# Speeding Up Sum-Check Proving

Quang Dao
CMU

Joint work with Zachary DeStefano (NYU), Justin Thaler (a16z),
Suyash Bagad and Yuval Domb (Ingonyama)

*CMU Crypto Seminar, Nov 2025*

# The Verifiability Problem



Discord's face scanning age checks 'start of a bigger shift'

The Problem with Science: The Reproducibility Crisis and What to do About It

*Can We No Longer Believe Anything We See?*

Can Metadata Standards Like C2PA Fight Deepfakes?

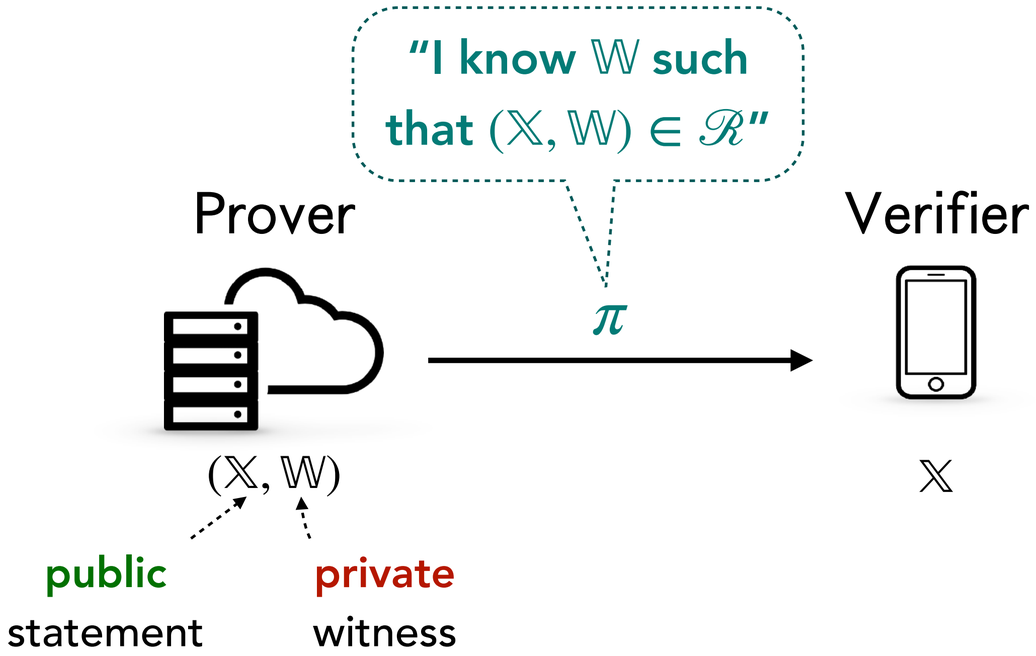
The Secret Bias Hidden in Mortgage-Approval Algorithms

How can we ***know*** a computation was carried out correctly…

…without seeing all of its inputs?

…without re-executing it ourselves?

# Cryptographic Proof Systems




**Completeness:** If $(\mathbb{X}, \mathbb{W})$ is satisfying, and P generates proofs honestly, then V accepts

**Knowledge Soundness:** If P* produces $\pi$ such that V accepts, then any P* must *"know"* a satisfying $\mathbb{W}$.

**Succinctness:** $\pi$ is short & V verifies $\pi$ faster than checking $(\mathbb{X}, \mathbb{W})$ is satisfying

**Zero-Knowledge:** $\pi$ reveals nothing other than the fact that $(\mathbb{X}, \mathbb{W})$ is satisfying

*[GMR85] The Knowledge Complexity of Interactive Proof Systems

# Examples of Proving Tasks

**What is $(\mathbb{X}, \mathbb{W}) \in \mathscr{R}$ in the real world?**

"My digital ID shows that I am above 18"

"The AI model with these weights produced this output"

"A batch of transactions on Ethereum executed correctly"

"This image is captured by a real camera & not AI generated"

...

$\Longrightarrow$ **Zero-Knowledge Virtual Machine (zkVM)**

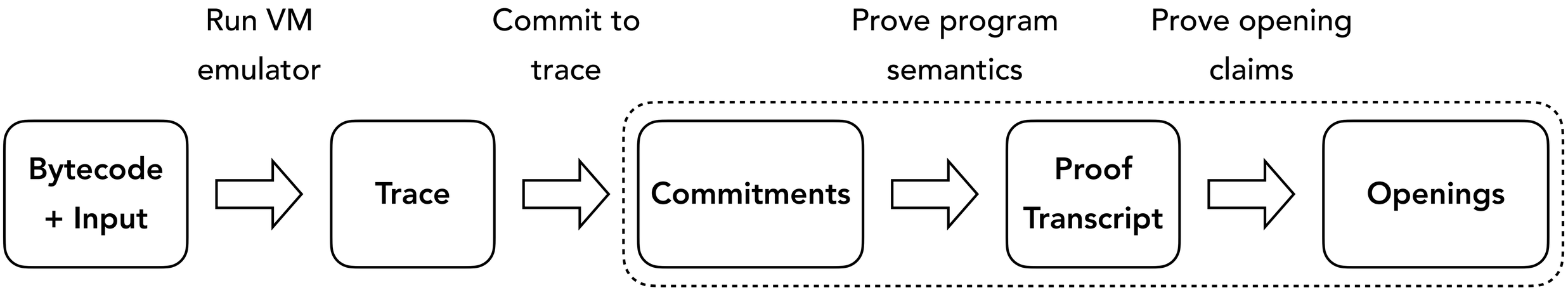
**A program, when executed on this input, produces this claimed output**

a sequence of instructions
in some ISA (i.e. RISC-V)

may be a mix of public & private

# zkVM Proving Pipeline



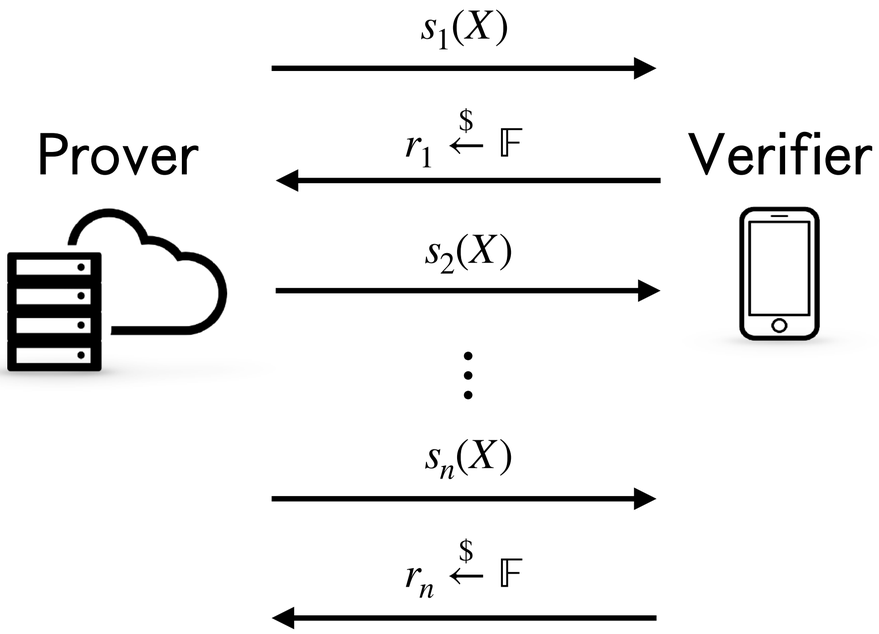

**Are there common sub-protocols between different zkVMs?**

**What are the bottlenecks?**

# The Sum-Check Protocol

known to both P and V

**Setup:** $\mathbb{F}$ = finite field, $p \in \mathbb{F}^{\leq d}[X_1, \ldots, X_n]$, $c \in \mathbb{F}$




**Input Claim:** $\sum_{x_1 \in H_1} \cdots \sum_{x_n \in H_n} p(x_1, \ldots, x_n) = c$

(usually $H_1 = \ldots = H_n = \{0,1\}$)

**Output Claim:** $p(r_1, \ldots, r_n) = c'$

*[LFKN92] Algebraic Methods for Interactive Proof Systems

# The Sum-Check Protocol




Jolt - the sum-check-based zkVM for 64-bit RISC-V

$\approx 600$ KHz on a Macbook, $\approx 1.8$ MHz on a 64-core server

70% of total prover time!




Can we speed up sum-check proving?

*[LFKN92] Algebraic Methods for Interactive Proof Systems

*proving repeated SHA3 computation in Jolt, totalling ~1 billion RISC-V cycles

# Our Results

A **suite** of optimizations to various settings of **sum-check proving**

- Especially in the context of zkVMs such as **Jolt**

**Key ideas:**

- Non-black-box handling of field arithmetic
- Batching computation across rounds

} This talk

- Recursive, tree-based algorithm for high-degree polynomial product

**Concrete & asymptotic speedups:**

- 40% over classic algorithm in generic setting, 3x in zkVM setting
- $O(d^2) \mapsto O(d \log d)$ for degree, poly($N_{limbs}$) factor with "small" polys

# Talk Outline

1. The Sum-Check Protocol and Existing Algorithms

2. New Techniques for Sum-Check Proving

3. Evaluation & Summary

# Talk Outline

1. The Sum-Check Protocol and Existing Algorithms

2. New Techniques for Sum-Check Proving

3. Evaluation & Summary

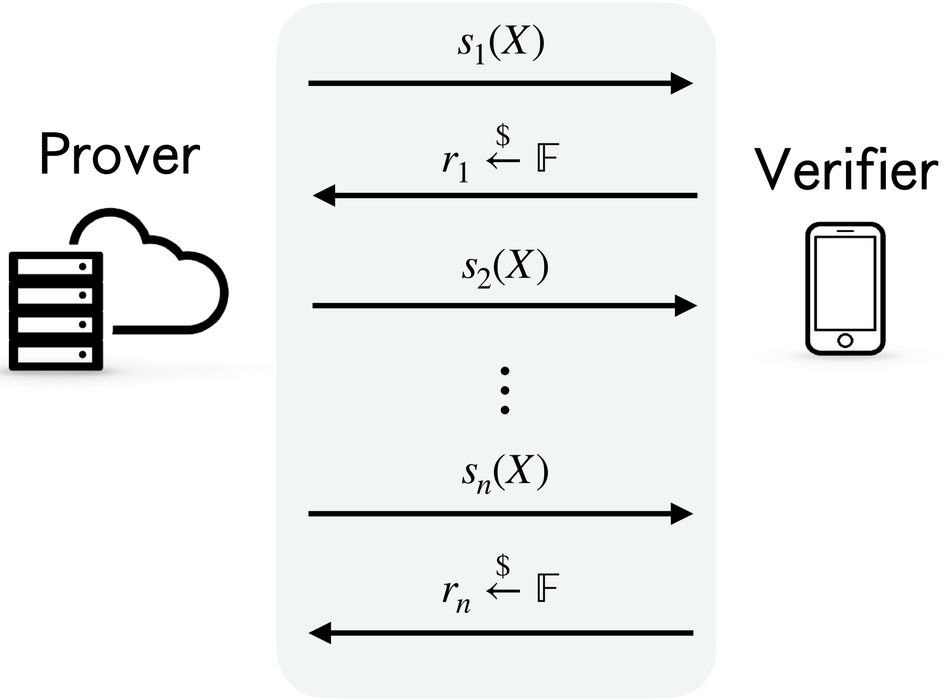
# The Sum-Check Protocol

**Setup:** $\mathbb{F}$ = finite field,

$p \in \mathbb{F}^{\leq d}[X_1, \ldots, X_n]$, $c \in \mathbb{F}$

(known to both P and V)

**Input Claim:** $\sum_{(x_1,\ldots,x_n)\in\{0,1\}^n} p(x_1, \ldots, x_n) = c_0$




For each round $i = 1, \ldots, n$:

1. P sends $s_i(X) = \sum_{x'\in\{0,1\}^{n-i-1}} P(r_1, \ldots, r_{i-1}, X, x')$
2. V sends $r_i \xleftarrow{\$} \mathbb{F}$
3. V rejects if $\deg s > d$ or $s_i(0) + s_i(1) \neq c_{i-1}$, otherwise sets new claim $c_i := p_i(r_i)$.

**Output Claim:** $p(r_1, \ldots, r_n) = c_n$

**Soundness:** if input claim is false, then output claim also false except with probability $\leq \frac{n \cdot d}{|\mathbb{F}|}$

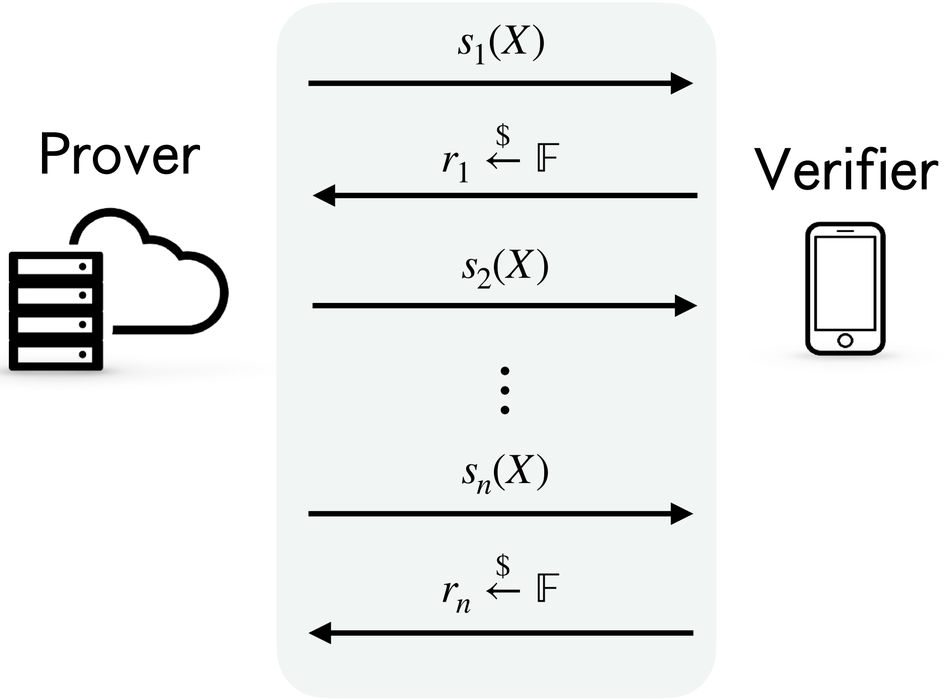
# The Sum-Check Protocol

**Setup:** $\mathbb{F}$ = finite field,

$p \in \mathbb{F}^{\leq d}[X_1, \ldots, X_n]$, $c \in \mathbb{F}$

(known to both P and V)

**Input Claim:** $\sum_{(x_1,\ldots,x_n)\in\{0,1\}^n} p(x_1, \ldots, x_n) = c_0$




For each round $i = 1, \ldots, n$:

1. P computes $s_i(u) = \sum_{x'\in\{0,1\}^{n-i-1}} P(r_1, \ldots, r_{i-1}, u, x')$

   for $u \in \{1,2,\ldots,d\}$, and sends $\{s_i(u) \mid u \in \{1,\ldots,d\}\}$

2. V sends $r_i \xleftarrow{\$} \mathbb{F}$

3. V rejects if it does not receive exactly $d$ field elements. Then V derives $s_i(0) = c_{i-1} - s_i(1)$, interpolates $s_i(X)$, then sets next claim $c_i = s_i(r_i)$.

**Output Claim:** $p(r_1, \ldots, r_n) = c_n$

# Primer on Multilinear Polynomials

- **Multilinear polynomial:** an algebraic way to interpret vectors

$$\vec{v} = (v_0, v_1, \ldots, v_{2^n-1}) \in \mathbb{F}^{2^n}$$

- **Monomial encoding:**

$$\hat{v}(X_1, \ldots, X_n) = v_0 + v_1 X_1 + v_2 X_1 X_2 + \ldots + v_{2^n-1} (X_1 \cdot \ldots \cdot X_n)$$

- **Evaluation encoding:** $\Longleftarrow$ what we will use

$\widetilde{v}(X_1, \ldots, X_n)$ such that $\widetilde{v}(b_1, \ldots, b_n) = v_{(2^{n-1}b_1 + 2^{n-2}b_2 + \ldots + b_n)}$ for all $(b_1, \ldots, b_n) \in \{0,1\}^n$

(example: $\widetilde{v}(0,0) = v_0$, $\widetilde{v}(0,1) = v_1$, $\widetilde{v}(1,0) = v_2$, $\widetilde{v}(1,1) = v_3$)

- **Explicit formula for $\widetilde{v}(X_1, \ldots, X_n)$ ?**

# Primer on Multilinear Polynomials

- **Equality polynomial:** indicator for the $\{0,1\}^n$ evaluation domain

$$\widetilde{eq}\left((X_1,\ldots,X_n),(Y_1,\ldots,Y_n)\right)=\prod_{i=1}^{n}\left((1-X_i)\cdot(1-Y_i)+X_i\cdot Y_i\right)$$

- **Key property:** for all $a,b\in\{0,1\}^n$, $\quad\widetilde{eq}(a,b)=\begin{cases}1 & \text{if } a=b\\ 0 & \text{otherwise}\end{cases}$

- **Multilinear extension formula:**

$$\widetilde{v}(X_1,\ldots,X_n)=\sum_{b\in\{0,1\}^n} v_{\text{bin_num}(b)}\cdot\widetilde{eq}(\vec{X},b)$$

(**example:** $\widetilde{v}(X_1,X_2)=v_0\cdot(1-X_1)(1-X_2)+v_1\cdot(1-X_1)X_2+v_2\cdot X_1(1-X_2)+v_3\cdot X_1X_2$ )

# Example Sum-Check Application: Zero-Check

- **Inputs**: $\vec{a} = (a_0, \ldots, a_{2^n-1}), \quad \vec{b} = (b_0, \ldots, b_{2^n-1})$
- **Claim:** $a_i \cdot b_i = 0 \quad \forall\, i = 0,\ldots,2^n - 1 \quad \iff \quad \widetilde{a}(x) \cdot \widetilde{b}(x) = 0 \quad \forall\, x \in \{0,1\}^n$
- **Take the multilinear extension:**

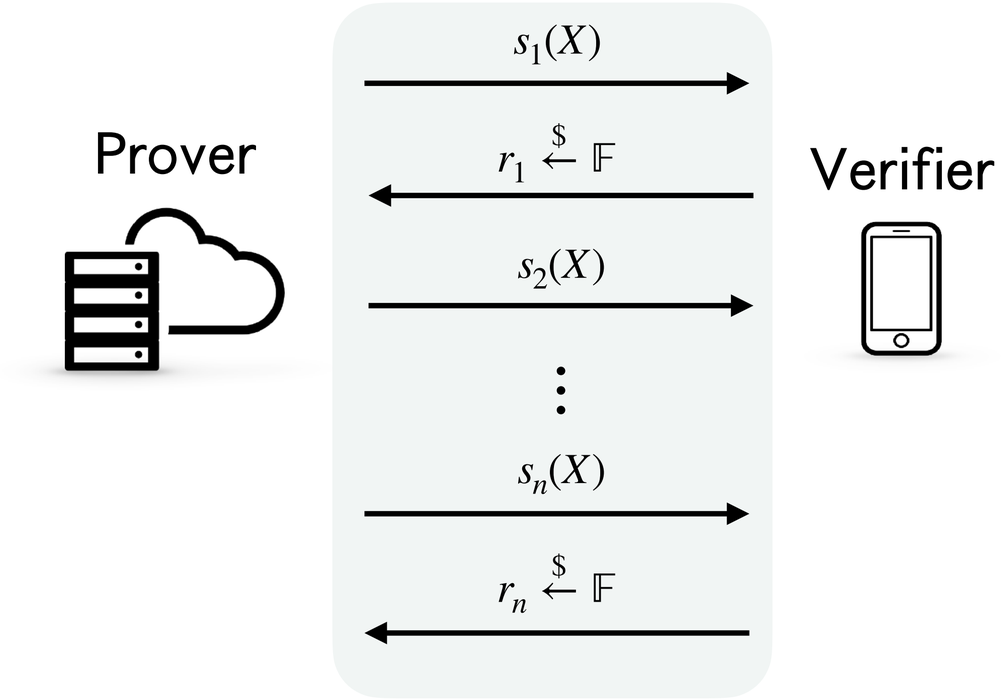
$$\sum_{x \in \{0,1\}^n} \widetilde{\mathsf{eq}}(\vec{X}, x) \cdot \widetilde{a}(x) \cdot \widetilde{b}(x) = 0$$

- **Randomized check at a random point $\tau \xleftarrow{\$} \mathbb{F}^n$ :**

$$\sum_{x \in \{0,1\}^n} \widetilde{\mathsf{eq}}(\tau, x) \cdot \widetilde{a}(x) \cdot \widetilde{b}(x) = 0$$

# Classic Linear-Time Sum-Check Algorithm

**Input Claim:** $\sum_{(x_1,\ldots,x_n)\in\{0,1\}^n} p(x_1, \ldots, x_n) = c_0$



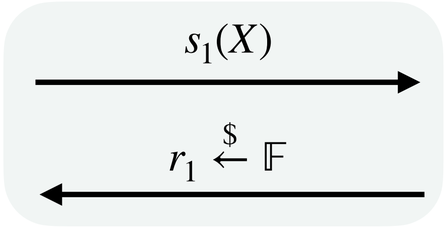

**Output Claim:** $p(r_1, \ldots, r_n) = c_n$

*[Thaler13] Time-Optimal Interactive Proofs for Circuit Evaluation

*[VSBW13] A Hybrid Architecture for Verifiable Computation

# Classic Linear-Time Sum-Check Algorithm

**Input Claim:** $\sum_{(x_1,\ldots,x_n)\in\{0,1\}^n} p(x_1, \ldots, x_n) = c_0$




**Claim 1:** $\sum_{(x_2,\ldots,x_n)\in\{0,1\}^{n-1}} p(r_1, x_2, \ldots, x_n) = c_1$

$\vdots$

**Output Claim:** $p(r_1, \ldots, r_n) = c_n$

**Idea:** from

$$\Big(p(x_1, x_2, \ldots, x_n)\Big)_{(x_1,\ldots,x_n)\in\{0,1\}^n},$$

derive

$$\Big(p(r_1, x_2, \ldots, x_n)\Big)_{(x_2,\ldots,x_n)\in\{0,1\}^{n-1}},$$

then recurse

**Specialize:** low-degree functions of multilinear polynomials

e.g., $\sum_{x\in\{0,1\}^n} p(x) \cdot q(x) = c$

*[Thaler13] Time-Optimal Interactive Proofs for Circuit Evaluation

*[VSBW13] A Hybrid Architecture for Verifiable Computation

# Classic Linear-Time Sum-Check Algorithm

**Claim:** $\sum_{x \in \{0,1\}^n} p(x) \cdot q(x) = c$

**Inputs:** $\big(p(x)\big)_{x \in \{0,1\}^n}$, $\big(q(x)\big)_{x \in \{0,1\}^n}$

1. Compute first polynomial $s_1(X) = \sum_{x' \in \{0,1\}^{n-1}} p(X, x') \cdot q(X, x')$

$$s_1(1) = \sum_{x' \in \{0,1\}^{n-1}} p(1,x') \cdot q(1,x')$$

$$s_1(2) = \sum_{x' \in \{0,1\}^{n-1}} p(2,x') \cdot q(2,x') = \sum_{x' \in \{0,1\}^{n-1}} (2 \cdot p(1,x') - p(0,x')) \cdot (2 \cdot q(1,x') - q(0,x'))$$

# Classic Linear-Time Sum-Check Algorithm

**Claim:** $\sum_{x \in \{0,1\}^n} p(x) \cdot q(x) = c$ **Inputs:** $\big(p(x)\big)_{x \in \{0,1\}^n}$, $\big(q(x)\big)_{x \in \{0,1\}^n}$

1. Compute first polynomial $s_1(\{1,2\}) = \sum_{x' \in \{0,1\}^{n-1}} p(\{1,2\}, x') \cdot q(\{1,2\}, x')$

2. **Bind** $p, q$ to first challenge $r_1 \xleftarrow{\$} \mathbb{F}$

$$p_1(x') = p(r_1, x') = (1 - r_1) \cdot p(0,x') + r_1 \cdot p(1,x') = p(0,x') + r_1 \cdot (p(1,x') - p(0,x'))$$

$$q_1(x') = q(r_1, x') = (1 - r_1) \cdot q(0,x') + r_1 \cdot q(1,x') = q(0,x') + r_1 \cdot (q(1,x') - q(0,x'))$$

3. Recurse on $\big(p_1(x')\big)_{x' \in \{0,1\}^{n-1}}$, $\big(q_1(x')\big)_{x' \in \{0,1\}^{n-1}}$ with claim $\sum_{x' \in \{0,1\}^{n-1}} p_1(x') \cdot q_1(x') = c'$

# Classic Linear-Time Sum-Check Algorithm

**Claim:** $\sum_{x \in \{0,1\}^n} p(x) \cdot q(x) = c$ **Inputs:** $\big(p(x)\big)_{x \in \{0,1\}^n}$, $\big(q(x)\big)_{x \in \{0,1\}^n}$

1. Compute first polynomial $s_1(\{1,2\}) = \sum_{x' \in \{0,1\}^{n-1}} p(\{1,2\}, x') \cdot q(\{1,2\}, x')$

$2 \cdot 2^{n-1}$ field multiplications

2. Bind $p, q$ to first challenge $r_1 \xleftarrow{\$} \mathbb{F}$

$p_1(x') = p(r_1, x') = (1 - r_1) \cdot p(0, x') + r_1 \cdot p(1, x') = p(0, x') + r_1 \cdot (p(1, x') - p(0, x'))$

$q_1(x' = q(r_1, x') = (1 - r_1) \cdot q(0, x') + r_1 \cdot q(1, x') = q(0, x') + r_1 \cdot (q(1, x') - q(0, x'))$

Total Time: $4 \cdot (2^{n-1} + 2^{n-2} + \ldots + 1) \approx 4 \cdot 2^n$ field multiplications

Max Space: $O(2^n)$ field elements

3. Recurse on $\big(p_1(x')\big)_{x' \in \{0,1\}^{n-1}}$, $\big(q_1(x')\big)_{x' \in \{0,1\}^{n-1}}$ with claim $\sum_{x' \in \{0,1\}^{n-1}} p_1(x') \cdot q_1(x') = c'$

# Talk Outline

1. The Sum-Check Protocol and Existing Algorithms

2. New Techniques for Sum-Check Proving

3. Evaluation & Summary

# Summary of Improvements

**How can we improve on the linear-time algorithm?**

- Break down the black box of finite field arithmetic
  - Modify field operations to align with the sum-check proving algo

- Capitalize on the properties of the polynomials arising in practice
  - E.g., zkVM program values are $64$ bits over a $\approx 256$ - bit field

# Primer on Field Arithmetic

**For this talk:** $\approx 256$ - bit fields, specifically

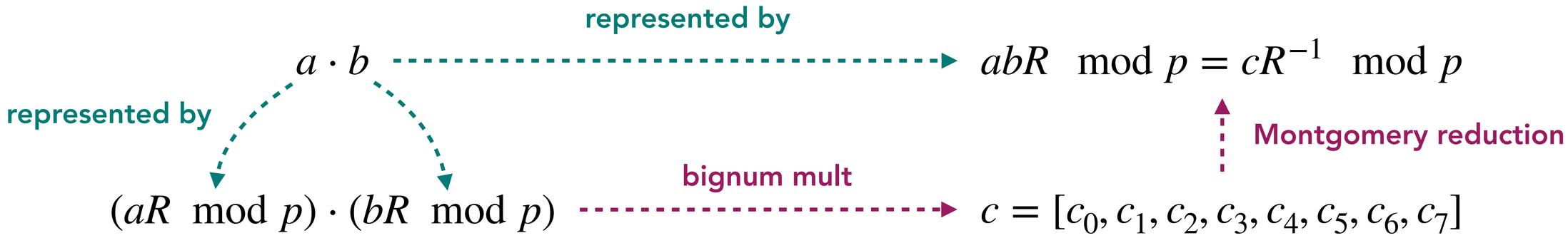
$|\mathbb{F}_{BN254}| = 21888242871839275222246405745257275088548364400416034343698204186575808495617$

**Field element storage:** $[a_0, a_1, a_2, a_3]$ where $a_0, a_1, a_2, a_3$ are 64-bit unsigned integers

such that $a_0 + a_1 \cdot 2^{64} + a_2 \cdot 2^{128} + a_3 \cdot 2^{192} <$ BN254_modulus

**Montgomery representation:** $a \in \{0,1,\ldots,p-1\}$ is stored as $aR \mod p$, where $R = 2^{256}$

**Montgomery multiplication:** given $a, b \in \{0,\ldots,p-1\}$



# Primer on Field Arithmetic

**For this talk:** $\approx 256$ - bit fields, specifically

$|\mathbb{F}_{BN254}| = 21888242871839275222246405745257275088548364400416034343698204186575808495617$

**Field element storage:** $[a_0, a_1, a_2, a_3]$ where $a_0, a_1, a_2, a_3$ are 64-bit unsigned integers

such that $a_0 + a_1 \cdot 2^{64} + a_2 \cdot 2^{128} + a_3 \cdot 2^{192} <$ BN254_modulus

**Montgomery representation:** $a \in \{0,1,\ldots,p-1\}$ is stored as $aR \mod p$, where $R = 2^{256}$

**Montgomery multiplication:** given $a, b \in \{0,\ldots,p-1\}$

*technically $u64 \times u64 \to u128$



# Technique 1a: Delayed Reduction

**Recall prover's message:**

**Claim:** $\sum_{x \in \{0,1\}^n} p(x) \cdot q(x) = c$

$$s_1(1) = \sum_{x' \in \{0,1\}^{n-1}} p(1,x') \cdot q(1,x')$$

bignum add + conditional subtract

$$= p(1,0\ldots00) \cdot q(1,0\ldots00) + p(1,0\ldots1) \cdot q(1,0\ldots01) + \ldots + p(1,1\ldots11) \cdot q(1,1\ldots11)$$

bignum mult + Mont reduce | bignum mult + Mont reduce | bignum mult + Mont reduce

$$s_1(2) = \sum_{x' \in \{0,1\}^{n-1}} p(2,x') \cdot q(2,x') = \sum_{x' \in \{0,1\}^{n-1}} (2 \cdot p(1,x') - p(0,x')) \cdot (2 \cdot q(1,x') - q(0,x'))$$

# Technique 1a: Delayed Reduction

**Recall prover's message:**

**Claim:** $\sum_{x \in \{0,1\}^n} p(x) \cdot q(x) = c$

$$s_1(1) = \sum_{x' \in \{0,1\}^{n-1}} p(1,x') \cdot q(1,x')$$

$$= p(1,0\ldots00) \cdot q(1,0\ldots00) + p(1,0\ldots1) \cdot q(1,0\ldots01) + \ldots + p(1,1\ldots11) \cdot q(1,1\ldots11)$$

bignum add

bignum mult

bignum mult

bignum mult

final Mont reduce

$2\times$ reduction in $u64$ mults!

$$s_1(2) = \sum_{x' \in \{0,1\}^{n-1}} p(2,x') \cdot q(2,x') + \sum_{x' \in \{0,1\}^{n-1}} (2 \cdot p(1,x') - p(0,x')) \cdot (2 \cdot q(1,x') - q(0,x'))$$

similar!

# Technique 1b: Special Challenge Set

**Recall prover's bind:**

**Claim:** $\sum_{x \in \{0,1\}^n} p(x) \cdot q(x) = c$

$$p_1(x') = p(r_1, x') = (1 - r_1) \cdot p(0,x') + r_1 \cdot p(1,x') = p(0,x') + r_1 \cdot (p(1,x') - p(0,x'))$$

$$q_1(x') = q(r_1, x') = (1 - r_1) \cdot q(0,x') + r_1 \cdot q(1,x') = q(0,x') + r_1 \cdot (q(1,x') - q(0,x'))$$

**Can we make $r_1 \cdot (\ldots)$ strictly faster?**

**Recall: cheating probability** $\leq \frac{2n}{|\text{ChalSet}|}$

**Key observation:** for $\approx 128$ bits of security, we only need $r_1$ drawn from a $\approx 2^{128}$ - sized subset of $\mathbb{F}$

$\Longrightarrow$ which subset can speed up multiplication?

# Technique 1b: Special Challenge Set

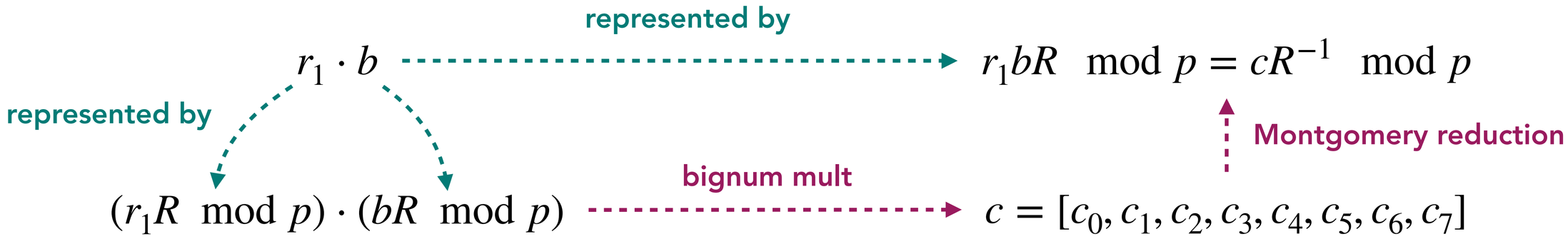
Recall prover's bind:

**Claim:** $\sum_{x \in \{0,1\}^n} p(x) \cdot q(x) = c$

$$p_1(x') = p(r_1, x') = (1 - r_1) \cdot p(0,x') + r_1 \cdot p(1,x') = p(0,x') + r_1 \cdot (p(1,x') - p(0,x'))$$

$$q_1(x') = q(r_1, x') = (1 - r_1) \cdot q(0,x') + r_1 \cdot q(1,x') = q(0,x') + r_1 \cdot (q(1,x') - q(0,x'))$$

**Answer:** sample $r_1$ from the set of $a \in \{0, \ldots, p-1\}$ such that $aR \mod p = [0, 0, a_2, a_3]$ !



# Technique 1b: Special Challenge Set

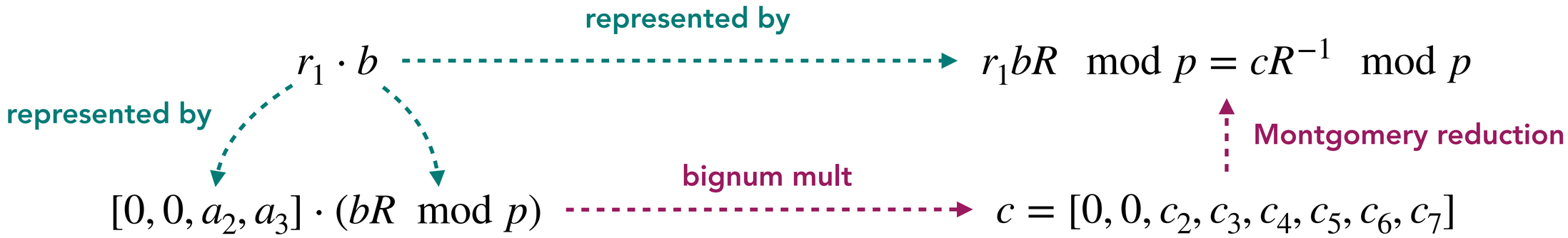
Recall prover's bind:

**Claim:** $\sum_{x \in \{0,1\}^n} p(x) \cdot q(x) = c$

$$p_1(x') = p(r_1, x') = (1 - r_1) \cdot p(0,x') + r_1 \cdot p(1,x') = p(0,x') + r_1 \cdot (p(1,x') - p(0,x'))$$

$$q_1(x') = q(r_1, x') = (1 - r_1) \cdot q(0,x') + r_1 \cdot q(1,x') = q(0,x') + r_1 \cdot (q(1,x') - q(0,x'))$$

**Answer:** sample $r_1$ from the set of $a \in \{0, \ldots, p-1\}$ such that $aR \mod p = [0, 0, a_2, a_3]$ !



# Technique 1b: Special Challenge Set

Recall prover's bind:

**Claim:** $\sum p(x) \cdot q(x) = c$

$c = [0, 0, c_2, c_3, c_4, c_5, c_6, c_7]$

one step (5 $u64$ mults)

$[0, 0, 0, c_3', c_4', c_5', c_6', c_7']$

5 $u64$ mults

$[0, 0, 0, 0, c_4'', c_5'', c_6'', c_7'']$

conditional subtract

$cR^{-1} \mod p = [c_4''', c_5''', c_6''', c_7''']$

**Answer:** sa… $[0, 0, a_2, a_3]$ !

another $2\times$ reduction in $u64$ mults!

represented by

$r_1 \cdot b$

$[0, 0, a_2, a_3] \cdot (bR \mod p)$

8 $u64$ mults

$c = [0, 0, c_2, c_3, c_4, c_5, c_6, c_7]$

10 $u64$ mults

$\mod p = cR^{-1} \mod p$

# Summary of Improvements

**How can we improve on the linear-time algorithm?**

- Break down the black box of finite field arithmetic
  - Modify field operations to align with the sum-check proving algo

$\Longrightarrow 2\times$ **reduction in number of** $u64$ **mults**

- Capitalize on the properties of the polynomials arising in practice
  - E.g., zkVM program values are $64$ bits over a $\approx 256$ - bit field

# Technique 2a: Round Batching

**Assume:** $p(x), q(x)$ are 64-bit for all $x \in \{0,1\}^n$ **Claim:** $\sum_{x \in \{0,1\}^n} p(x) \cdot q(x) = c$

**Round 1:** $s_1(1) = \sum_{x' \in \{0,1\}^{n-1}} p(1,x') \cdot q(1,x')$

small × small

$s_1(2) = \sum_{x' \in \{0,1\}^{n-1}} (2 \cdot p(1,x') - p(0,x')) \cdot (2 \cdot q(1,x') - q(0,x'))$

small × field

Can we keep using small ops in round $> 1$?

$p_1(x') = p(r$ $(1,x') - p(0,x'))$

$q_1(x') = q(r_1, x') = (1 - r_1) \cdot q(0,x') + r_1 \cdot q(1,x') = q(0,x') + r_1 \cdot (q(1,x') - q(0,x'))$

**Round 2:** $s_2(1) = \sum_{x' \in \{0,1\}^{n-1}} p_1(1,x') \cdot q_1(1,x')$

field × field

# Technique 2a: Round Batching

**Assume:** $p(x), q(x)$ are 64-bit for all $x \in \{0,1\}^n$ **Claim:** $\sum_{x \in \{0,1\}^n} p(x) \cdot q(x) = c$

**Key Idea:** compute enough materials to answer rounds $1,2,\ldots,k$ simultaneously

$$s(X_1, \ldots, X_k) = \sum p(X_1, \ldots, X_k, x') \cdot q(X_1, \ldots, X_k, x')$$

First $k$ rounds only have "small × small" and "small × field"

$$s(x^\star) = c$$

$x^\star \in \{0,1\}^k$

**Multivariate Interpolation:** suffices to know $s(u_1, \ldots, u_k)$ for $(u_1, \ldots, u_k) \in \{0,1,2\}^k$ (can omit $s(0,\ldots,0)$ )

**Example** $(k = 2)$**:**

small × small

$$s(1,2) = \sum_{x' \in \{0,1\}^{n-2}} p(1,2,x') \cdot q(1,2,x') = \sum_{x'} \big(2 \cdot p(1,1,x') - p(1,0,x')\big) \cdot \big(2 \cdot q(1,1,x') - q(1,0,x')\big)$$

# Summary of Improvements

**How can we improve on the linear-time algorithm?**

- Break down the black box of finite field arithmetic
  - Modify field operations to align with the sum-check proving algo

$\Longrightarrow 2\times$ **reduction in number of** $u64$ **mults**

- Capitalize on the properties of the polynomials arising in practice
  - E.g., zkVM program values are $64$ bits over a $\approx 256$ - bit field

$\Longrightarrow$ **trade** $\approx 4 \cdot 2^{n-1}$ **field mults for** $2 \cdot 3^k \cdot 2^{n-k}$ **"small** $\times$ **small" and** $2 \cdot 2^n$ **"small** $\times$ **field" mults**

**and** $2^k$ **factor reduction in space!**

# Technique 2b: Split Equality Polynomial

**Final blocker:** many applications of sum-check are to claims like

$$\sum_{x \in \{0,1\}^n} \widetilde{eq}(\tau, x) \cdot p(x) \cdot q(x) = c \,, \qquad \text{not just} \qquad \sum_{x \in \{0,1\}^n} p(x) \cdot q(x) = c$$

Apply linear-time proving algorithm for degree-3 $\implies$ $9 \cdot 2^n$ field mults

**Key idea:** utilize decomposability of $\widetilde{eq}$ to compute degree-2 polynomial in each round

$$s_i(X) = \widetilde{eq}(\tau_1, r) \cdot \widetilde{eq}(\tau_2, X) \cdot \sum_{x' \in \{0,1\}^{n-i}} \widetilde{eq}(\tau_3, x') \cdot p(r, X, x') \cdot q(r, X, x') \qquad \tau = (\tau_1, \tau_2, \tau_3)$$

suffices to compute this

$\implies$ $6 \cdot 2^n$ field mults instead

$\implies$ compatible with round batching (but with extra "small $\times$ field" mults)

*[Gruen24] Some Improvements for the PIOP for ZeroCheck + our refinements

# Talk Outline

1. The Sum-Check Protocol and Existing Algorithms

2. New Techniques for Sum-Check Proving

3. Evaluation & Summary

# Evaluation

**How do our optimizations transfer to practice:**

1. In a standalone setting

A degree-2 sum-check $\sum_x p(x) \cdot q(x) = 0$ (evaluations are random field elements)

2. In a deployed system such as Jolt

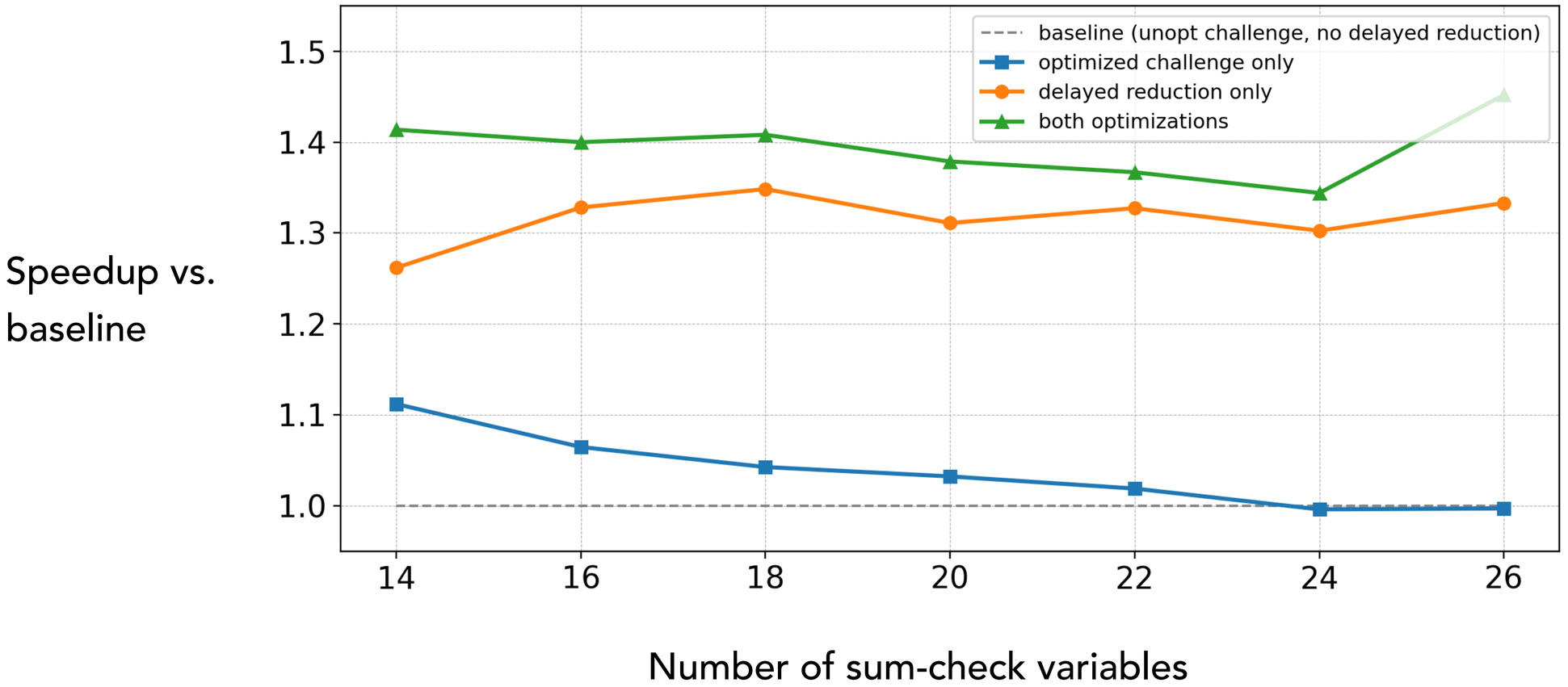
The Spartan sub-protocol $\sum_x \widetilde{eq}(\tau, x) \cdot \Big(a(x) \cdot b(x) - c(x)\Big) = 0$

($a, b, c$ are "small", up to $128$ - bit signed integers)

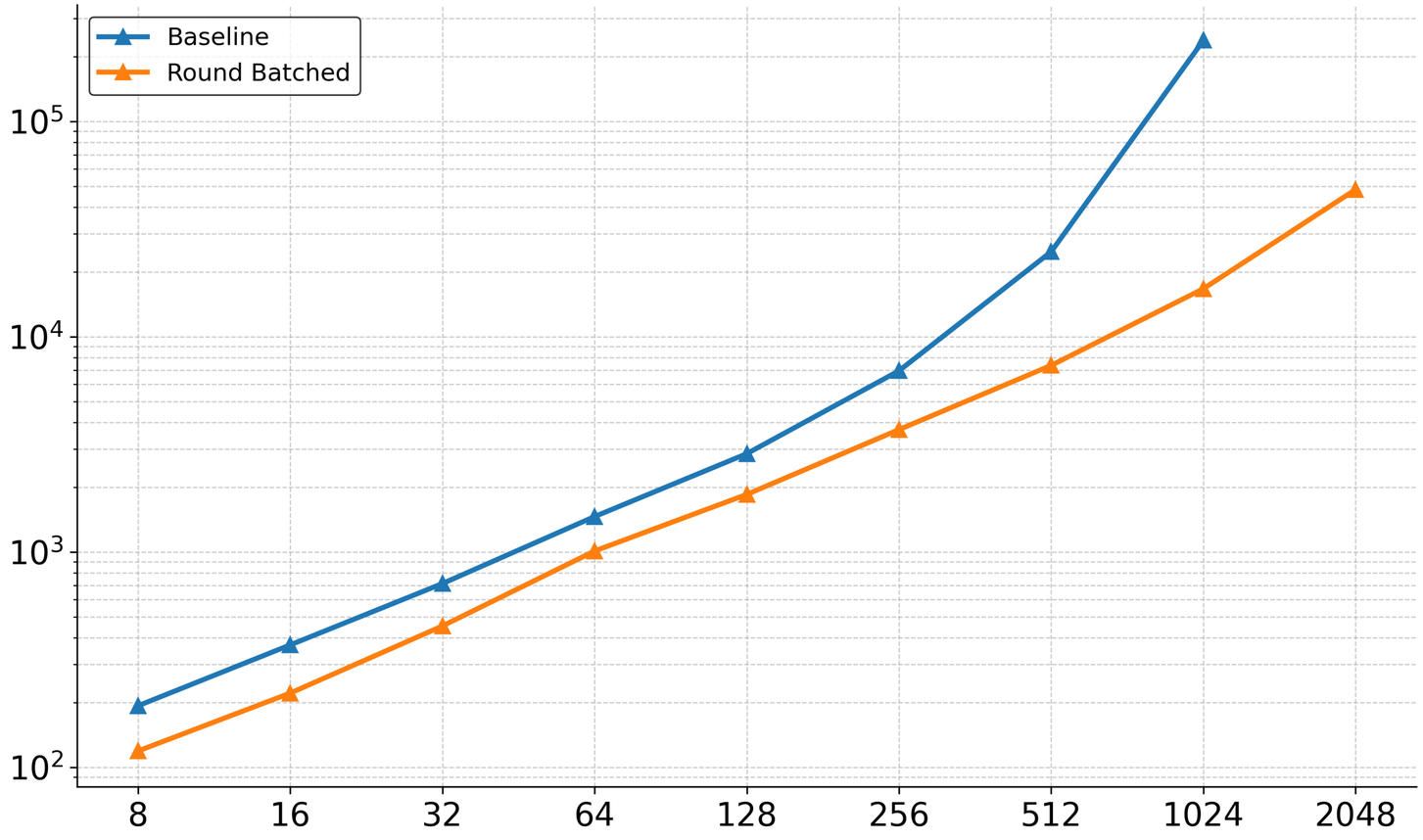
**Machine:** M4 Macbook Air with 10 cores & 24GB RAM

# Evaluation: Generic Degree-2 Sum-Check

**Single-threaded runs**



# Evaluation: Spartan Small-Value Sum-Check




**Multi-threaded runs**

**$\approx 1.5 - 2$ times speedup when baseline is not memory bound**

**$\approx 4$ times fewer memory usage**

# Summary

**Takeaways:**

- Sum-check is a major component of modern proof systems, especially zkVMs like Jolt
- Simple tweaks to the sum-check prover can give solid speedups
- Properly implementing proposed optimizations is hard!

**Future directions:**

- Further optimize core subroutine
- Better low-level implementation
- Better integration into Jolt

**ePrint 2025/1117**

## Thank you! Questions?