{r}_{\boldsymbol{y}}$) where $\mathsf{c}_{x,i} \in C$ corresponds to $\mathsf{r}_{x,i} \in \mathcal{R}_X$.

The security that is required for VEs is that, if $\mathbf{c}_{\boldsymbol{x}}$ are valid fingerprints of $\boldsymbol{x}$ and the verifier accepts, then $\mathbf{c}_{\boldsymbol{y}}$ are guaranteed to be valid fingerprints of $f(\boldsymbol{x})$ (except with negligible probability). As we will show later, this property is very useful for composing VEs. We remark that security only holds when the fingerprints of the inputs $\mathbf{c}_{\boldsymbol{x}}$ are honest.

*Definition 3.3 (*VE *Soundness).* A VE scheme VE is statistically (resp. computationally) sound if for any stateful unbounded (resp. PPT) adversary $\mathcal{A}$, the following probability is $\mathsf{negl}(\lambda)$:

$$\Pr\left[\begin{array}{l} \mathbf{c}^*_{\boldsymbol{y}} \neq \mathbf{H}(f(\boldsymbol{x}), \mathbf{r}_{\boldsymbol{y}}) \\ \wedge\, b \end{array} \,\middle|\, \begin{array}{l} \mathbf{r}_{\boldsymbol{x}}, \mathbf{r}_{\boldsymbol{y}} \leftarrow\!\!\$\ \mathcal{D}_{\mathcal{F}} \\ (\mathbf{c}^*_{\boldsymbol{y}}, \boldsymbol{x}, f) \leftarrow \mathcal{A}(\mathbf{r}_{\boldsymbol{y}}) \\ (\mathbf{c}^*_{\boldsymbol{x}}; \mathbf{r}_{\boldsymbol{x}}; b) \leftarrow\!\!\$\ \langle \mathcal{A}(\boldsymbol{x}), \mathbf{VE.V}(\mathbf{r}_{\boldsymbol{x}})\rangle \\ \hfill (\mathbf{c}^*_{\boldsymbol{y}}, \mathbf{r}_{\boldsymbol{y}}, f) \\ \mathbf{c}^*_{\boldsymbol{x}} = \mathbf{H}(\boldsymbol{x}, \mathbf{r}_{\boldsymbol{x}}) \end{array}\right]$$

where the probability is taken over the choices of $\mathbf{r}_{\boldsymbol{x}}, \mathbf{r}_{\boldsymbol{y}}$, the randomness of $\mathcal{A}$ and any additional randomness used by VE.V.

VE security is indeed sufficient for building a sound interactive proof as described in Figure 1. The proof can be found in the full version.

Proposition 3.4. *The protocol in Figure 1 is an interactive proof.*

3.2 Composition of VEs

Next, we show that the composition of VEs that use the same fingerprint scheme is also a VE. This allows for constructions of modular interactive protocols for sequential functions.

Let f be composed of several sub-functions $f_1, \ldots, f_n$, that can place left-to-right in a pipeline fashion (see Figure 2). The high-level approach of this procedure is the following: 1) start on a fingerprint of the output of f (i.e., on the right) that both prover and verifier trust; 2) run the VE schemes for the f_i in a right-to-left order (starting with f_n); while 3) collecting fingerprints to inputs of the f_i obtained throughout the interaction and using them as output fingerprints for sub-functions on the left. At the end of the interaction, the verifier needs to check one or multiple input fingerprints. In Figure 2, we show this procedure in a block diagram.

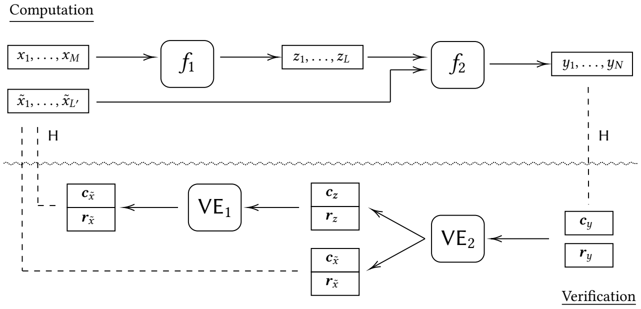

Figure 2: Composition of VEs for functions f_1, f_2 following Proposition 3.5. Top half: computation of $f_2(f_1(\boldsymbol{x}), \bar{\boldsymbol{x}})$, operations are left-to-right. Bottom half: composition of VE$_2$ **and** VE$_1$**, interaction is right-to-left.**

Proposition 3.5 (Composition of VEs). *Let $\mathcal{X} = \prod_{i=1}^{M} \mathcal{X}_i$, $\mathcal{Z} = \prod_{i=1}^{L} \mathcal{Z}_i$, $\bar{\mathcal{X}} = \prod_{i=1}^{L'} \bar{\mathcal{X}}_i$ and $\mathcal{Y} = \prod_{i=1}^{N} \mathcal{Y}_i$ be domains. Let also $f_1 : \mathcal{X} \to \mathcal{Z}$ and $f_2 : \mathcal{Z} \times \bar{\mathcal{X}} \to \mathcal{Y}$. Finally, let $f : \mathcal{X} \times \bar{\mathcal{X}} \to \mathcal{Y}$ be the function given by the (partial) composition $f(\boldsymbol{x}, \bar{\boldsymbol{x}}) := f_2(f_1(\boldsymbol{x}), \bar{\boldsymbol{x}})$.*

Then, given verifiable evaluation schemes VE_1 and VE_2 for f_1 and f_2 based on the same fingerprint scheme, the composition protocol VE obtained by running VE_2 and then VE_1 as in Figure 2 is a verifiable evaluation scheme for f.

The proof is available in the full version of this work. By combining Proposition 3.5 and Proposition 3.4, we obtain a framework for composing arbitrary evaluation schemes for different functions that can be later compiled into an interactive proof. Regarding efficiency, the communication complexity and running time of the resulting protocol grows additively for both prover and verifier, as VEs are run sequentially.

For clarity, in the following sections we use a parametrization for VE schemes that we define as follows.

Definition 3.6 (Parametrization of VEs). A verifiable evaluation scheme VE is parametrized by:

- the fingerprint scheme H,
- the (family of) admissible functions $\mathcal{F} = \{f : \mathcal{X} \to \mathcal{Y}\}$,
- the input and output (vectors of) fingerprints $\mathbf{c}_{\boldsymbol{x}}$, $\mathbf{c}_{\boldsymbol{y}}$,
- the communication complexity $|\pi|$ (of prover messages, i.e., we do not consider verifier challenges)
- the prover and verifier running time t_P, t_V,
- and the soundness ϵ.

3.3 From VEs to Arguments of Knowledge

We show how to turn a VE scheme for $\boldsymbol{y} = f(\boldsymbol{x})$ into a commit-and-prove argument of knowledge for the NP relation

$$\mathcal{R}_\Pi = \{(f, \mathsf{com}_x, \boldsymbol{y}; \boldsymbol{x}, o_x) : f \in \mathcal{F} \wedge f(\boldsymbol{x}) = \boldsymbol{y} \\ \wedge \ \mathsf{Com.Vf}(\mathsf{ck}, \mathsf{com}_x, \boldsymbol{x}, o_x)\}$$

The full scheme is presented in Appendix B, and additional details appear in the full version. The idea is a generalization of the vSQL approach [49] and relies on the observation that in the VE protocol the verifier does not need to know neither $\boldsymbol{x}$ nor $\boldsymbol{y}$ but only their fingerprints $c_{\boldsymbol{x}}, c_{\boldsymbol{y}}$. In the VE-to-IP construction, the verifier would test if $c_{\boldsymbol{x}} = \mathsf{H}(\boldsymbol{x}, r_x)$ and $c_{\boldsymbol{y}} = \mathsf{H}(\boldsymbol{y}, r_y)$. In the AoK, the verifier instead holds the commitment com_x, and we let the prover show the correctness of the fingerprint $c_{\boldsymbol{x}}$ w.r.t. the committed $\boldsymbol{x}$. To enable this proof we only need a commit-and-prove AoK for the computation of H (instantiatable with a multilinear polynomial commitment).

Finally, we observe that, similarly to zkCNN, we can obtain a zero-knowledge AoK for $\mathcal{R}_\Pi$ by using existing approaches [8, 41] based on zero-knowledge sumcheck and low-degree extensions. More precisely, starting from the (non-ZK) VE scheme, we first apply the information-theoretic compiler based on zero-knowledge sumcheck from Libra ([41], Section 4.1). Then, we require a ZK-AoK for H in the compilation to a succinct argument. For the first step, we also need to mask the fingerprints obtained by the verifier to avoid leakage of intermediate values. This can also be done following ([41], Section 4.2).

4 VERIFIABLE EVALUATION FOR MULTILINEAR POLYNOMIALS

In this section, we reinterpret the line of work for the delegation of computation via sumchecks of multilinear polynomials, initiated by the GKR protocol [19] and continued by [11, 35, 41, 47], in the framework introduced in Section 3. We show that the notion of verifiable evaluation scheme captures the soundness properties of these core protocols, and we provide a modular approach such that they are easily composable with function-specific VEs. This allows us to compose these existing protocols with the new VE schemes that we propose in the next section.

First of all, we define a fingerprint based on multilinear extensions. From this point, we adopt the convention that $\lambda = \lfloor \log |\mathbb{F}| \rfloor$ for a field $\mathbb{F}$.

Proposition 4.1. *Let $\mathbb{F}$ be a field, $\tilde{x}$ be the multilinear extension of $\boldsymbol{x} \in \mathbb{F}^n$, and $\ell = \lceil \log n \rceil$. Then, the evaluation of a multilinear extension at a point $\boldsymbol{r} \in \mathbb{F}^\ell$, given by $\tilde{x}(\boldsymbol{r}) \leftarrow \mathsf{H}_{\mathsf{MLE}}(\boldsymbol{x}, \boldsymbol{r})$, is a statistically sound fingerprint for the uniform distribution over $\mathbb{F}^\ell$.*

Proof. Given two inputs $\boldsymbol{x}, \boldsymbol{x}^*$ and $\boldsymbol{r} \leftarrow\!\$\ \mathbb{F}^d$ such that $\boldsymbol{x} \neq \boldsymbol{x}^*$, we have that

$$\Pr[\tilde{x}(\boldsymbol{r}) = \tilde{x}^*(\boldsymbol{r})] = \Pr[(\tilde{x} - \tilde{x}^*)(\boldsymbol{r}) = 0] \leq d/|\mathbb{F}|.$$

where the bound follows by the Schwartz-Zippel lemma. □

Multinear sumcheck VE. The following result is a generalization of the multilinear sumcheck-based delegation schemes in the literature, particularly of those introduced in [35, 41]. The prover time depends on the time required to compute the multilinear extension of each polynomial factor $f_{k,i}$ as described below. Note that when the multilinear sumcheck is described in the VE framework, the function f corresponds to the sum of the evaluations over $\{0,1\}^\ell$, while the polynomial factors $f_{k,i} \in \mathbb{F}[x_1, \ldots, x_\ell]$ correspond to the input and are not necessarily known to the verifier. In most practical cases, $s = 1$ and t is a small constant (such as $t = 2$).

Proposition 4.2. *Let $\boldsymbol{x}$ be a vector of ℓ variables, $\mathbb{F}$ a finite field and $\alpha_i \in \mathbb{F}$ for $i = 1, \ldots, s$. Let also*

$$f(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{s} \alpha_i \prod_{k=1}^{t} f_{k,i}(\boldsymbol{x}_{k,i}, \boldsymbol{y})$$

where each factor $f_{k,i}$ is a multilinear polynomial over $\mathbb{F}$ evaluated on a subvector $\boldsymbol{x}_{k,i} \subset \boldsymbol{x}$. Then, the multilinear sumcheck protocol $\mathsf{VE}_{\mathsf{ML}}$ in Figure 3 is a MLE-based VE scheme for the relation

$$f_y(\boldsymbol{r}_y) = \sum_{\boldsymbol{x} \in \{0,1\}^\ell} f(\boldsymbol{x}, \boldsymbol{r}_y).$$

$\mathsf{VE}_{\mathsf{SC}}$ is parametrized by one output fingerprint $\mathsf{c}_{f_y} = f_y(\boldsymbol{r}_y)$, $s \cdot t$ input fingerprints $\mathsf{c}_{k,i} = f_{k,i}(\boldsymbol{r}_{k,i}, \boldsymbol{r}_y)$ where each $\boldsymbol{r}_{k,i} \subset \boldsymbol{r} \in \mathbb{F}^\ell$, communication complexity $|\pi| = (\ell + s) \cdot t \cdot \lambda$, verification time $\mathsf{t}_\mathsf{V} = O(t \cdot \ell)$, and soundness $\epsilon = t\ell/|\mathbb{F}|$. Furthermore, given that $\tau_{k,i}$ is the time required to compute the MLE of $f_{k,i}(\boldsymbol{x}_{k,i}, \cdot)$, the prover time is $\mathsf{t}_\mathsf{P} = O\big(s \cdot t^2 \cdot \max_{k,i} \tau_{k,i}\big)$.

Proof. First, we recall that the sumcheck protocol over a field $\mathbb{F}$ for a ℓ-variate polynomial of degree t has soundness $t\ell/|\mathbb{F}|$ [29].

Correctness, communication complexity and efficiency follow from inspection of Figure 3 and from the efficient sumcheck and padding techniques in previous work [41, 47]. For soundness, consider a successful adversary against VE soundness that, given an output fingerprint $\mathsf{c}^*_{f_y} \neq f_y(\boldsymbol{r}_y)$, makes $\mathsf{VE_{SC}.V}$ accept. Let also $g'_1(x_1), \dots, g'_\ell(x_\ell)$ be the sequence of degree t polynomials that correspond to running the protocol honestly, in addition to the constant polynomial $g'_0 = f_y(\boldsymbol{r}_y)$. By definition of VE soundness, we have that all input fingerprints are honestly computed, i.e., $\mathsf{c}_{k,i} = f_{k,i}(\boldsymbol{r})$ for every k, i. Therefore, as the check in line 12 of Figure 3 verifies, it must be that $\hat{g}_\ell(r_\ell) = g'_\ell(r_\ell)$. We conclude that the adversary must have found a collision during the sumcheck, which occurs with probability $\epsilon = t\ell/|\mathbb{F}|$. □

```
$\mathsf{VE_{SC}.P}(\mathsf{c}_{f_y}, r_y, f)$                                  $\mathsf{VE_{SC}.V}(\mathsf{c}_{f_y}, r_y, \boldsymbol{r})$
01 Evaluate $f_{k,i}(\boldsymbol{x}_{k,i}, \boldsymbol{r}_y)$ for all $k, i$.
02                                                     $\hat{g}_0 \leftarrow \mathsf{c}_{f_y}$.
03 for $j = 1 \dots \ell$ :
04   for $d = 0 \dots t$ :
05     $m_{j,d} \leftarrow \sum_{\boldsymbol{b} \in \{0,1\}^{\ell-j}} \sum_{i=1}^{s} \alpha_i \prod_{k=1}^{t} f_{k,i}(r_1, \dots, r_{j-1}, d, \boldsymbol{b}, \boldsymbol{r}_y)$
06   Send $\boldsymbol{m}_j = (m_{j,0}, \dots, m_{j,t}) \in \mathbb{F}^{t+1}$
07                                   Interpolate $\hat{g}_{j-1}$ from $\boldsymbol{m}_{j-1}$
08                                   Check $[\hat{g}_{j-1}(r_{j-1}) = m_{j,0} + m_{j,1}]$
09                                                     Send $r_j$
                          Final round:
10 Send $\mathsf{c}_{k,i} = f_{k,i}(\boldsymbol{r}_{k,i}, \boldsymbol{r}_y)$, for all $k, i$.
11                                   Interpolate $\hat{g}_\ell$ from $\boldsymbol{m}_\ell$
12                      Check $\left[\hat{g}_\ell(r_\ell) = \sum_{i=1}^{s} \alpha_i \prod_{k=1}^{t} \mathsf{c}_{k,i}\right]$
13                                   Set $b \leftarrow 1$ if all checks pass
  Output $(\{\mathsf{c}_{k,i}\}_{k,i}, \boldsymbol{r})$                        Output $(\{\mathsf{c}_{k,i}\}_{k,i}, b)$
```

Figure 3: Multilinear sumcheck protocol $\mathsf{VE_{SC}}$.

4.1 VE for GKR layers

In the celebrated GKR protocol [19], prover and verifier interact in a series of sumchecks that take place at every layer of the circuit. Each of these sumchecks can be written as a VE scheme with multiple input fingerprints (and possibly multiple output fingerprints too). This interpretation is straightforward following Proposition 4.2; it also addresses the observation that the add and mult gate predicates can be replaced by alternative gate predicates, in order to support other operations efficiently as mentioned in [41], or larger fan-in such as in [27].

Following the notation from Libra [41], we write V_i for the output values at the gates of the circuit at layer i (interpreted as a function $V_i : \{0,1\}^{\ell_i} \to \mathbb{F}$) and $\tilde{V}_i$ its multilinear extension. We define the wiring predicates $\mathsf{add}_i, \mathsf{mult}_i : \{0,1\}^{\ell_i + 2\ell_{i-1}} \to \mathbb{F}$, which take one gate label $y \in \{0,1\}^{\ell_i}$ and two gate labels $x_1, x_2 \in \{0,1\}^{\ell_{i-1}}$, and output 1 if gate y is an addition (respectively a multiplication) gate that takes the outputs from gates x_1, x_2 in the previous layer. Therefore, for any $y \in \{0,1\}^{\ell_i}$, we can write V_{i+1} as

$$V_{i+1}(y) = \sum_{x_1, x_2 \in \{0,1\}^{\ell_i}} \mathsf{add}_i(y, x_1, x_2)(V_i(x_1) + V_i(x_2)) + \mathsf{mult}_i(y, x_1, x_2)(V_i(x_1) \cdot V_i(x_2)). \quad (1)$$

In the protocol, prover and verifier start on a common fingerprint of the output $V_{i+1}(\boldsymbol{r}_y)$ and then run the multilinear sumcheck from Figure 3. At the end of the sumcheck, in which the prover sends a total of $2 \cdot \ell_i$ polynomials, the verifier needs to check the consistency of the prover's claims by using the wiring predicates. Namely, it needs to compute (or re-use in a layer above) the following fingerprints: $\tilde{V}_i(\boldsymbol{r}_1)$, $\tilde{V}_i(\boldsymbol{r}_2)$, $\tilde{\mathsf{add}}_i(\boldsymbol{r}_y, \boldsymbol{r}_1, \boldsymbol{r}_2)$, $\tilde{\mathsf{mult}}_i(\boldsymbol{r}_y, \boldsymbol{r}_1, \boldsymbol{r}_2)$.

The following result is a reinterpretation of [41], and in particular the observation that the prover time is linear in 2^ℓ where $\ell = \max\{\ell_i, \ell_{i+1}\}$ due to the sparsity of $\tilde{\mathsf{add}}_i$, $\tilde{\mathsf{mult}}_i$ and Lemma 2.2. The proof follows from Proposition 4.2.

Proposition 4.3. *The interactive protocol that takes place at a GKR layer is a VE scheme $\mathsf{VE_{GKR}}$ for all functions computable by a single-layered arithmetic circuit with gates of fan-in 2. The scheme is parametrized by 1 output fingerprint (of V_{i+1}), 4 input fingerprints (2 of V_i, 1 of add_i, 1 of mult_i), communication complexity $|\pi| = (3 \cdot \ell + 4) \cdot \lambda$, prover time $\mathsf{t_P} = O(2^\ell)$, verifier time $\mathsf{t_V} = O(\ell)$, and soundness $\epsilon = 2\ell/|\mathbb{F}|$.*

4.2 VE for Many-to-One Reductions

Multivariate sumcheck-based VEs often present the issue that, from a single output fingerprint, the interaction yields multiple input fingerprints to be checked by the verifier at a later time. For GKR layers, the two input fingerprints of V_i obtained shall be used as output fingerprint for layer $i-1$. To avoid an exponential blow-up on the number of fingerprints to be checked, the original GKR protocol proposes a 2-to-1 reduction protocol that, given two fingerprints of any $\boldsymbol{x}$, it reduces them to a single fingerprint. An alternative to the 2-to-1 reduction is to use a random linear combination on the sum [8].

Below we formalize 2-to-1 reductions in the VE framework and generalize it to a m-to-1 reduction. The result extends GKR-specific techniques from Virgo++ [47].

Proposition 4.4. *Let $\boldsymbol{x} \in \mathbb{F}^n$ and let $\tilde{\boldsymbol{x}}(\boldsymbol{r}_1), \dots, \tilde{\boldsymbol{x}}(\boldsymbol{r}_m)$ be MLE fingerprints on $\boldsymbol{r}_i \in \mathbb{F}^\ell$. Let also $\alpha_i \in \mathbb{F}$ for $i = 1, \dots, m$, let $I(\boldsymbol{u}, \boldsymbol{v})$ be the indicator function on the boolean hypercube such that $I(\boldsymbol{u}, \boldsymbol{v}) = 1$ if $\boldsymbol{u} = \boldsymbol{v}$ and is zero elsewhere, and define*

$$f(\boldsymbol{y}) = \sum_{i=1}^{m} \alpha_i \cdot \boldsymbol{x}(\boldsymbol{r}_i) = \left(\sum_{i=1}^{m} \alpha_i \cdot \tilde{I}(\boldsymbol{r}_i, \boldsymbol{y}) \right) \cdot \tilde{\boldsymbol{x}}(\boldsymbol{y}).$$

Then, running the multilinear sumcheck protocol from Figure 3 on $f(\boldsymbol{y})$ yields a VE scheme $\mathsf{VE_{m\text{-}1}}$ parametrized by m output fingerprints $\tilde{\boldsymbol{x}}(\boldsymbol{r}_i)$, $m+1$ input fingerprints ($I(\boldsymbol{r}_i, \boldsymbol{r}_y)$ for $i = 1, \dots, m$ and $\tilde{\boldsymbol{x}}(\boldsymbol{r}_y)$), communication complexity $|\pi| = (3 \cdot \ell + m + 1) \cdot \lambda$, prover time $\mathsf{t_P} = O(m \cdot 2^\ell)$, verifier time $\mathsf{t_V} = O(m + \ell)$, and soundness $\epsilon = (2\ell + 1)/|\mathbb{F}|$.

Note the additional soundness loss of $1/|\mathbb{F}|$ with respect to the sumcheck, which comes from the choice of the α_i. It is straightforward to express the random linear combination approach from [8] as a VE, also following Proposition 4.2. Such VE is parametrized by 2 input fingerprints (of V_{i+1}), and 6 output fingerprints (2 of V_i, 2 of add_i, 2 of mult_i).

Evaluation of mult*,* add *and structured predicates.* In all VEs introduced so far, including those in Proposition 4.3 and 4.4, the number of input fingerprints is larger than the number of output fingerprints. Some of these fingerprints correspond to unstructured data (such as the values at a circuit layer or an external input), but most of them have a regular structure such as wiring predicates mult, add and indicator functions.

When multiple VEs are composed, fingerprints coming from structured data may be checked directly by the verifier, as opposed to plugged into other VEs. There exist essentially two design choices available:

- The verifier recomputes the multilinear extensions on its own. In many cases, one can benefit from parallelism [10], or from sparsity [41]. In [22], it is shown that most *simple* predicates (those expressible as read-only branching programs), including many regular wiring patterns such as indicator functions, can be evaluated in logarithmic time (i.e. polynomial in ℓ).
- The verifier performs a pre-processing phase or relies on a trusted third party to compute (multilinear) polynomial commitments to the data. Then, the prover provides an opening proof on the required point. In this setting, the evaluation is outsourced to the prover, similarly to what is done for instance in Spartan [33].

4.3 Efficient Matrix Multiplication

Among the protocols that we can capture in our framework, a notable example is the efficient interactive protocol for matrix multiplication from [35]. The main idea of the protocol is to express the product of two matrices $C = A \cdot B$ where $A, B, C \in \mathbb{F}^{n\times n}$ as a polynomial identity as

$$C(\boldsymbol{x}_1, \boldsymbol{x}_2) = \sum_{\boldsymbol{y}\in\{0,1\}^\ell} A(\boldsymbol{x}_1, \boldsymbol{y}) \cdot B(\boldsymbol{y}, \boldsymbol{x}_2) \qquad (2)$$

Then, the interaction follows the sumcheck in Figure 3. Namely, given $\boldsymbol{r}_1, \boldsymbol{r}_2 \in \mathbb{F}^\ell$, both parties carry out a sumcheck over

$$\tilde{C}(\boldsymbol{r}_1, \boldsymbol{r}_2) = \sum_{\boldsymbol{y}\in\{0,1\}^\ell} \tilde{A}(\boldsymbol{r}_1, \boldsymbol{y}) \cdot \tilde{B}(\boldsymbol{y}, \boldsymbol{r}_2). \qquad (3)$$

The protocol is therefore a VE scheme parametrized by two input fingerprints $\tilde{A}(\boldsymbol{r}_1, \boldsymbol{r}_3), \tilde{B}(\boldsymbol{r}_3, \boldsymbol{r}_2)$, an output fingerprint $\tilde{C}(\boldsymbol{r}_1, \boldsymbol{r}_2)$, communication complexity $|\pi| = (3 \cdot \ell + 2) \cdot \lambda$, prover time $\mathsf{t_P} = O(n^2)$, verifier time $\mathsf{t_V} = O(\ell)$ and soundness $\epsilon = 2\ell/|\mathbb{F}|$.

5 VERIFIABLE EVALUATION FOR MACHINE LEARNING

In this section, we introduce efficient proofs for common ML operations, following our VE framework. We focus on Convolutional Neural Networks (CNNs) though we note that many of these operations are also usual in image processing. We start by introducing ML preliminaries.

5.1 Preliminaries

5.1.1 CNNs. A Convolutional Neural Network (CNN) is a layered model where the initial input X is transformed sequentially from layer to layer. Let $X = X^{(1)}$ be the array of input values and $\{X^{(k)}\}_{k=1}^L$ the intermediate values between layers, as defined before. Each $X^{(k)} \in \mathbb{F}^{c^{(k)}\times n^{(k)}\times n^{(k)}}$, where $c^{(k)}$ is the number of channels at layer k, and $n^{(k)} \times n^{(k)}$ is the dimension of the arrays at layer k. Namely, at each intermediate layer we have $c^{(k)}$ "parallel" arrays of the same size. An example of multiple channels in an input layer is a coloured image, which commonly has 3 channels: the red, blue, and green values of each pixel.

CNNs apply layer functions $f^{(k)}$ sequentially, such that $X^{(k+1)} = f^{(k)}(X^{(k)}, W^{(k)})$. Usually, models interleave linear layers, such as convolutional layers and fully connected layers, and nonlinear layers such as ReLU and Pooling. At some of these layers, including convolutional layers, we have parameters $W^{(k)}$ (aka weights). For convolutional layers, these are $c^{(k)} \times c^{(k+1)}$ matrices of size $m^{(k)} \times m^{(k)}$. We denote each of these matrices as $W^{(k)}_{\sigma,\tau}$ where $\sigma \in \{0, \dots, c^{(k)} - 1\}$ and $\tau \in \{0, \dots, c^{(k+1)} - 1\}$.

5.1.2 Convolution. The equation of a plain 2D convolution[2] in a CNN for a given output channel τ is

$$X^{(k+1)}_\tau[u, v] = \sum_{\sigma=0}^{c^{(k)}-1} \sum_{i,j=0}^{m^{(k)}-1} X^{(k)}_\sigma[u+i, v+j] \cdot W^{(k)}_{\sigma,\tau}[i, j]. \qquad (4)$$

If no padding and strides (i.e. "jumps" in the convolution) are applied, the output matrix $X^{(k+1)}_\tau$ is a square matrix of size $n^{(k+1)} \times n^{(k+1)}$ where $n^{(k+1)} = n^{(k)} - m^{(k)} + 1$. It is very common in practice to apply a zero or mirror padding such that $n^{(k)} = n^{(k+1)}$. Convolutions can be carried out via (naive) dot products, via Fast Fourier Transforms (FFTs), via polynomial multiplication, or via matrix multiplication[3].

A related common operation is *transposed convolution*, which is an upsampling operation that increases the size of the output with respect to the input. We refer to [15] for a good introduction to convolution arithmetic.

In the full version, we briefly discuss other relevant layer types; namely, activation, pooling, fully connected and batch normalization.

5.1.3 Quantisation. Generally, CNNs need to be quantised to be embedded in proof systems, since these require that values belong to some finite field. Quantisation is actually used beyond verification, as typical models reach a similar accuracy on short integers (such as 8-bit). A usual quantization scheme is [23], which, as shown in zkCNN [28], can be integrated into large fields easily. A possible avenue for building verifiable CNNs without quantisation consists of using proof systems with native ring arithmetic such as [6, 34].

[2] Note that 1D, 2D and 3D convolutions are equivalent in practice if the arrays are arranged adequately.

[3] It may seem that FFTs are best-performing, but in some practical cases [7] matrix multiplication is actually preferred.

5.2 Our VE for Convolution

In this section we present a novel approach to proving convolutions efficiently by exploiting the symmetrical structure of a convolution operation. We write convolutions as matrix multiplications, seeking a more convenient form than the commonly used Toeplitz or circulant matrices (see [37] for further details).

Rewriting convolution. We observe that it is possible to re-write a convolution operation in the following compact form, where we specify a convolution of a 3×3 input X by a 2×2 kernel W.

$$\begin{bmatrix} x_0 & x_1 & x_3 & x_4 \\ x_1 & x_2 & x_4 & x_5 \\ x_3 & x_4 & x_6 & x_7 \\ x_4 & x_5 & x_7 & x_8 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_3 \\ w_4 \end{bmatrix} = \begin{bmatrix} w_0x_0 + w_1x_1 + w_3x_3 + w_4x_4 \\ w_0x_1 + w_1x_2 + w_3x_4 + w_4x_5 \\ w_0x_3 + w_1x_4 + w_3x_6 + w_4x_7 \\ w_0x_4 + w_1x_5 + w_3x_7 + w_4x_8 \end{bmatrix} \tag{5}$$

The example is easily extended to an $n_x \times n_x$ input and $m \times m$ kernel. The matrix on the left-hand side has dimensions[4] $(n - m + 1)^2 \times m^2$. More generically, this is the dimension of the flattened output times the dimension of the flattened weight matrix, which is $n_y^2 \times m^2$ for a convolutional layer that has an output of size $n_y \times n_y$.

We can extend this approach to capture multiple channels in a convolutional neural network. Let us recover usual CNN notation while ignoring layer indices; let X_σ be the input with channel $\sigma \in [c]$, and let $W_{\sigma,\tau}$ be the weight matrix where $\tau \in [d]$ is the output channel. Then, in matrix form (where $\hat{X}, \hat{W}$ are the transformed matrix representations of the data and weights in the form of Equation 5), we have that the layer's output Y is given by

$$Y = [Y_1 | \cdots | Y_d] = \sum_{\sigma=1}^{c} \hat{X}_\sigma \cdot [\hat{W}_{\sigma,1} | \cdots | \hat{W}_{\sigma,d}]. \tag{6}$$

Namely, for each input channel σ we have the product of a $(n_y)^2 \times m^2$ matrix and a $m^2 \times d$ matrix. Each Y_τ is a column vector of length n_y^2 (i.e., a flattened channel of the output of the layer). If we apply the efficient VE for matrix multplication at this stage, we need to prove the result of a sum of c matrix multiplications, where the size of the matrices is $(n_y)^2 \times m^2$ and $m^2 \times d$.

Combining all input channels. The main efficiency advantage of our approach is that it is straightforward to extend the sumcheck equation for matrix multiplication (eq. (3)) to sum over the multiple channels. To do this, we can encode both $\hat{X}$ and $\hat{W}$ as trivariate polynomials given by $\hat{X}(x, y, \sigma) := \hat{X}_\sigma(x, y)$ and $\hat{W}(x, y, \sigma) := \hat{W}_\sigma(x, y)$ for every $\sigma \in [c]$. Then, we obtain the following sumcheck equation over $\boldsymbol{x}_1, \boldsymbol{x}_2$

$$\tilde{Y}(\boldsymbol{y}_1, \boldsymbol{y}_2) = \sum_{\substack{(\boldsymbol{x}_1,\boldsymbol{x}_2)\in \\ \{0,1\}^{2\lceil \log m\rceil + \lceil \log c\rceil}}} \tilde{X}(\boldsymbol{y}_1, \boldsymbol{x}_1, \boldsymbol{x}_2) \cdot \tilde{W}(\boldsymbol{x}_1, \boldsymbol{y}_2, \boldsymbol{x}_2). \tag{7}$$

Proposition 5.1. *Let* $\mathsf{VE}_{\mathsf{conv}}$ *be the VE scheme for two-dimensional convolution that is obtained by running the multivariate sumcheck protocol in Figure 3 on Equation 7. Then,* $\mathsf{VE}_{\mathsf{conv}}$ *is parametrized by two input fingerprints (one for* $\hat{X}$ *and one for* $\hat{W}$*), one output fingerprint (for* Y*), communication complexity* $|\pi| = (3 \cdot (2\lceil \log m\rceil + \lceil \log c\rceil) + 2) \cdot \lambda$*, prover time* $\mathsf{t_P} = O\Big(c(n_y^2 m^2 + m^2 d)\Big)$*, verifier time* $\mathsf{t_V} = O\big(\log(cm^2)\big)$*, and soundness* $\epsilon = 2 \cdot (2\lceil \log m\rceil + \lceil \log c\rceil)/|\mathbb{F}|$.

Intuitively, the asymptotic benefit of our approach compared to previous work is essentially given by expressing the input channels in columns in eq. (6), avoiding the overhead of padding the kernels to the input size.

We also note that it is straightforward to extend equation 5 to support arbitrary padding or stride settings by modifying the reshaped input $\hat{X}$, as done in our implementation. An advantage of our method is that the output Y does not need to be reshaped after the VE is applied.

5.2.1 Transpose Convolution. The transpose convolution operation can be re-written as in Equation 5. For an example, let $m = n_x = 2$ over a single input channel $X_\sigma^{(k)}$. A basic upscaling transposed convolution yields $n_y = 3$ as below.

$$\begin{bmatrix} 0 & 0 & 0 & x_0 \\ 0 & 0 & x_0 & x_1 \\ 0 & 0 & x_1 & 0 \\ 0 & x_0 & 0 & x_2 \\ x_0 & x_1 & x_2 & x_3 \\ x_1 & 0 & x_3 & 0 \\ 0 & x_2 & 0 & 0 \\ x_2 & x_3 & 0 & 0 \\ x_3 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \end{bmatrix} = \begin{bmatrix} x_0w_3 \\ x_0w_2 + x_1w_3 \\ x_1w_2 \\ x_0w_1 + x_2w_3 \\ \sum_{i=0}^{3} x_iw_i \\ x_1w_0 + x_3w_2 \\ x_2w_1 \\ x_2w_0 + x_3w_1 \\ x_3w_0 \end{bmatrix} \tag{8}$$

For arbitrary input channels, the output will be a $n_y^2 \times d$ matrix. As before, we need to compute the sum over all input channels $\sigma \in [c]$, which can be done by extending the sumcheck as in Equation 7. This yields a prover time of $\mathsf{t_P} = O\Big(c(n_y^2 m^2 + m^2 d)\Big)$ and a verifier time of $\mathsf{t_V} = O\big(\log(cm^2)\big)$, exactly as for convolutions.

5.3 Neural Network Layers

Neural networks, and in general many data processing algorithms, incorporate several (generally simple) steps beyond convolution. We succinctly describe efficient ways of constructing VEs for the most usual operations.

5.3.1 Layer reshaping and pooling. For any sequence of operations that can be expressed without any multiplication gate (such as padding, rotation, compression, averaging, or any input rearrangement –e.g., the pre-processing required for the input of $\mathsf{VE}_{\mathsf{conv}}$), one can encode the desired pattern in a wiring predicate $P(\boldsymbol{x}, \boldsymbol{y})$ and apply the multilinear sumcheck $\mathsf{VE}_{\mathsf{SC}}$ as follows. For an input layer X and output layer Y, let $P(\boldsymbol{x}, \boldsymbol{y}) = t$ if the value $t \cdot X(\boldsymbol{x})$ is added to $Y(\boldsymbol{y})$. Then, $\mathsf{VE}_{\mathsf{SC}}$ can be applied over $Y(\boldsymbol{y}) = \sum_{x \in \{0,1\}^\ell} P(\boldsymbol{x}, \boldsymbol{y}) \cdot X(\boldsymbol{x})$. Note that $\tilde{P}$ is sparse for most operations (except for weighted sums of many input values).

The predicate P natively supports average pooling. For max pooling, we recall the approach using auxiliary bit decompositions by zkCNN [27], that can be expressed as a VE. We also note that the described VE for reshaping can be easily used in combination with a many-to-one VE.

[4] In this explanation, we are ignoring padding and stride parameters.

5.3.2 Normalization and linear transformations. Point-wise normalization, and in general input re-scaling operations that can be expressed as linear transformations of the form $x \mapsto \alpha x + \beta$, where $\alpha, \beta \in \mathbb{F}$, can be verified via a linear shift without any prover work. Indeed, multilinear fingerprints satisfy that given $X, Y \in \mathbb{F}^n$ such that $Y(\boldsymbol{x}) = \alpha \cdot X(\boldsymbol{x}) + \beta$ for all $\boldsymbol{x} \in \{0,1\}^\ell$, then $c_Y = \tilde{Y}(\boldsymbol{r}) = \alpha \cdot \tilde{X}(\boldsymbol{r}) + \beta$.

5.3.3 Activation functions. Due to their non-linearity, the verification of activation layers is particularly challenging and essentially reduces to two possibilities:

- Dedicated VEs with additional input. For instance, zkCNN [27] introduces a protocol for ReLU that requires additional bit decomposition, and can be easily seen as a VE.
- Approximate activation functions via polynomials, as is usual in the privacy-preserving ML literature. Quadratic polynomials may already offer good approximations [1]. For this approach, one can construct a VE that evaluates quadratic polynomials via the following multilinear sumcheck (which follows from a GKR-like encoding):

$$\tilde{Y}(\boldsymbol{y}) = \sum_{\boldsymbol{x}_1, \boldsymbol{x}_2 \in \{0,1\}^\ell} \tilde{I}(\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{y}) \cdot \tilde{X}(\boldsymbol{x}_1) \cdot \tilde{X}(\boldsymbol{x}_2)$$

  For a degree d polynomial, it is possible to use a binary tree of multiplications, such that prover time, verifier time, and communication complexity scale with $\log d$.

An alternative approach is using efficient lookup arguments [16, 31, 43, 44], where one can benefit from storing all values of the activation function (for quantised inputs) in a lookup table. We leave the investigation of lookups in our VE framework as interesting future work.

5.4 Neural Networks

To construct a dedicated proof system for neural networks, we build a large VE scheme (denoted by $\mathsf{VE}_{\mathsf{NN}}$), composed by several "gadget" VE_{k} for each of the layers of the network. Then, we use a multilinear polynomial commitment scheme to build a commit-and-prove AoK that achieves succinctness and efficient verification, following the blueprint of Proposition 3.5.

Following previous notation, let $X^{(k)}$ be the input and $f^{(k)}$ the function at layer k. We consider two general kinds of layers:

- Layers $f^{(k)}(X^{(k)})$ that apply an input transformation without additional parameters. For such $f^{(k)}$ we consider VE_{k} that take output fingerprints $\boldsymbol{c}_X^{(k+1)}$ (on randomness $\boldsymbol{r}_X^{(k+1)}$) and produce input fingerprints $\boldsymbol{c}_X^{(k)}$ (on randomness $\boldsymbol{r}_X^{(k)}$) and a (possibly empty) vector of fingerprints $\boldsymbol{c}_P^{(k)}$ (on randomness $\boldsymbol{r}_P^{(k+1)}$) to an auxiliary predicate P (see below).
- Layers $f^{(k)}(X^{(k)}, W^{(k)})$ that require additional parameters, not necessarily known to the verifier. For these functions, we consider VE_{k} that take output fingerprints $(\boldsymbol{c}_X^{(k+1)}, \boldsymbol{r}_X^{(k+1)})$ and produce input fingerprints $(\boldsymbol{c}_X^{(k)}, \boldsymbol{c}_W^{(k)}, \boldsymbol{c}_P^{(k)}, \boldsymbol{r}_X^{(k)}, \boldsymbol{r}_W^{(k)}, \boldsymbol{r}_P^{(k)})$.

Additionally, we require VE_{k} to take as many output fingerprints to $X^{(k+1)}$ as input fingerprints produced by $\mathsf{VE}_{\mathsf{k+1}}$, such that they are compatible. Note, we can always achieve compatibility as one can reduce input fingerprints by applying $\mathsf{VE}_{\mathsf{m-1}}$ (Proposition 4.4).

The predicates $P^{(k)}$ englobe any additional predicate that expresses the circuit at each layer, such as the wiring predicates in Equation (1) or additional auxiliary input as in [27]. For both $P^{(k)}$ and $W^{(k)}$, we define $W(\boldsymbol{k}, \boldsymbol{x}) := W^{(k)}(\boldsymbol{x})$ and $P(\boldsymbol{k}, \boldsymbol{x}) := P^{(k)}(\boldsymbol{x})$ via interpolation as

$$T(\boldsymbol{x}_k, \boldsymbol{x}) = \sum_{k=0}^{L-1} I(\boldsymbol{x}_k, k) \cdot T^{(k)}(\boldsymbol{x}) \tag{9}$$

where $T \in \{X, W\}$ and $I(\boldsymbol{x}_k, k)$ is the indicator function on $\lceil \log L \rceil$ variables. Without loss of generality, we pad every $T^{(k)}$ to have the same number of variables. For concrete implementations, it is possible to optimize the padding.

We describe $\mathsf{VE}_{\mathsf{NN}}$ and its compiled AoK Π_{NN} in Figure 4. Soundness of $\mathsf{VE}_{\mathsf{NN}}$ follows by Proposition 3.5 and the soundness of VE_{k} and $\mathsf{VE}_{\mathsf{m-1}}$. Π_{NN} is an instantiation of the compiler of Section 3.3. By expressing the model parameters and predicates as single polynomials, it is possible to obtain, via many-to-one reductions, a single input fingerprint for each of $X := X^{(0)}$, W, and P. These fingerprints are verified in $\Pi_{\mathsf{NN}}.\mathsf{V}$ by three polynomial commitment opening proofs.

Proposition 5.2. *The protocol $\mathsf{VE}_{\mathsf{NN}}$ is a VE scheme for a neural network architecture $F_{\mathsf{NN},P}$, parameterized by 1 output fingerprint (of $\boldsymbol{y}$), and 3 input fingerprints (of $X^{(0)}$, W, and P). Communication complexity, prover time, verifier time, and soundness result from the sum of the respective parameters of each VE_{k} and $\mathsf{VE}_{\mathsf{m-1}}$ on Figure 4. Besides, Π_{NN} is an argument of knowledge for the relation*

$$\begin{aligned}\mathcal{R}_{\mathsf{NN}} = \{&(\mathsf{com}_X, \mathsf{com}_W, \mathsf{com}_P, \boldsymbol{y}; X, W, P, o_X, o_W, o_P) : \\ &F_{\mathsf{NN},P}(X, W) = \boldsymbol{y} \wedge \mathsf{Com.Vf}(\mathsf{ck}, \mathsf{com}_T, T, o_T), \forall T \in \{X, W, P\}\}.\end{aligned}$$

Finally, we remark that our modular approach allows verifying pre- or post-processing operations in addition to the model, such as an aggregation phase. In this case, one can extend $\mathsf{VE}_{\mathsf{NN}}$ and compose it with additional VE schemes for these operations.

5.5 Proof Batching

Our techniques are amenable to efficient batching where many evaluations $Y_i = F(X_i, W)$ for $i = 1, \ldots, N$ are verified in a single step. For VE schemes that rely on the multilinear sumcheck protocol from Figure 3, including the convolution VE introduced in this section, it is possible to reduce the verification time and communication complexity from linear to constant in the number of instances N.

Let $X(\boldsymbol{i}, \boldsymbol{x}) \in \mathbb{F}[X_1, \ldots X_{\log N + \ell_x}]$ be defined by $X(\boldsymbol{i}, \boldsymbol{x}) := X_i(\boldsymbol{x})$, and let $Y(\boldsymbol{i}, \boldsymbol{y})$ be defined analogously following equation (9). Then, one can run the protocol in Figure 3 over $Y(\boldsymbol{r}_i, \boldsymbol{r}_y)$ where $\boldsymbol{r}_i \in \mathbb{F}^{\log N}$ and $\boldsymbol{r}_y \in \mathbb{F}^{\ell_y}$. For instance, the sumcheck on the convolution VE (equation (7)) can be written as

$$\tilde{Y}(\boldsymbol{i}, \boldsymbol{y}_1, \boldsymbol{y}_2) = \sum_{\substack{(\boldsymbol{x}_1, \boldsymbol{x}_2) \in \\ \{0,1\}^{2\lceil \log m \rceil + \lceil \log c \rceil}}} \tilde{X}(\boldsymbol{i}, \boldsymbol{y}_1, \boldsymbol{x}_1, \boldsymbol{x}_2) \cdot \tilde{W}(\boldsymbol{x}_1, \boldsymbol{y}_2, \boldsymbol{x}_2). \tag{10}$$

```
$\underline{\mathsf{VE}_{\mathsf{NN}}.\mathsf{P}(c_y, r_y, F, (X, W, P))}$

$c_X^{(L)} \leftarrow c_y, r_X^{(L)} \leftarrow r_y$
for $k = L-1, \ldots, 0$ :
    $\underline{\text{Run}}$ $(\boldsymbol{c}_X^{(k)}, \boldsymbol{c}_W^{(k)}, \boldsymbol{c}_P^{(k)}, \boldsymbol{r}_X^{(k)}, \boldsymbol{r}_W^{(k)}, \boldsymbol{r}_P^{(k)}) \leftarrow$
                    $\mathsf{VE}_{\mathsf{k}}.\mathsf{P}\left(\boldsymbol{c}_X^{(k+1)}, \boldsymbol{r}_X^{(k+1)}, F^{(k)}, (X^{(k)}, W^{(k)}, P^{(k)})\right)$
for $T \in \{W, P\}$ :
    $\underline{\text{Run}}$ $(c_T, r_T) \leftarrow \mathsf{VE}_{\mathsf{m\text{-}1}}.\mathsf{P}\left(\boldsymbol{c}_T^{(0)}, \ldots, \boldsymbol{c}_T^{(L-1)}, \boldsymbol{r}_T^{(0)}, \ldots, \boldsymbol{r}_T^{(L-1)}, T\right)$
$\underline{\text{Run}}$ $(c_X, r_X) \leftarrow \mathsf{VE}_{\mathsf{m\text{-}1}}.\mathsf{P}\left(\boldsymbol{c}_X^{(0)}, \boldsymbol{r}_X^{(0)}, X^{(0)}\right)$
return $(c_X, c_W, c_P, r_X, r_W, r_P)$

$\underline{\Pi_{\mathsf{NN}}.\mathsf{P}((\mathsf{crs}, \mathsf{crs}'), (\mathsf{com}_X, \mathsf{com}_W, \mathsf{com}_P, \boldsymbol{y}; X, W, P, o_X, o_W, o_P)):}$

for $T \in \{W, P\} : \pi_{1,T} \leftarrow \mathsf{AoK}_{\mathsf{Com}}.\mathsf{Prove}(\mathsf{crs}', \mathsf{com}_T, (T, o_T))$
$\underline{\text{Send}}$ $\pi_1 \leftarrow (\pi_{1,X}, \pi_{1,W}, \pi_{1,P})$
$\underline{\text{Get}}$ $r_y \leftarrow\!\!\$\ \mathcal{D}_y$ from V
$c_y \leftarrow \mathsf{H}(\boldsymbol{y}, r_y)$
$\underline{\text{Run}}$ $(c_X, c_W, c_P, r_X, r_W, r_P) \leftarrow \mathsf{VE}_{\mathsf{NN}}.\mathsf{P}(c_y, r_y, F, (X, W, P))$.
for $T \in \{W, P\} : \pi_T \leftarrow \mathsf{AoK}_{\mathsf{H}}.\mathsf{Prove}(\mathsf{crs}, (c_T, \mathsf{com}_T), (T, o_T))$
$\underline{\text{Send}}$ $(\pi_X, \pi_W, \pi_P)$ to V

$\underline{\Pi_{\mathsf{NN}}.\mathsf{V}(\mathsf{ck}, (\mathsf{com}_X, \mathsf{com}_W, \mathsf{com}_P, \boldsymbol{y})):}$

$\underline{\text{Get}}$ $(\pi_{1,X}, \pi_{1,W}, \pi_{1,P})$
$\underline{\text{Send}}$ $r_y \leftarrow\!\!\$\ \mathcal{D}_y$ and compute $c_y \leftarrow \mathsf{H}(\boldsymbol{y}, r_y)$
$r_T \leftarrow\!\!\$\ \mathcal{D}_{\mathcal{T}}$ for $T \in \{X, W, P\}$
$\underline{\text{Run}}$ $(c_X, c_W, c_P, b_0) \leftarrow \mathsf{VE}_{\mathsf{NN}}.\mathsf{V}(c_y, r_y, F, r_X, r_W, r_P)$
$\underline{\text{Get}}$ $(\pi_X, \pi_W, \pi_P)$
for $T \in \{X, W, P\}$ $b_T \leftarrow \mathsf{AoK}_{\mathsf{Com}}.\mathsf{Vf}(\mathsf{crs}', \mathsf{com}_T, \pi_{1,T})$
                    $\wedge\ \mathsf{AoK}_{\mathsf{H}}.\mathsf{Vf}(\mathsf{crs}, (\mathsf{com}_T, c_T, r_T), \pi_T)$.
return $b_0 \wedge b_X \wedge b_W \wedge b_T$
```

Figure 4: Modular construction of $\mathsf{VE}_{\mathsf{NN}}$ and compilation to an argument of knowledge Π_{NN}. The verifier $\mathsf{VE}_{\mathsf{NN}}.\mathsf{V}$ is omitted as it simply runs $\mathsf{VE}_{\mathsf{k}}.\mathsf{V}$ sequentially.

The resulting VE increases the prover time by a factor of $\lceil \log N \rceil$ and maintains the same soundness, communication complexity and verifier time as their single-input counterpart.

5.6 Verifiable Recurrent Neural Networks

As an additional application of our modular framework, we show how to construct a protocol for the verification of recurrent neural network (RNN) predictions, a problem that has not been addressed efficiently in the literature. RNNs are a type of neural network designed to process sequential data such as time series or natural language text. Unlike feedforward neural networks, which process input data in a single pass and do not maintain memory, RNNs have a loop that allows information to be passed from one time step to the next, following a cyclic computation graph.

Let T be the length of the longest cycle in the graph described by a RNN of L layers. For example, $T = 1$ in the RNN in Figure 5, as the only cycle is a self-loop. We construct a VE that verifies the computation of S predictions $(Y^{(1)}, \ldots, Y^{(S)})$ from (streaming) inputs $(X_0^{(1)}, \ldots, X_0^{(S)})$ as follows.

- The prover computes the predictions and stores all intermediate values $X_k^{(i)}$ for $i = 0, \ldots, S$. Then, it "unrolls" the intermediate computations of the RNN as in Figure 6. The resulting computation trace is a circuit of depth $D = L + S \cdot T$ with an evident layer structure.
- The prover embeds each layer of the computation trace in a multilinear polynomial $Z_k(\boldsymbol{j}, \boldsymbol{x}) \coloneqq Z_k^{(j)}(\boldsymbol{x})$ as in equation (9), and defines W_k, P_k accordingly. In total, one obtains D multilinear polynomials, structured as the layers in Figure 6.
- The VE proceeds similarly to the $\mathsf{VE}_{\mathsf{NN}}$ of Figure 4. Instead of obtaining fingerprints for each $X_k^{(i)}$ via separate VEs, one can work directly with the (batched) Z_k as follows. Let $g_{k,k+1}$ be the product of multilinear polynomials that relates $Z_{k+1}(\boldsymbol{i}, \boldsymbol{y})$ and $Z_k(\boldsymbol{j}, \boldsymbol{x})$. $g_{k,k+1}$ contains factors of Z_k, W_k, P_k, subsequently defined over variables $(\boldsymbol{j}, \boldsymbol{x})$. Then, we have

$$Z_{k+1}(\boldsymbol{i}, \boldsymbol{y}) = \sum_{\boldsymbol{x}, \boldsymbol{j}} g_{k,k+1}(Z_k, W_k, P_k)(\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{x}, \boldsymbol{y}).$$

  Finally, by summing over all layers that are input to Z_{k+1} and polynomials $g_{k',k+1}$ for $k' \leq k$, we can verify a fingerprint of Z_{k+1} in a single sumcheck. The evaluation of $g_{k,k+1}$ yields fingerprints of Z_k, W_k, P_k that can be handled as in $\mathsf{VE}_{\mathsf{NN}}$.

The resulting VE scheme has communication complexity and verifier time $\mathsf{t}_{\mathsf{V}} = |\pi| = O(D \cdot \log S \cdot \ell_{\max})$, where $\ell_{\max} = \max_{k=0}^{L-1} \lceil \log n_k \rceil$ and n_k is the size of $\tilde{X}_k^{(\cdot)}$. Even if the proof size scales linearly in the length of the stream S, we believe that our approach may present good concrete performance for small streams, in particular due to the batching technique.

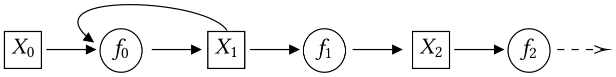

Figure 5: Illustration of a RNN with a loop at layer X_1.

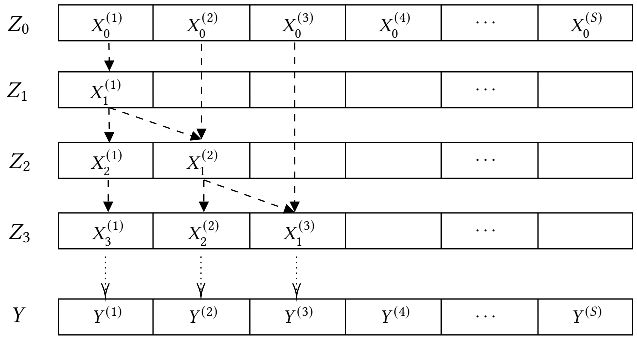

Figure 6: Computation trace of a sequence of inputs $X_0^{(1)}, \ldots, X_0^{(S)}$ in the RNN in Figure 5.

5.7 Image Processing

The techniques developed in these sections find a direct application in the verification of image processing operations. For instance, convolution is used in applications such as edge detection (such as using Sobel or Canny kernels), image blurring (Gaussian blur), and feature extraction. Below we provide a brief description how to construct a VE for some common applications.

- Operations that require geometric modifications or rearrangements of the original picture, such as cropping, rotation, mirroring, padding, or partial censoring (i.e. removal or replacement of sectors of an image) can be verified following Section 5.3.1.
- For convolution-related operations, one can directly apply our $\mathsf{VE}_{\mathsf{CNN}}$ with the desired parameters.
- Multiple transformations can be merged in a single sumcheck by merging wiring predicates. For instance, rotation + cropping + input reshaping (1) and a posterior convolutional filtering (2) can be verified with only two sumchecks.

For images encoded in RGB or other multi-channel format, we can apply batching techniques for the channels as shown in equation (10). If negative values appear in convolution kernels, linear shifts need to be applied to avoid wrapping of field elements. We compare the performance of our approach to ZK-IMG [24] and PhotoProof [30] in Section 6.3.

6 EVALUATION

In this section we discuss the performance of our solution and compare it to previous work. We focus the evaluation on our $\mathsf{VE}_{\mathsf{conv}}$ for convolution operations introduced in Section 5.2, as this is the most novel proof gadget compared to previous work.

6.1 Theoretical comparison

Recalling previous notation, let $n \times n$ be the input size, $m \times m$ the kernel size, and c, d the number of input and output channels, respectively. Our $\mathsf{VE}_{\mathsf{conv}}$ achieves short $|\pi| = (3 \cdot (2\lceil \log m \rceil + \lceil \log c \rceil) + 2) \cdot \lambda$, prover time $\mathsf{t_P} = O\Big(c(n_y^2 m^2 + m^2 d)\Big)$, and verifier time $\mathsf{t_V} = O\big(\log(cm^2)\big)$. In zkCNN [27], the proving time for a convolutional layer using the FFT-based approach involves a prover time $\mathsf{t_P} = O\big(n^2 cd\big)$ and verification $\mathsf{t_V} = O\Big(\log^2(n^2 cd)\Big)$, where $n = \max\{n_x, n_y\}$. Hence, our approach is always more efficient in communication complexity and verification time, while our prover is more efficient asymptotically when $m^2 \leq d$, which is often the case in practice (e.g., VGG16 presents $m = 3$ and d grows up to 512), and its running time is independent of d when the term $n_y^2 m^2$ dominates in the sum. Additionally, in zkCNN they need to either compute the FFT matrix or outsource this to the prover, thereby increasing proof size. We avoid all the complications of the multiple sumchecks in our direct approach. We also note that their FFT-sumcheck-based protocol can be easily expressed as a VE.

We note that, in many typical ML models, $n \gg m, c$ in early layers, and $c \gg n \approx m$ in 'deep' intermediate layers. Hence, even if our approach does not outperform the prover time of the FFT-based polynomial multiplication approach in all parameter regimes, it will improve it for many parameter sets in intermediate layers. Based on the characteristics of the layer, one could select the most efficient VE for convolution.

6.2 Experimental comparison

We implemented $\mathsf{VE}_{\mathsf{conv}}$ in Rust.[5] We use the `arkworks` library [2] for implementing field arithmetic over the 256-bit prime field from the `bls12-381` curve, the same field used in [27]. We also utilize several components of the `arkworks` sumcheck library that implements the doubly efficient protocol in [41].

We carry out different benchmarks in a virtual machine running Debian GNU/Linux with 8 cores Xeon-Gold-6154 at 3GHz and with 98 GB of RAM. Our implementation can be run using the natively supported parallelisation in `arkworks`, but we run our experiments on a single thread to facilitate comparison to previous work. All timings correspond to the average over 10 executions.

Single-channel convolution. Our first set of benchmarks run a single convolution with different input and kernel sizes. For small kernels $m = 4$, our VE prover requires 1.3 ms for a $n = 32$ input, and 98 ms for $n = 256$. In this parameter regime, our prover time is 5× faster than the FFT prover (and also the naive prover) in [27]. Our prover also outperforms [26] by two orders of magnitude. For large convolution kernels, the prover in zkCNN remains faster.

Verification is very fast and scales logarithmically on the kernel size, as expected. Verifying a moderate-size convolution such as $n = 256$ (in fact, for any n) and $m = 8$ takes 0.157 ms, whereas large kernels $m = 128$ require 0.362 ms.

Multiple channels. Our approach is optimized for multiple convolution channels, as we show in Figure 7. We display our results for a small fixed kernel $m = 4$ and input $n = 64$, for c up to 64 and $d = 1, 32, 128$. As seen in the chart, the prover time is essentially constant in d since $m^2 \cdot n^2$ dominates the sum. The verifier time is also very small, ranging from 0.07 ms for $c = 1$ to 0.210 ms for $c = 64$, and also constant in d.

We do not have concrete running times for multiple channels in zkCNN, but we expect their prover time to increase linearly on $c \cdot t$.

Communication complexity. We also provide concrete figures of the communication complexity (equivalently, the proof size of the non-interactive protocol), which is deterministic for $\mathsf{VE}_{\mathsf{conv}}$ (Proposition 5.1). For the single-channel experiments, the proof size amounts to 0.64 KB for $m = 8$, and 1.4 KB for $m = 128$, for any input size. This is a 8× improvement over zkCNN, which ranges from 5.6 KB to 8.4 KB for the same experiments. For the multi-channel setting in Figure 7, the instance $n = 64$, $m = 8$ and $c = 32$ yields a proof size of 1.12 KB for any d.

Image Processing. Finally, we benchmark a convolution proof of a 8×8 kernel (such as blurring) with a RGB image (720×480) with the goal of comparing to ZK-IMG [24], which already outperforms [30] by several orders of magnitude. The comparison is only approximate as their benchmarks are run on more powerful hardware than ours, and image sizes are not identical.

In this regime, $\mathsf{VE}_{\mathsf{conv}}$ takes 3.3 s of proving time, 0.12 ms of verification time and yields a proof size of 0.64 KB. In ZK-IMG, a

[5] Our code is available at https://github.com/imdea-software/MSCProof

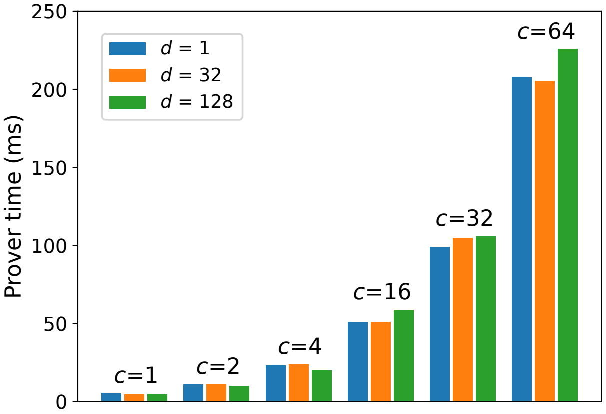

Figure 7: Prover time for varying number of channels c, d and fixed $n = 64$ and $m = 4$.

3× larger 1280 × 720 convolution input involves 78 s of proving (ignoring key generation), 8.12 ms of verification, and 11 KB proof size (a 20× increase).

For a 128 × 128 input, they report 2.7 s of proving time and 5.3 ms of verification on standard hardware. For the same size and a 8 × 8 kernel, our prover takes 110 ms (25× faster) and our verifier 0.117 ms.

Nevertheless, ZK-IMG implements a complete proof system, while our approach requires an additional polynomial commitment. We expect other simple transformations (cropping, padding, partial censoring...) to present similar running times.

Pre-Processing in $\mathsf{VE}_{\mathsf{conv}}$. As discussed in Sections 5.2 and 5.3.1, a pre-processing reshaping step, which can often be embedded into other steps such as activation layers, is required if $\mathsf{VE}_{\mathsf{conv}}$ is used to prove a standalone convolution. In that case, the sumcheck in Section 5.3.1 needs to be executed after $\mathsf{VE}_{\mathsf{conv}}$. We do not include this step in our benchmarks, but note that it induces a minimal overhead as (1) the sumcheck involves strictly less variables and rounds than $\mathsf{VE}_{\mathsf{conv}}$, and (2) the prover already has the fingerprints to the reshaped input.

Polynomial Commitment Overhead. A polynomial commitment is used in the AoK described in Proposition 5.2 but not at the VE level. The overhead induced by the PC depends on the chosen scheme and affects the efficiency of our solution and prior work's [27] in the same way. In the case of zkCNN, sumchecks take roughly 2/3 of the total prover time, whereas PCs take the remaining 1/3 (see [27], Table 1). Our improvements in the information-theoretic protocol significantly reduce the fraction taken by the sumchecks.

For completeness, we benchmark the multilinear KZG from HyperPlonk [4] together with our $\mathsf{VE}_{\mathsf{conv}}$. For a single-channel convolution of $n = 256$, $m = 4$, a PC opening takes 400 ms, whereas the VE sumcheck prover takes 98 ms. The commit operation takes 191 ms. We remark that the PC opening cost gets further amortized when more VEs are composed sequentially. In general, the *deeper* the model is, the more significant the sumcheck overhead becomes.

6.3 Discussion

Our protocols achieve, overall, faster prover times, reduced communication and faster verification times than existing solutions. As in other works [24, 26, 27], we found memory usage to be the main bottleneck, the reason being the dynamic programming technique used by the prover to compute the multilinear extensions. Yet, our approach allows for clearing the memory after every sequential step, as opposed to solutions such as [26] or [24] (built upon general-purpose proof systems). A solution towards improving memory bottlenecks is to trade memory usage for proving time by applying streaming algorithms for multilinear extensions [12], which is an interesting direction for future work.

ACKNOWLEDGMENTS

This work is supported by the PICOCRYPT project that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant agreement No. 101001283), partially supported by projects PRODIGY (TED2021-132464B-I00) and ESPADA (PID2022-142290OB-I00) funded by MCIN/AEI/10.13039/501100011033/ and the European Union NextGenerationEU / PRTR, partially funded by the European Commission through the HORIZON-JU-SNS-2022 ACROSS project with Grant Agreement number 101097122, partially funded by Ministerio de Universidades (FPU21/00600) and MINECO Project CREEME (PID2019-109379RB-I00).

REFERENCES

[1] Ramy E. Ali, Jinhyun So, and Amir Salman Avestimehr. 2020. On Polynomial Approximations for Privacy-Preserving and Verifiable ReLU Networks. *CoRR* abs/2011.05530 (2020). arXiv:2011.05530 https://arxiv.org/abs/2011.05530

[2] arkworks contributors. 2022. arkworks *zkSNARK ecosystem*. https://arkworks.rs

[3] Matteo Campanelli, Dario Fiore, and Anaïs Querol. 2019. LegoSNARK: Modular Design and Composition of Succinct Zero-Knowledge Proofs. In *ACM CCS 2019*, Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz (Eds.). ACM Press, London, UK, 2075–2092. https://doi.org/10.1145/3319535.3339820

[4] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. 2023. HyperPlonk: Plonk with Linear-Time Prover and High-Degree Custom Gates. In *Advances in Cryptology – EUROCRYPT 2023*, Carmit Hazay and Martijn Stam (Eds.). Springer Nature Switzerland, Cham, 499–530.

[5] Haixia Chen, Xinyi Huang, Jianting Ning, Futai Zhang, and Chao Lin. 2022. VILS: A Verifiable Image Licensing System. *IEEE Transactions on Information Forensics and Security* 17 (2022), 1420–1434. https://doi.org/10.1109/TIFS.2022.3162105

[6] Shuo Chen, Jung Hee Cheon, Dongwoo Kim, and Daejun Park. 2019. Verifiable Computing for Approximate Computation. Cryptology ePrint Archive, Report 2019/762. https://eprint.iacr.org/2019/762.

[7] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. 2014. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759* (2014).

[8] Alessandro Chiesa, Michael A. Forbes, and Nicholas Spooner. 2017. A Zero Knowledge Sumcheck and its Applications. Cryptology ePrint Archive, Report 2017/305. https://eprint.iacr.org/2017/305.

[9] Alessandro Chiesa and Eran Tromer. 2010. Proof-Carrying Data and Hearsay Arguments from Signature Cards. In *Innovations in Computer Science - ICS 2010, Tsinghua University, Beijing, China, January 5-7, 2010. Proceedings.* 310–331.

[10] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. 2012. Practical verified computation with streaming interactive proofs. In *ITCS 2012*, Shafi Goldwasser (Ed.). ACM, Cambridge, MA, USA, 90–112. https://doi.org/10.1145/2090236.2090245

[11] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. 2012. Practical Verified Computation with Streaming Interactive Proofs. In *Proceedings of*

the 3rd Innovations in Theoretical Computer Science Conference (Cambridge, Massachusetts) *(ITCS '12)*. Association for Computing Machinery, New York, NY, USA, 90–112. https://doi.org/10.1145/2090236.2090245

[12] Graham Cormode, Justin Thaler, and Ke Yi. 2011. Verifying Computations with Streaming Interactive Proofs. *Proc. VLDB Endow.* 5, 1 (sep 2011), 25–36. https://doi.org/10.14778/2047485.2047488

[13] Council of European Union. 2000. Council Directive 2000/43/EC of 29 June 2000 implementing the principle of equal treatment between persons irrespective of racial or ethnic origin.
https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32000L0043.

[14] Council of European Union. 2004. Council Directive 2004/113/EC of 13 December 2004 implementing the principle of equal treatment between men and women in the access to and supply of goods and services.
https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32004L0113.

[15] Vincent Dumoulin and Francesco Visin. 2016. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285* (2016).

[16] Liam Eagen, Dario Fiore, and Ariel Gabizon. 2022. cq: Cached quotients for fast lookups. Cryptology ePrint Archive, Report 2022/1763. https://eprint.iacr.org/2022/1763.

[17] Federal Office for Information Security Germany (BSI). 2022. Auditing machine learning algorithms - A white paper for public auditors. https://www.hhi.fraunhofer.de/fileadmin/Departments/AI/TechnologiesAndSolutions/2022-05-23-whitepaper-tuev-bsi-hhi-towards-auditable-ai-systems.pdf.

[18] Boyuan Feng, Lianke Qin, Zhenfei Zhang, Yufei Ding, and Shumo Chu. 2021. ZEN: An Optimizing Compiler for Verifiable, Zero-Knowledge Neural Network Inferences. Cryptology ePrint Archive, Report 2021/087. https://ia.cr/2021/087.

[19] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. 2008. Delegating computation: interactive proofs for muggles. In *40th ACM STOC*, Richard E. Ladner and Cynthia Dwork (Eds.). ACM Press, Victoria, BC, Canada, 113–122. https://doi.org/10.1145/1374376.1374396

[20] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. 1985. The Knowledge Complexity of Interactive Proof-Systems (Extended Abstract). In *17th ACM STOC*. ACM Press, Providence, RI, USA, 291–304. https://doi.org/10.1145/22145.22178

[21] Alexander Golovnev, Jonathan Lee, Srinath Setty, Justin Thaler, and Riad S. Wahby. 2021. Brakedown: Linear-time and post-quantum SNARKs for R1CS. Cryptology ePrint Archive, Report 2021/1043. https://ia.cr/2021/1043.

[22] Justin Holmgren and Ron Rothblum. 2018. Delegating Computations with (Almost) Minimal Time and Space Overhead. In *59th FOCS*, Mikkel Thorup (Ed.). IEEE Computer Society Press, Paris, France, 124–135. https://doi.org/10.1109/FOCS.2018.00021

[23] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2704–2713.

[24] Daniel Kang, Tatsunori Hashimoto, Ion Stoica, and Yi Sun. 2022. ZK-IMG: Attested Images via Zero-Knowledge Proofs to Fight Disinformation. arXiv:2211.04775 [cs.CR]

[25] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. 2010. Constant-Size Commitments to Polynomials and Their Applications. In *ASIACRYPT 2010 (LNCS, Vol. 6477)*, Masayuki Abe (Ed.). Springer, Heidelberg, Germany, Singapore, 177–194. https://doi.org/10.1007/978-3-642-17373-8_11

[26] Seunghwa Lee, Hankyung Ko, Jihye Kim, and Hyunok Oh. 2020. vCNN: Verifiable Convolutional Neural Network. Cryptology ePrint Archive, Report 2020/584. https://eprint.iacr.org/2020/584.

[27] Tianyi Liu, Xiang Xie, and Yupeng Zhang. 2021. zkCNN: Zero Knowledge Proofs for Convolutional Neural Network Predictions and Accuracy. In *ACM CCS 2021*, Giovanni Vigna and Elaine Shi (Eds.). ACM Press, Virtual Event, Republic of Korea, 2968–2985. https://doi.org/10.1145/3460120.3485379

[28] Tianyi Liu, Xiang Xie, and Yupeng Zhang. 2021. zkCNN: Zero knowledge proofs for convolutional neural network predictions and accuracy. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 2968–2985.

[29] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. 1992. Algebraic Methods for Interactive Proof Systems. *J. ACM* 39, 4 (oct 1992), 859–868. https://doi.org/10.1145/146585.146605

[30] Assa Naveh and Eran Tromer. 2016. PhotoProof: Cryptographic Image Authentication for Any Set of Permissible Transformations. In *2016 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, San Jose, CA, USA, 255–271. https://doi.org/10.1109/SP.2016.23

[31] Jim Posen and Assimakis A. Kattis. 2022. Caulk+: Table-independent lookup arguments. Cryptology ePrint Archive, Report 2022/957. https://eprint.iacr.org/2022/957.

[32] scipr-lab. 2017. libsnark: a C++ library for zkSNARK proofs. https://github.com/scipr-lab/libsnark.

[33] Srinath Setty. 2020. Spartan: Efficient and General-Purpose zkSNARKs Without Trusted Setup. In *CRYPTO 2020, Part III (LNCS, Vol. 12172)*, Daniele Micciancio and Thomas Ristenpart (Eds.). Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 704–737. https://doi.org/10.1007/978-3-030-56877-1_25

[34] Eduardo Soria-Vazquez. 2022. Doubly Efficient Interactive Proofs over Infinite and Non-commutative Rings. In *TCC 2022, Part I (LNCS)*. Springer, Heidelberg, Germany, 497–525. https://doi.org/10.1007/978-3-031-22318-1_18

[35] Justin Thaler. 2013. Time-Optimal Interactive Proofs for Circuit Evaluation. In *CRYPTO 2013, Part II (LNCS, Vol. 8043)*, Ran Canetti and Juan A. Garay (Eds.). Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 71–89. https://doi.org/10.1007/978-3-642-40084-1_5

[36] the Supreme Audit Institutions of Finland, Germany, the Netherlands, Norway and the UK. 2020. Towards Auditable AI Systems - From Principles to Practice. https://www.auditingalgorithms.net/.

[37] Stanford University. [n. d.]. CS231: Convolutional Neural Networks for Pattern Recognition. https://cs231n.github.io/convolutional-networks/#convert.

[38] Victor Vu, Srinath T. V. Setty, Andrew J. Blumberg, and Michael Walfish. 2013. A Hybrid Architecture for Interactive Verifiable Computation. In *2013 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, Berkeley, CA, USA, 223–237. https://doi.org/10.1109/SP.2013.48

[39] Riad S. Wahby, Ye Ji, Andrew J. Blumberg, abhi shelat, Justin Thaler, Michael Walfish, and Thomas Wies. 2017. Full Accounting for Verifiable Outsourcing. In *ACM CCS 2017*, Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM Press, Dallas, TX, USA, 2071–2086. https://doi.org/10.1145/3133956.3133984

[40] Riad S. Wahby, Ioanna Tzialla, abhi shelat, Justin Thaler, and Michael Walfish. 2018. Doubly-Efficient zkSNARKs Without Trusted Setup. In *2018 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, San Francisco, CA, USA, 926–943. https://doi.org/10.1109/SP.2018.00060

[41] Tiancheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. 2019. Libra: Succinct Zero-Knowledge Proofs with Optimal Prover Computation. In *CRYPTO 2019, Part III (LNCS, Vol. 11694)*, Alexandra Boldyreva and Daniele Micciancio (Eds.). Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 733–764. https://doi.org/10.1007/978-3-030-26954-8_24

[42] Tiancheng Xie, Yupeng Zhang, and Dawn Song. 2022. Orion: Zero Knowledge Proof with Linear Prover Time. In *CRYPTO 2022, Part IV (LNCS, Vol. 13510)*, Yevgeniy Dodis and Thomas Shrimpton (Eds.). Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 299–328. https://doi.org/10.1007/978-3-031-15985-5_11

[43] Arantxa Zapico, Vitalik Buterin, Dmitry Khovratovich, Mary Maller, Anca Nitulescu, and Mark Simkin. 2022. Caulk: Lookup Arguments in Sublinear Time. In *ACM CCS 2022*, Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi (Eds.). ACM Press, Los Angeles, CA, USA, 3121–3134. https://doi.org/10.1145/3548606.3560646

[44] Arantxa Zapico, Ariel Gabizon, Dmitry Khovratovich, Mary Maller, and Carla Ràfols. 2022. Baloo: Nearly Optimal Lookup Arguments. Cryptology ePrint Archive, Report 2022/1565. https://eprint.iacr.org/2022/1565.

[45] zcash. 2022. halo2. https://zcash.github.io/halo2/.

[46] Jiaheng Zhang, Zhiyong Fang, Yupeng Zhang, and Dawn Song. 2020. Zero Knowledge Proofs for Decision Tree Predictions and Accuracy. In *ACM CCS 2020*, Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna (Eds.). ACM Press, Virtual Event, USA, 2039–2053. https://doi.org/10.1145/3372297.3417278

[47] Jiaheng Zhang, Tianyi Liu, Weijie Wang, Yinuo Zhang, Dawn Song, Xiang Xie, and Yupeng Zhang. 2021. Doubly Efficient Interactive Proofs for General Arithmetic Circuits with Linear Prover Time. In *ACM CCS 2021*, Giovanni Vigna and Elaine Shi (Eds.). ACM Press, Virtual Event, Republic of Korea, 159–177. https://doi.org/10.1145/3460120.3484767

[48] Jiaheng Zhang, Tiancheng Xie, Yupeng Zhang, and Dawn Song. 2020. Transparent Polynomial Delegation and Its Applications to Zero Knowledge Proof. In *2020 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, San Francisco, CA, USA, 859–876. https://doi.org/10.1109/SP40000.2020.00052

[49] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. 2017. vSQL: Verifying Arbitrary SQL Queries over Dynamic Outsourced Databases. In *2017 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, San Jose, CA, USA, 863–880. https://doi.org/10.1109/SP.2017.43

[50] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. 2018. vRAM: Faster Verifiable RAM with Program-Independent Preprocessing. In *2018 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, San Francisco, CA, USA, 908–925. https://doi.org/10.1109/SP.2018.00013

A CRYPTOGRAPHIC PRIMITIVES

Definition A.1 (Commitments). A commitment scheme Com is a tuple of algorithms $(\mathsf{Setup}, \mathsf{Com}, \mathsf{Vf})$ such that

$\mathsf{Setup}(1^\lambda) \to \mathsf{ck}$ takes the security parameter and outputs the commitment key ck.

$\mathsf{Com}(\mathsf{ck}, x) \rightarrow (\mathsf{com}, o)$ on input the commitment key ck and a value x, outputs a commitment com and an opening o.

$\mathsf{Vf}(\mathsf{ck}, \mathsf{com}, x, o) \rightarrow b$ on input a commitment com, a value x and an opening o, it outputs 1 (accept) or 0 (reject).

Correctness. $\forall \lambda \in \mathbb{N}$ and any honestly generated commitment key $\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda)$ and any input x, if $(\mathsf{com}, o) \leftarrow \mathsf{Com}(\mathsf{ck}, x)$, then $\mathsf{Vf}(\mathsf{ck}, \mathsf{com}, x, o) = 1$.

Computational Binding. For every PPT adversary $\mathcal{A}$, the following probability is negligible

$$\Pr\left[\begin{array}{c} x \neq x \\ \wedge \mathsf{Vf}(\mathsf{ck}, \mathsf{com}, x, o) = 1 \\ \wedge \mathsf{Vf}(\mathsf{ck}, \mathsf{com}, x', o') = 1 \end{array} : \begin{array}{c} \mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{com}, x, o, x', o') \leftarrow \mathcal{A}(\mathsf{ck}) \end{array}\right]$$

Statistical Hiding. For $\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda)$ and every pair of inputs x, x', the following distributions are statistically close:

$$\{\mathsf{com} : (\mathsf{com}, o) \leftarrow \mathsf{Com}(\mathsf{ck}, x)\} \approx$$
$$\{\mathsf{com}' : (\mathsf{com}', o') \leftarrow \mathsf{Com}(\mathsf{ck}, x')\}.$$

The scheme is *perfectly* hiding if both distributions are identical.

Definition A.2 (Arguments of Knowledge). An argument of knowledge AoK for an NP relation $\mathcal{R}$ is a tuple of algorithms (Setup, Prove, Vf) such that:

$\mathsf{Setup}(1^\lambda, \mathcal{R}) \rightarrow \mathsf{crs}$ outputs a common reference string crs.

$\mathsf{Prove}(\mathsf{crs}, x, w) \rightarrow \pi$ on input crs, a statement x and a witness w such that $(x, w) \in \mathcal{R}$, it returns a proof π.

$\mathsf{Vf}(\mathsf{crs}, x, \pi) \rightarrow b$ given crs, a statement x and a proof π, it outputs 1 (accept) or 0 (reject).

Completeness. AoK is complete if for any $\lambda \in \mathbb{N}$ and $(x, w) \in \mathcal{R}$ it holds $\Pr[\mathsf{Vf}(\mathsf{crs}, x, \mathsf{Prove}(\mathsf{crs}, x, w)) = 1] = 1$ where $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{R})$.

Knowledge-soundness. For any PT adversary $\mathcal{A}$ there exists an extractor Ext (taking the same input of $\mathcal{A}$ including the random tape ρ) such that

$$\Pr\left[\begin{array}{c} \mathsf{Vf}(\mathsf{crs}, x, \pi) = 1 \\ \wedge \\ (x, w) \notin \mathcal{R} \end{array} : \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{R}) \\ (x, \pi) \leftarrow \mathcal{A}(\mathsf{crs}; \rho) \\ w \leftarrow \mathsf{Ext}(\mathsf{crs}; \rho) \end{array}\right] = \mathsf{negl}(\lambda)$$

Zero-knowledge. AoK is computationally (resp. statistical, perfect) zero-knowledge if there exists a simulator $\mathsf{Sim} = (\mathsf{Sim}_0, \mathsf{Sim}_1)$ such that: (i) $\mathsf{Sim}_0(1^\lambda, \mathcal{R}) \rightarrow (\mathsf{crs}, \mathsf{td})$ generates a crs that is computationally (resp. statistically, perfectly) indistinguishable from that generated by Setup; (ii) for any $(x, w) \in \mathcal{R}$, and $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Sim}_0(1^\lambda, \mathcal{R})$, $\mathsf{Sim}_1(\mathsf{td}, x)$ generates proofs that are computationally (resp. statistically, perfectly) indistinguishable from those generated by $\mathsf{Prove}(\mathsf{crs}, x, w)$.

Commit-and-Prove AoK**.** A commit-and-prove argument of knowledge for a relation $\mathcal{R}$ and a commitment scheme Com is an argument of knowledge for the NP relation $\mathcal{R}_{\mathsf{Com}}$ such that $((x, \mathsf{com}); (u, o, w)) \in \mathcal{R}_{\mathsf{Com}}$ iff $(x, (u, w)) \in \mathcal{R}$ and $\mathsf{Com}.\mathsf{Vf}(\mathsf{ck}, \mathsf{com}, u, o) = 1$.

B FROM VE TO ARGUMENTS OF KNOWLEDGE

VE schemes can be easily leveraged to construct cryptographic arguments (of knowledge) for verifiable computation by using cryptographic primitives. The building blocks that we require for this construction are

- a commitment scheme $\mathsf{Com} := (\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Vf})$,
- an argument of knowledge $\mathsf{AoK}_\mathsf{H} := (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Vf})$ for the relation $\mathcal{R}_\mathsf{H} = \{\mathsf{c}_x, \mathsf{com}_x, r_x; \boldsymbol{x}, o_x) : \mathsf{Com}.\mathsf{Vf}(\mathsf{ck}, \mathsf{com}_x, \boldsymbol{x}, o_x) = 1 \wedge \mathsf{c}_x = \mathsf{H}(\boldsymbol{x}, r_x)\}$, where $\mathsf{ck} \leftarrow \mathsf{Com}.\mathsf{Setup}(1^\lambda)$,
- an $\mathsf{AoK}_{\mathsf{Com}}$ for the "proof of knowledge" relation "I know the $\boldsymbol{x}$ committed in com_x" given by $\mathcal{R}_{\mathsf{PoK}} = \{(\mathsf{com}_x; \boldsymbol{x}, o_x) : \mathsf{Com}.\mathsf{Vf}(\mathsf{ck}, \mathsf{com}_x, \boldsymbol{x}, o_x)\}$.
- and a VE scheme for a family of functions $\mathcal{F}$.

In this section, we show how to use a VE to build an interactive argument of knowledge $\Pi := (\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ for the relation

$$\mathcal{R}_\Pi = \{(f, \mathsf{com}_x, \boldsymbol{y}; \boldsymbol{x}, o_x) : f \in \mathcal{F} \wedge f(\boldsymbol{x}) = \boldsymbol{y}$$
$$\wedge\ \mathsf{Com}.\mathsf{Vf}(\mathsf{ck}, \mathsf{com}_x, \boldsymbol{x}, o_x)\}$$

where o_x is the opening for the committed $\boldsymbol{x}$.

We describe our construction, which is a generalization of the construction in [49], in Figure 8. We refer to the full version for the security proof.

```
$\Pi.\mathsf{Setup}(1^\lambda, \mathsf{ck})$:
  $\mathsf{crs} \leftarrow \mathsf{AoK}_\mathsf{H}.\mathsf{Setup}(1^\lambda, (\mathsf{ck}, \mathcal{R}_\mathsf{H}))$
  $\mathsf{crs}' \leftarrow \mathsf{AoK}_{\mathsf{Com}}.\mathsf{Setup}(1^\lambda, (\mathsf{ck}, \mathcal{R}_{\mathsf{Com}}))$
  return $(\mathsf{crs}, \mathsf{crs}')$.
$\Pi.\mathsf{P}((\mathsf{crs}, \mathsf{crs}'), (f, \mathsf{com}_x, \boldsymbol{y}; \boldsymbol{x}, o_x))$:
  $\pi_1 \leftarrow \mathsf{AoK}_{\mathsf{Com}}.\mathsf{Prove}(\mathsf{crs}', (\mathsf{com}_x; \boldsymbol{x}, o_x))$
  Send $\pi_1$ to V
  Get $r_y \leftarrow\$ \mathcal{D}_\mathcal{F}$ from V
  $c_{\boldsymbol{y}} \leftarrow \mathsf{H}(\boldsymbol{y}, r_y)$
  Run $(c_{\boldsymbol{x}}, r_x) \leftarrow \mathsf{VE}.\mathsf{P}(\boldsymbol{x}, f, c_{\boldsymbol{y}}, r_y)$ interactively with V.
  $\pi_2 \leftarrow \mathsf{AoK}_\mathsf{H}.\mathsf{Prove}(\mathsf{crs}, (c_{\boldsymbol{x}}, \mathsf{com}_x, r_x; \boldsymbol{x}, o_x))$
  Send $\pi_2$ to V
$\Pi.\mathsf{V}((\mathsf{crs}, \mathsf{crs}'), (f, \mathsf{com}_x, \boldsymbol{y}))$:
  Get $\pi_1$ from P
  Send $r_y \leftarrow\$ \mathcal{D}_\mathcal{F}$ to P and compute $c_{\boldsymbol{y}} \leftarrow \mathsf{H}(\boldsymbol{y}, r_y)$
  $r_x \leftarrow\$ \mathcal{D}_\mathcal{X}$
  Run $b_0 \leftarrow \mathsf{VE}.\mathsf{V}(r_x, f, c_{\boldsymbol{y}}, r_y)$ interactively with P.
  Get $\pi_2$ from P
  $b_1 \leftarrow \mathsf{AoK}_{\mathsf{Com}}.\mathsf{Vf}(\mathsf{crs}', \mathsf{com}_x, \pi_1)$
  $b_2 \leftarrow \mathsf{AoK}_\mathsf{H}.\mathsf{Vf}(\mathsf{crs}, (c_{\boldsymbol{x}}, \mathsf{com}_x, r_x), \pi_2)$
  return $b_0 \wedge b_1 \wedge b_2$
```

Figure 8: Construction of an interactive argument of knowledge Π for the relation $\mathcal{R}_\Pi$ from a commitment scheme Com **such that** $\mathsf{ck} \leftarrow \mathsf{Com}.\mathsf{Setup}(1^\lambda)$**, arguments of knowledge** $\mathsf{AoK}_{\mathsf{Com}}$ **for $\mathcal{R}_{\mathsf{PoK}}$ and** AoK_H **for $\mathcal{R}_\mathsf{H}$, and a** VE **scheme for $\mathcal{F}$.**