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

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 $L_F = f(f;x;y) : f(x) = yg$ is a pair of algorithms $hP;Vi(f;x;y) \; ! \; b$ that are *complete* and *sound*.

Fingerprints and Interactive Proofs

An **interactive proof** (IP) for the language $L_F = f(f;x;y) : f(x) = yg$ is a pair of algorithms $hP;Vi(f;x;y) \;!\; b$ that are *complete* and *sound*.

Fingerprint

Let c_x H$(x;r)$ be the **fingerprint** of x on r.
H is (statistically) sound if for *any* pair x 6= x ,

$$\Pr_r[\mathrm{H}(x;r) = \mathrm{H}(x\ ;r)] = \mathrm{negl}(\ ):$$

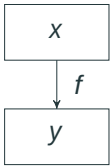
Example: for x 2 F^n, the poly. evaluation $\mathrm{H}(x;r) = x_0 + x_1 r +$ $+ x_n\ _1 r^n\ ^1$ over F.

Structure of Common IPs

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

$\hbar P; Vi(f; x; y)$

Prover	**Verifier**
$c_y \quad H(y; r_y)$	$c_y \quad H(y; r_y)$

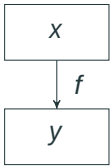


Structure of Common IPs

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

$\langle P; Vi(f; x; y)$

Prover		**Verifier**
c_y $\quad H(y; r_y)$		c_y $\quad H(y; r_y)$
$c_x; r_x$	$\langle P_{VE}(x); V_{VE}\, i(f; c_y; r_y)$	$c_x; r_x; b$

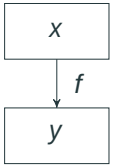


Structure of Common IPs

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

$\langle P; V \rangle i(f; x; y)$

Prover		**Verifier**
c_y	$\mathsf{H}(y; r_y)$	c_y $\quad \mathsf{H}(y; r_y)$
$c_x; r_x$	$\langle P_{\mathsf{VE}}(x); V_{\mathsf{VE}} \rangle i(f; c_y; r_y)$	$c_x; r_x; b$
		b' $\quad [c_x = \mathsf{H}(x; r_x)]$
		return $b \wedge b'$

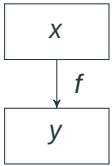


Structure of Common IPs

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

$\langle P; VI(f; x; y)$

Prover		**Verifier**
c_y $\quad H(y; r_y)$		c_y $\quad H(y; r_y)$
$c_x; r_x$	$\langle P_{VE}(x); V_{VE}\, i(f; c_y; r_y)$	$c_x; r_x; b$



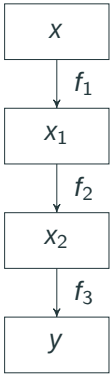
Subroutines named **verifiable evaluation schemes** *(VE) on ngerprinted data*.

Structure of Common IPs

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

$\langle P; V \rangle^{i}(f; x; y)$

Prover		**Verifier**
$c_y \quad H(y; r_y)$		$c_y \quad H(y; r_y)$
$c_2; r_2$	$\langle P_{VE}(x_2); V_{VE}^{i}(f_3; c_y; r_y)$	$c_2; r_2; b_2$
$c_1; r_1$	$\langle P_{VE}(x_1); V_{VE}^{i}(f_2; c_2; r_2)$	$c_1; r_1; b_1$
$c_x; r_x$	$\langle P_{VE}(x); V_{VE}^{i}(f_1; c_1; r_1)$	$c_x; r_x; b_0$



Subroutines named **verifiable evaluation schemes** *(VE) on ngerprinted 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:

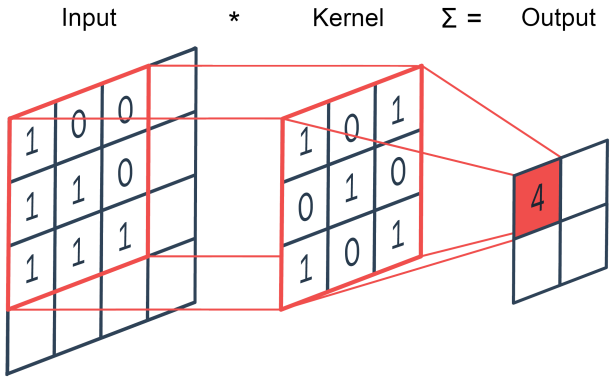
- Matrix multiplication [Thaler13]
- GKR [GKR08, CMT12]
- Libra [XZZPS19], Virgo [ZLW+20] (and follow-ups)
- FFT-based convolution [LXZ21]
- *...*

We provide a compiler *from VEs to succinct cryptographic arguments*. Fingerprints can be evaluated efficiently by 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 $X \overset{?}{=} \hat{X}$.
2. A *convolution* sumcheck $\hat{X} \quad W \overset{?}{=} Y$.

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 $X \overset{?}{=} \hat{X}$.
2. A *convolution* sumcheck $\hat{X} \quad W \overset{?}{=} Y$.

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

Performance

For c input channels, d output channels,

	Ours	zkCNN [LXZ21]
Prover	$O \; c \;\; jWj \;\; (jYj + d)$	$O \; c \;\; d \;\; jXj$
Verifier	$O \; \log(c \;\; jYj)$	$O \; \log^2(c \;\; d \;\; jXj)$
Size	$O \; \log(c \;\; jYj)$	$O \; \log^2(c \;\; d \;\; jXj)$

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** 0.1s for 256 256 input and 4 4 kernel.
  5 *faster than zkCNN,* 100 *faster than vCNN.*

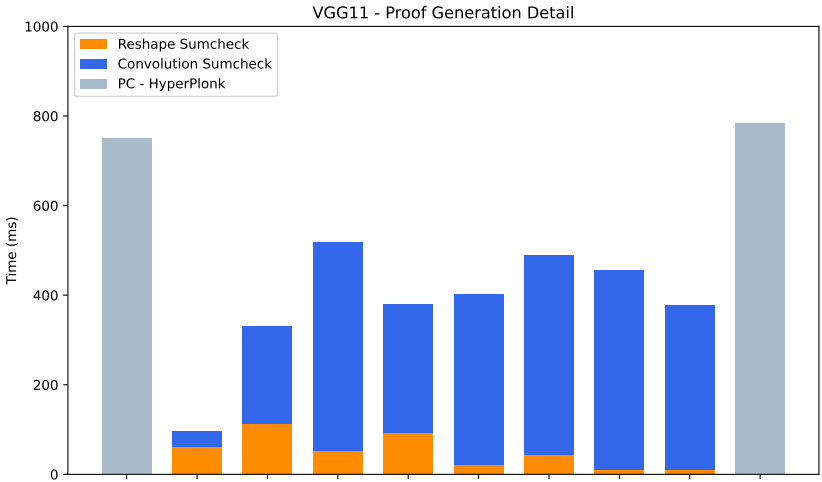
Benchmarking

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

Benchmarking

- **Prover** 0.1s for 256 × 256 input and 4 × 4 kernel.
  5 × *faster than zkCNN*, 100 × *faster than vCNN*.
- **Verification** 0.1ms.
- **Proof size** 1KB.
  10 × *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 $t = (t_1; \ldots; t_\ell)$, F a finite field, and

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

where each x_i is a multilinear polynomial over F.

We can generalize multilinear sumcheck as a VE for the relation

$$f_y(r_y) = \sum_{t \in \{0,1\}^\ell} x(t; r_y):$$

$P; V$ *start on ngerprint* $c_y = f_y(r_y)$. At the end, they *obtain* $c_x = x(r_t; r_y)$.

VE for Matrix Multiplication [Tha13]

Let $C = A \ B$ where $A; B; C \ 2 \ \mathbb{F}^{n \ n}$. We can write matrix multiplication as

$$\tilde{C}(x_1; x_2) = \sum_{y \, 2 \, f0;1g^{`}} \tilde{A}(x_1; y) \ \tilde{B}(y; x_2)$$

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

Given $r_1; r_2 \ 2 \ \mathbb{F}^{`}$, we apply our multilinear sumcheck VE on:

$$\tilde{C}(r_1; r_2) = \sum_{y \, 2 \, f0;1g^{`}} \tilde{A}(r_1; y) \ \tilde{B}(y; r_2):$$

$P; V$ *start on ngerprint* $c_C = \tilde{C}(r_1; r_2)$. At the end, they *obtain ngerprints* $c_A = \tilde{A}(r_1; r_3)$ *and* $c_B = \tilde{B}(r_3; r_2)$.

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

VE for Convolution

Convolution equations can be compacted as

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

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

$$Y = [Y_1 j \quad j Y_d] = \sum_{=1}^{\hat{X}} \hat{X} \quad [W_{;1} j \quad j W_{;d}]:$$

Where $2\ [c]$ represents input channels, and $2\ [d]$ output channels.