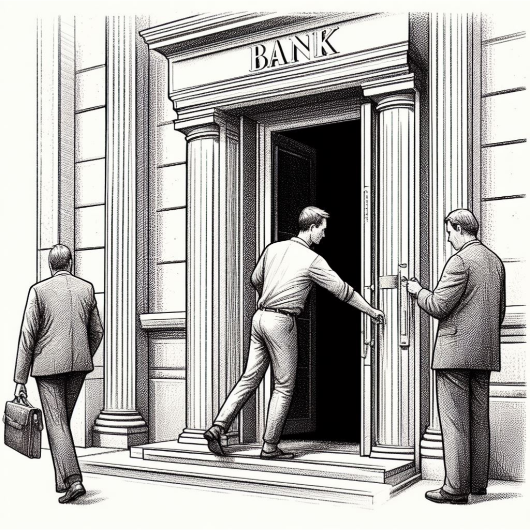
Modular Sumcheck Proofs with Applications to Machine Learning and Image Processing

D. Balbás[1,2], D. Fiore[1], M. González-Vasco[3], D. Robissout[1], C. Soriente[4]

28th November 2023

[1]IMDEA Software Institute, Madrid, Spain
[2]Universidad Politécnica de Madrid, Spain
[3]Universidad Carlos III de Madrid, Spain
[4]NEC Laboratories Europe, Madrid, Spain

ACM CCS 2023, Copenhagen, Denmark


BANK


BANK

Proof Systems in the Wild

Towards proving larger models, we require:

- **Efficient Verification**
- **Efficient Proof Generation:** $\tilde{O}(n)$ time, usually achieved by *sumcheck-based proofs*. Low memory usage.
- **Privacy** for model parameters.

Proof Systems in the Wild

Towards proving larger models, we require:

- **Efficient Verification**
- **Efficient Proof Generation:** $\tilde{O}(n)$ time, usually achieved by *sumcheck-based proofs*. Low memory usage.
- **Privacy** for model parameters.

General-purpose proof systems (e.g. SNARKs) not great for *data intensive computations*.

Proof Systems in the Wild

Towards proving larger models, we require:

- **Efficient Verification**
- **Efficient Proof Generation:** $\tilde{O}(n)$ time, usually achieved by *sumcheck-based proofs*. Low memory usage.
- **Privacy** for model parameters.

General-purpose proof systems (e.g. SNARKs) not great for *data intensive computations*.

Special-purpose proofs (e.g. vCNN, zkCNN, zkIMG) better but lack *composability/reusability*.

This Work

- **Framework** for composing sumcheck-based proofs at an information-theoretic level.

This Work

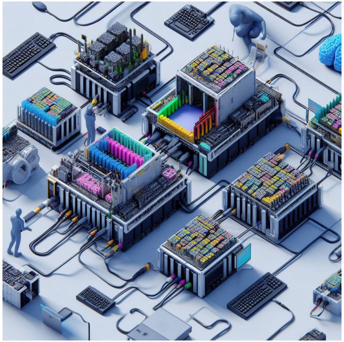
- **Framework** for composing sumcheck-based proofs at an information-theoretic level.
- Better efficiency for **combined special-purpose protocols**.

This Work

- **Framework** for composing sumcheck-based proofs at an information-theoretic level.
- Better efficiency for **combined special-purpose protocols**.
- Efficient proofs for multi-channel **convolution**.

This Work

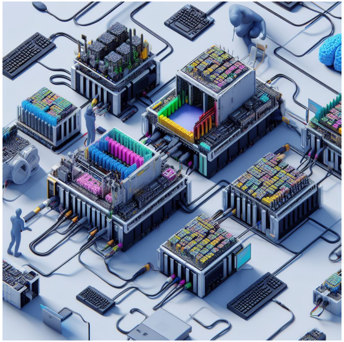
- **Framework** for composing sumcheck-based proofs at an information-theoretic level.
- Better efficiency for **combined special-purpose protocols**.
- Efficient proofs for multi-channel **convolution**.
- Modular, extendable Rust **implementation**, 5-10x faster & shorter than special-purpose proofs for ML and IP.



This work as seen by Dall·E 3

This Work

- **Framework** for *composing sumcheck-based proofs* at an information-theoretic level.
- Better efficiency for **combined special-purpose protocols**.
- Efficient *proofs for multi-channel* **convolution**.
- Modular, extendable Rust **implementation**, 5-10x *faster & shorter than special-purpose proofs* for ML and IP.



This work as seen by Dall·E 3

Our Framework

Fingerprints and Interactive Proofs

An **interactive proof** (IP) for the language $\mathcal{L}_{\mathcal{F}} = \{(f, x, y) : f(x) = y\}$ is a pair of algorithms $\langle \mathcal{P}, \mathcal{V} \rangle (f, x, y) \to b$ that are *complete* and *sound*.

Fingerprints and Interactive Proofs

An **interactive proof** (IP) for the language $\mathcal{L}_{\mathcal{F}} = \{(f, x, y) : f(x) = y\}$ is a pair of algorithms $\langle \mathcal{P}, \mathcal{V} \rangle (f, x, y) \to b$ that are *complete* and *sound*.

Fingerprint

Let $c_x \leftarrow \mathsf{H}(x, r)$ be the **fingerprint** of $\boldsymbol{x}$ on $\boldsymbol{r}$.

H is (statistically) sound if for *any* pair $x \neq x^*$,

$$\Pr_r[\mathsf{H}(x, r) = \mathsf{H}(x^*, r)] = \mathsf{negl}(\lambda).$$

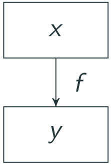
Example: for $\boldsymbol{x} \in \mathbb{F}^n$, the poly. evaluation $\mathsf{H}(\boldsymbol{x}, r) = x_0 + x_1 r + \cdots + x_{n-1} r^{n-1}$ over $\mathbb{F}$.

Structure of Common IPs

We can explain many efficient IPs for (f, x, y) with the following abstraction:

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

Prover	**Verifier**
$\mathsf{c}_y \leftarrow \mathsf{H}(y, r_y)$	$\mathsf{c}_y \leftarrow \mathsf{H}(y, r_y)$



Structure of Common IPs

We can explain many efficient IPs for (f, x, y) with the following abstraction:

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

Prover		**Verifier**
$\mathsf{c}_y \leftarrow \mathsf{H}(y, r_y)$		$\mathsf{c}_y \leftarrow \mathsf{H}(y, r_y)$
c_x, r_x	$\xleftrightarrow{\langle \mathcal{P}_{\mathsf{VE}}(x), \mathcal{V}_{\mathsf{VE}} \rangle (f, \mathsf{c}_y, r_y)}$	c_x, r_x, b

$$\begin{tikzcd} \boxed{x} \arrow[d, "f"] \\ \boxed{y} \end{tikzcd}$$

Structure of Common IPs

We can explain many efficient IPs for (f, x, y) with the following abstraction:

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

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

$$\begin{tikzcd} \boxed{x} \arrow[d, "f"] \\ \boxed{y} \end{tikzcd}$$

Structure of Common IPs

We can explain many efficient IPs for (f, x, y) with the following abstraction:

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

Prover		**Verifier**
$\mathsf{c}_y \leftarrow \mathsf{H}(y, r_y)$		$\mathsf{c}_y \leftarrow \mathsf{H}(y, r_y)$
c_x, r_x	$\xleftrightarrow{\langle \mathcal{P}_{\mathsf{VE}}(x), \mathcal{V}_{\mathsf{VE}} \rangle (f, \mathsf{c}_y, r_y)}$	c_x, r_x, b

$$\begin{tikzcd} \boxed{x} \arrow[d, "f"] \\ \boxed{y} \end{tikzcd}$$

Subroutines named **verifiable evaluation schemes** *(VE) on fingerprinted data*.

Structure of Common IPs

We can explain many efficient IPs for (f, x, y) with the following abstraction:

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

Prover		**Verifier**
$\mathsf{c}_y \leftarrow \mathrm{H}(y, r_y)$		$\mathsf{c}_y \leftarrow \mathrm{H}(y, r_y)$
c_2, r_2	$\underleftrightarrow{\langle \mathcal{P}_{\mathsf{VE}}(x_2), \mathcal{V}_{\mathsf{VE}} \rangle (f_3, \mathsf{c}_y, r_y)}$	c_2, r_2, b_2
c_1, r_1	$\underleftrightarrow{\langle \mathcal{P}_{\mathsf{VE}}(x_1), \mathcal{V}_{\mathsf{VE}} \rangle (f_2, \mathsf{c}_2, r_2)}$	c_1, r_1, b_1
c_x, r_x	$\underleftrightarrow{\langle \mathcal{P}_{\mathsf{VE}}(x), \mathcal{V}_{\mathsf{VE}} \rangle (f_1, \mathsf{c}_1, r_1)}$	c_x, r_x, b_0

$$\boxed{x} \xrightarrow{f_1} \boxed{x_1} \xrightarrow{f_2} \boxed{x_2} \xrightarrow{f_3} \boxed{y}$$

Subroutines named **verifiable evaluation schemes** *(VE) on fingerprinted data*.

Our Framework

We characterize VEs and provide a formalism. **VEs can be composed sequentially at the information-theoretic level!**

VEs can express many sumcheck-based proof systems:

Our Framework

We characterize VEs and provide a formalism. **VEs can be composed sequentially at the information-theoretic level!**

VEs can express many sumcheck-based proof systems:

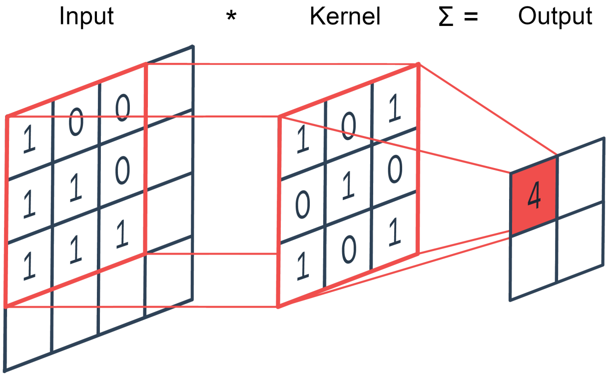
- Matrix multiplication [Thaler13]
- GKR [GKR08, CMT12]
- Libra [XZZPS19], Virgo [ZLW+20] (and follow-ups)
- FFT-based convolution [LXZ21]
- ...

We provide a compiler *from VEs to succinct cryptographic arguments*. Fingerprints can be evaluated efficiently by $\mathcal{V}$ via polynomial commitments.

Efficient Proofs for Convolution

Our Proofs for Convolution

We express convolutions between input X and kernel W as multilinear sumchecks.



Source: Christopher Melen, RNCM

Our Proofs for Convolution

Our protocol proceeds in two steps:

1. A *reshape* sumcheck that rearranges $X \mapsto \hat{X}$.
2. A *convolution* sumcheck $\hat{X} \circ W \mapsto Y$.

Built upon sumchecks for matrix multiplication [Tha13] and channel batching.

Our Proofs for Convolution

Our protocol proceeds in two steps:

1. A *reshape* sumcheck that rearranges $X \mapsto \hat{X}$.
2. A *convolution* sumcheck $\hat{X} \circ W \mapsto Y$.

Built upon sumchecks for matrix multiplication [Tha13] and channel batching.

Performance

For c input channels, d output channels,

	Ours	**zkCNN [LXZ21]**
Prover	$\mathcal{O}\left(c \cdot \lvert W\rvert \cdot (\lvert Y\rvert + d)\right)$	$\mathcal{O}\left(c \cdot d \cdot \lvert X\rvert\right)$
Verifier	$\mathcal{O}\left(\log(c \cdot \lvert Y\rvert)\right)$	$\mathcal{O}\left(\log^2(c \cdot d \cdot \lvert X\rvert)\right)$
Size	$\mathcal{O}\left(\log(c \cdot \lvert Y\rvert)\right)$	$\mathcal{O}\left(\log^2(c \cdot d \cdot \lvert X\rvert)\right)$

Applications and Benchmarking

Applications

We extend our framework to construct efficient proof systems for:

- **Convolutional Neural Networks**.
- **Recurrent NNs**.
- **Image Processing:** Native linear, reshaping, and convolutional operations (filtering, blurring...).

Our convolution prover and a general CNN prover are implemented in Rust and available open-source.

Benchmarking

- **Prover** $\approx$ 0.1s for 256×256 input and 4×4 kernel.
  5$\times$ *faster than zkCNN*, 100$\times$ *faster than vCNN.*

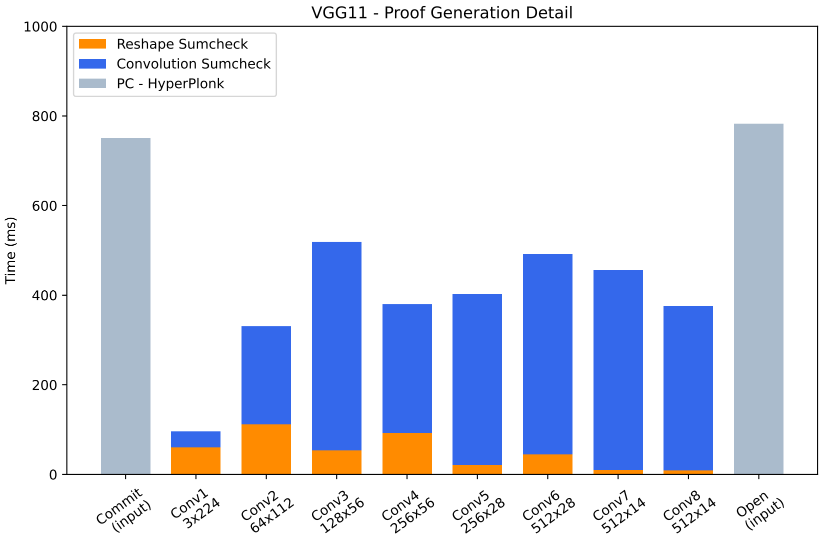
Benchmarking

- **Prover** $\approx$ 0.1s for 256×256 input and 4×4 kernel.
  5$\times$ *faster than zkCNN*, 100$\times$ *faster than vCNN.*
- **Verification** $\approx$ 0.1ms.

Benchmarking

- **Prover** $\approx$ 0.1s for 256×256 input and 4×4 kernel.
  5$\times$ *faster than zkCNN*, 100$\times$ *faster than vCNN.*
- **Verification** $\approx$ 0.1ms.
- **Proof size** $\approx$ 1KB.
  10$\times$ *shorter than zkCNN.*

Sequential composition reduces memory usage. Still room for improvement.



All kernels are 3×3. Run on single-core Xeon-Gold-6154 at 3GHz.

VGG11

Conv1: 3×224^2

Conv2: 64×112^2

Conv3: 128×56^2

Conv4: 256×56^2

Conv5: 256×28^2

Conv6: 512×28^2

Conv7: 512×14^2

Conv8: 512×14^2

Conclusions

- Theory **framework** for composition of sumcheck-based proofs.
- Efficient sumchecks for **convolution**.
- Efficient arguments for **data-intensive applications** such as ML and IP.
- Modular **implementation** in Rust.

Conclusions

- Theory **framework** for composition of sumcheck-based proofs.
- Efficient sumchecks for **convolution**.
- Efficient arguments for **data-intensive applications** such as ML and IP.
- Modular **implementation** in Rust.

Thank you!

`ia.cr/2023/1342`

`david.balbas@imdea.org`

Slides available @ davidbalbas.github.io

Additional Material

VE for Multilinear Polynomials

Let $\boldsymbol{t} = (t_1, \ldots, t_\ell)$, $\mathbb{F}$ a finite field, and

$$x(\boldsymbol{t}, \boldsymbol{y}) = \prod_{i=1}^{s} x_i(\boldsymbol{t}, \boldsymbol{y})$$

where each x_i is a multilinear polynomial over $\mathbb{F}$.

We can generalize multilinear sumcheck as a VE for the relation

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

$\mathcal{P}, \mathcal{V}$ *start on fingerprint* $c_y = f_y(\boldsymbol{r}_y)$. At the end, they *obtain* $c_x = x(\boldsymbol{r}_t, \boldsymbol{r}_y)$.

VE for Matrix Multiplication [Tha13]

Let $C = A \cdot B$ where $A, B, C \in \mathbb{F}^{n \times n}$. We can write matrix multiplication as

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

Where $\tilde{A}$ encodes A as a (unique) multilinear polynomial, $\tilde{A}(\boldsymbol{i}, \boldsymbol{j}) = A_{i,j}$

Given $\boldsymbol{r}_1, \boldsymbol{r}_2 \in \mathbb{F}^\ell$, we apply our multilinear sumcheck VE on:

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

$\mathcal{P}, \mathcal{V}$ *start on fingerprint* $c_C = \tilde{C}(\boldsymbol{r}_1, \boldsymbol{r}_2)$. At the end, they *obtain fingerprints* $c_A = \tilde{A}(\boldsymbol{r}_1, \boldsymbol{r}_3)$ *and* $c_B = \tilde{B}(\boldsymbol{r}_3, \boldsymbol{r}_2)$.

Communication and verifier $t_V = O(\ell) = O(\log n)$, prover $O(n^2)$.

VE for Convolution

Convolution equations can be compacted as

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

For **multiple kernels and inputs** (multi-channel), as usual in e.g. neural networks,

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

Where $\sigma \in [c]$ represents input channels, and $\tau \in [d]$ output channels.