



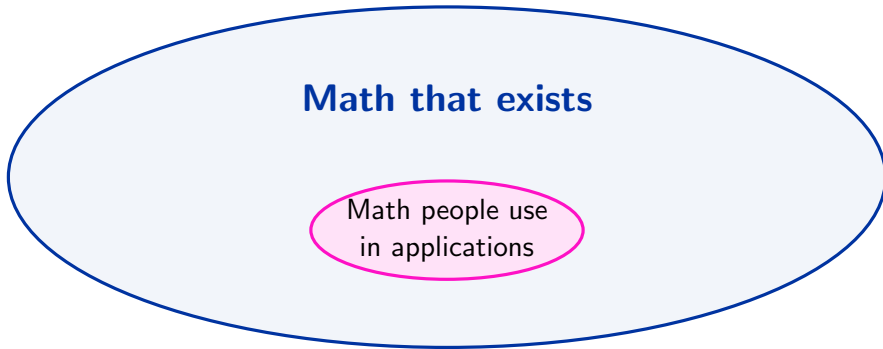
AG for SNARKs

i.e. why you should be much more ambitious about the math you can work with in this new AI-world

Rahul Dalal

Joint work with Benedikt Bünz, Tamir Hemo and Ron Rothblum

Currently



Why

Why can't we use more math techniques?

- **breadth**: broad space of techniques, don't know which one is useful



Why

Why can't we use more math techniques?

- **breadth**: broad space of techniques, don't know which one is useful
- **rabbit holes**: long time to learn each technique, may not get anywhere



Why

Why can't we use more math techniques?

- **breadth**: broad space of techniques, don't know which one is useful
- **rabbit holes**: long time to learn each technique, may not get anywhere
- **complexity**: lots of effort to work down to explicit computations



Why

Why can't we use more math techniques?

- **breadth**: broad space of techniques, don't know which one is useful
- **rabbit holes**: long time to learn each technique, may not get anywhere
- **complexity**: lots of effort to work down to explicit computations

Now: AI resolves issues?



Why

Why can't we use more math techniques?

- **breadth**: broad space of techniques, don't know which one is useful
- ~~rabbit holes~~: ~~long time to learn each technique, may not get anywhere~~
 - **Very cheap to test ideas**
- **complexity**: lots of effort to work down to explicit computations

Now: AI resolves issues?



Why

Why can't we use more math techniques?

- **breadth**: broad space of techniques, don't know which one is useful
- **rabbit holes**: long time to learn each technique, may not get anywhere
 - Very cheap to test ideas
- **complexity**: lots of effort to work down to explicit computations
 - Very easy to do mechanical computations

Now: AI resolves issues?



Why

Why can't we use more math techniques?

- **breadth**: broad space of techniques, don't know which one is useful
 - AI knows all the techniques, but not quite there yet in finding the right one
- **rabbit holes**: ~~long time to learn each technique, may not get anywhere~~
 - Very cheap to test ideas
- **complexity**: ~~lots of effort to work down to explicit computations~~
 - Very easy to do mechanical computations

Now: AI resolves issues? **with human idea input**



Case Study

Algebraic-Geometry codes to optimize Sumcheck round-skipping

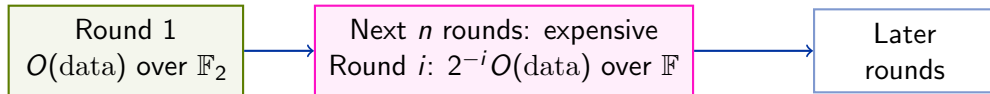
- 4th-5th idea using fancier codes to optimize parts of STARKS
- done with old models (opus 4.8, GPT 5.5 Pro)
 - current models should just be able to read the paper and reconstruct everything, even with different parameters



AG-skip: Binary Sumcheck Round-Skipping

Key Issue: The sumcheck protocol inputs data in $\mathbb{F}_2$ but works in $\mathbb{F} := \mathbb{F}_{2^{128}}$ to achieve soundness.

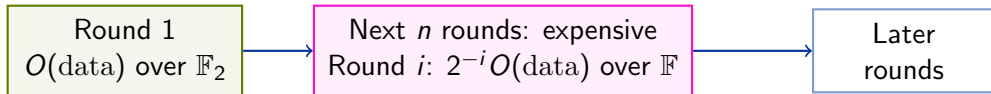
Standard sumcheck



AG-skip: Binary Sumcheck Round-Skipping

Key Issue: The sumcheck protocol inputs data in $\mathbb{F}_2$ but works in $\mathbb{F} := \mathbb{F}_{2^{128}}$ to achieve soundness.

Standard sumcheck

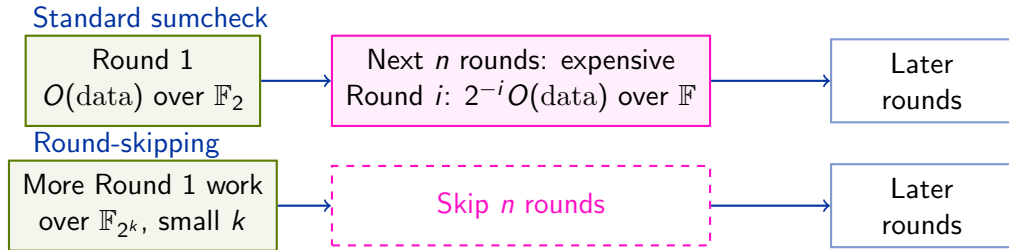


Round-skipping



AG-skip: Binary Sumcheck Round-Skipping

Key Issue: The sumcheck protocol inputs data in $\mathbb{F}_2$ but works in $\mathbb{F} := \mathbb{F}_{2^{128}}$ to achieve soundness.

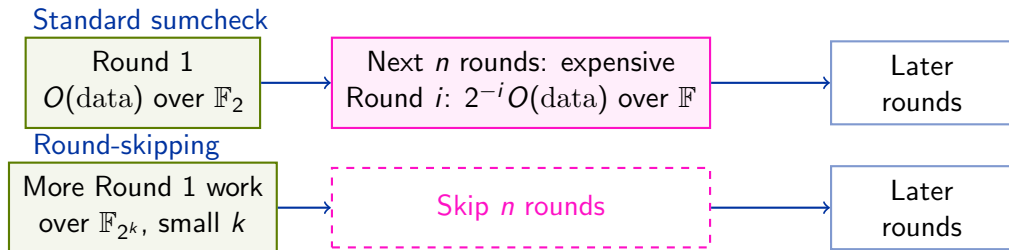


Trick: the **right kind** of error-correcting code over $\mathbb{F}$ with message length 2^n lets you skip n rounds. (Meir '13, RR24, RR25; tensor-code sumcheck)



AG-skip: Binary Sumcheck Round-Skipping

Key Issue: The sumcheck protocol inputs data in $\mathbb{F}_2$ but works in $\mathbb{F} := \mathbb{F}_{2^{128}}$ to achieve soundness.



Trick: the right kind of error-correcting code over $\mathbb{F}$ with message length 2^n lets you skip n rounds. (Meir '13, RR24, RR25; tensor-code sumcheck)

Standard choice: Reed–Solomon. (Gruen's trick, univariate skip)



AG-skip: Code Properties

Given C , there exists a **product code** C^* .



AG-skip: Code Properties

Given C , there exists a **product code** C^* .

Property	Condition	Purpose
Multiplicativity	$\exists v \times w : C(v) \circ C(w) = C^*(v \times w).$ $v \times w$ has prefix $v \circ w$.	For quadratic sumchecks

$\circ$ denotes the pointwise (Hadamard) product.



AG-skip: Code Properties

Given C , there exists a **product code** C^* .

Property	Condition	Purpose
Multiplicativity	$\exists v \times w : C(v) \circ C(w) = C^*(v \times w).$ $v \times w$ has prefix $v \circ w$.	For quadratic sumchecks
Distance	C^* has very large minimum distance.	For 1st-round challenge soundness

$\circ$ denotes the pointwise (Hadamard) product.



AG-skip: Code Properties

Given C , there exists a **product code** C^* .

Property	Condition	Purpose
Multiplicativity	$\exists v \times w : C(v) \circ C(w) = C^*(v \times w).$ $v \times w$ has prefix $v \circ w$.	For quadratic sumchecks
Distance	C^* has very large minimum distance.	For 1st-round challenge soundness
Small product field	v, w over $\mathbb{F}_2 \implies v \times w$ over $\mathbb{F}_{2^k}.$ Small k : Reed–Solomon n .	1st-round work over $\mathbb{F}_{2^k}$

$\circ$ denotes the pointwise (Hadamard) product.



AG-skip: Code Properties

Given C , there exists a **product code** C^* .

Property	Condition	Purpose
Multiplicativity	$\exists v \times w : C(v) \circ C(w) = C^*(v \times w).$ $v \times w$ has prefix $v \circ w$.	For quadratic sumchecks
Distance	C^* has very large minimum distance.	For 1st-round challenge soundness
Small product field	v, w over $\mathbb{F}_2 \implies v \times w$ over $\mathbb{F}_{2^k}.$ Small k : Reed–Solomon n .	1st-round work over $\mathbb{F}_{2^k}$

To Optimize: minimize $k (\dim C^* - \dim C)$, and the cost of computing $v \times w$.

$\circ$ denotes the pointwise (Hadamard) product.



AG-skip: Algebraic Geometry Input

Construction: Differential Goppa (AG) codes give $k = 1$



AG-skip: Algebraic Geometry Input

Construction: Differential Goppa (AG) codes give $k = 1$

- Choose: “1d” system X of polynomials in d -variables w/ $\geq 2^n$ solutions in $\mathbb{F}_2$.



AG-skip: Algebraic Geometry Input

Construction: Differential Goppa (AG) codes give $k = 1$

- Choose: “1d” system X of polynomials in d -variables w/ $\geq 2^n$ solutions in $\mathbb{F}_2$.

Algebraic geometry input: $X \mapsto$ vector spaces of rational functions L_1, L_2 on solutions to X satisfying



AG-skip: Algebraic Geometry Input

Construction: Differential Goppa (AG) codes give $k = 1$

- Choose: “1d” system X of polynomials in d -variables w/ $\geq 2^n$ solutions in $\mathbb{F}_2$.

Algebraic geometry input: $X \mapsto$ vector spaces of rational functions L_1, L_2 on solutions to X satisfying

- If $f, g \in L_1$, then $fg \in L_2$.



AG-skip: Algebraic Geometry Input

Construction: Differential Goppa (AG) codes give $k = 1$

- Choose: “1d” system X of polynomials in d -variables w/ $\geq 2^n$ solutions in $\mathbb{F}_2$.

Algebraic geometry input: $X \mapsto$ vector spaces of rational functions L_1, L_2 on solutions to X satisfying

- If $f, g \in L_1$, then $fg \in L_2$.
- $f \in L_1$ can be interpolated from evaluations of f at 2^n $\mathbb{F}_2$ -solutions to X



AG-skip: Algebraic Geometry Input

Construction: Differential Goppa (AG) codes give $k = 1$

- Choose: “1d” system X of polynomials in d -variables w/ $\geq 2^n$ solutions in $\mathbb{F}_2$.

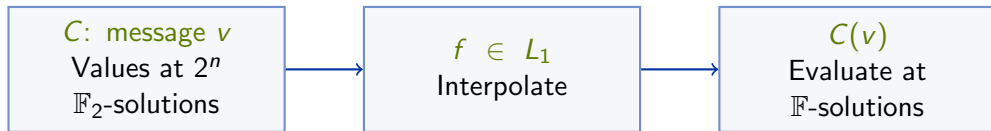
Algebraic geometry input: $X \mapsto$ vector spaces of rational functions L_1, L_2 on solutions to X satisfying

- If $f, g \in L_1$, then $fg \in L_2$.
- $f \in L_1$ can be interpolated from evaluations of f at 2^n $\mathbb{F}_2$ -solutions to X
- $h \in L_2$ can be interpolated from evaluations of h at 2^n $\mathbb{F}_2$ -solutions to X and enough derivatives of h at those points



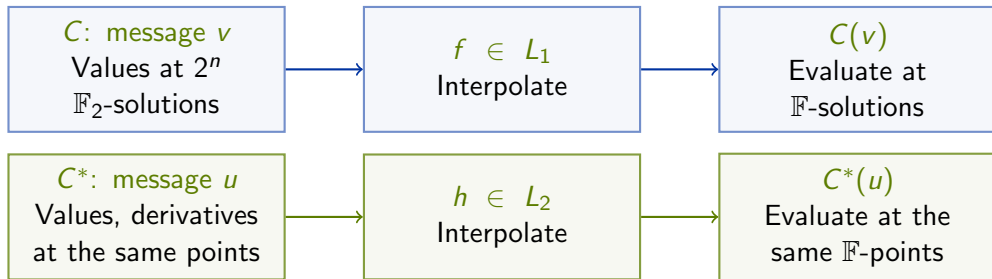
AG-skip: Abstract Code Construction

Deriving C , C^* and the product rule from X, L_1, L_2 :



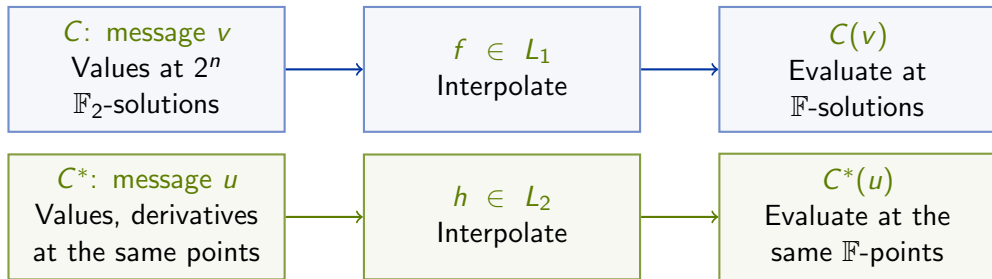
AG-skip: Abstract Code Construction

Deriving C , C^* and the product rule from X, L_1, L_2 :



AG-skip: Abstract Code Construction

Deriving C , C^* and the product rule from X, L_1, L_2 :

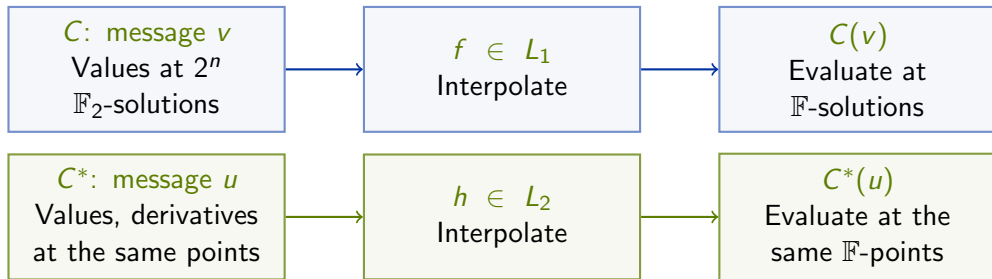


$v \times w$: interpolate, evaluate derivatives, and multiply using the product rule.



AG-skip: Abstract Code Construction

Deriving C , C^* and the product rule from X, L_1, L_2 :



$v \times w$: interpolate, evaluate derivatives, and multiply using the product rule.

Goal for explicit code: optimize $\dim C^*$.



AG-skip: Explicit Construction by AI

Focus: skip 6 rounds, $2^n = 64$



AG-skip: Explicit Construction by AI

Focus: skip 6 rounds, $2^n = 64$

- Algebraic Geometry $\implies$ $\dim C^*$ determined by genus of X



AG-skip: Explicit Construction by AI

Focus: skip 6 rounds, $2^n = 64$

- Algebraic Geometry $\implies$ $\dim C^*$ determined by genus of X
- $\implies$ Goal: find lowest genus X with 64 solutions over $\mathbb{F}_2$, explicitly compute L_1, L_2 and interpolation maps



AG-skip: Explicit Construction by AI

Focus: skip 6 rounds, $2^n = 64$

- Algebraic Geometry $\implies$ $\dim C^*$ determined by genus of X
- $\implies$ Goal: find lowest genus X with 64 solutions over $\mathbb{F}_2$, explicitly compute L_1, L_2 and interpolation maps
- GPT 5.5 Pro/Claude Opus 4.8 did in a weekend
 - vs months-long computation requiring experts in explicit curve construction pre-LLM



AG-skip: Explicit Construction by AI

Focus: skip 6 rounds, $2^n = 64$

- Algebraic Geometry $\implies$ $\dim C^*$ determined by genus of X
- $\implies$ Goal: find lowest genus X with 64 solutions over $\mathbb{F}_2$, explicitly compute L_1, L_2 and interpolation maps
- GPT 5.5 Pro/Claude Opus 4.8 did in a weekend
 - vs months-long computation requiring experts in explicit curve construction pre-LLM
- Key Point: explicit matrices/polynomials $\implies$ properties of C, C^* can be verified by script+elementary math, completely forgetting AG source: Easy verification!



AG-skip: Results

How well does the construction work?



AG-skip: Results

How well does the construction work?

Optimization Goal: $k (\dim C^* - \dim C)$

Reed–Solomon  $384 = 6 \cdot 64$

AG codes  158



AG-skip: Results

How well does the construction work?

Optimization Goal: $k (\dim C^* - \dim C)$

Reed–Solomon  $384 = 6 \cdot 64$

AG codes  158

$\mathbb{F}_2$ arithmetic is much more hardware friendly than $\mathbb{F}_{64}$ arithmetic (especially on GPUs).



AG-skip: Results

How well does the construction work?

Optimization Goal: $k (\dim C^* - \dim C)$

Reed–Solomon  $384 = 6 \cdot 64$

AG codes  158

$\mathbb{F}_2$ arithmetic is much more hardware friendly than $\mathbb{F}_{64}$ arithmetic (especially on GPUs).

$\Rightarrow \approx 2\times$ speed-up to the zerocheck part of the FLOCK protocol.



Further Directions

Other relevant fancy math?



Further Directions

Other relevant fancy math?

- AG-codes with fast encoding
 - hermitian curve AG-codes can have $n \log n$ encoding
 - payoff: still staying in smaller fields



Further Directions

Other relevant fancy math?

- AG-codes with fast encoding
 - hermitian curve AG-codes can have $n \log n$ encoding
 - payoff: still staying in smaller fields
- Rate Mechanisms for Tanner codes
 - Tanner codes with very low local rate can have positive global rate
 - codewords come from global algebraic structure, cohomology classes?? automorphic forms??.



Further Directions

Other relevant fancy math?

- AG-codes with fast encoding
 - hermitian curve AG-codes can have $n \log n$ encoding
 - payoff: still staying in smaller fields
- Rate Mechanisms for Tanner codes
 - Tanner codes with very low local rate can have positive global rate
 - codewords come from global algebraic structure, cohomology classes?? automorphic forms??.
- Expander complex constructions
 - Give codes with very useful properties (local testability, etc) but with impractical constants so far
 - Automorphic theory: HD Ramanujan complex with “bipartite-like” structure
 - OpenAI: coboundary expanders in every dimension

