Computational Mathematics and AI

Lecture 10: Mathematical Discovery and Verification

Lars Ruthotto
Departments of Mathematics and Computer Science
lruthotto@emory.edu
larsruthotto

slido.com #CBMS25

Reading List

Historical Context: AI-assisted mathematical discovery traces from automated theorem proving in the 1950s to modern neural-guided search, now discovering *new* mathematics.

Key Readings:

1. Fawzi et al. (2022) – Discovering faster matrix multiplication algorithms. *Nature*

   First AI to discover algorithms beating 50-year human records.

2. Novikov, Fawzi, et al. (2025) – AlphaEvolve: A coding agent for scientific discovery. *DeepMind*

   LLM-based evolution framework; complementary approach to AlphaTensor.

3. Moura and Ullrich (2021) – The Lean 4 Theorem Prover. *CADE*

   Modern proof assistant with Mathlib (210K+ theorems).

4. Song et al. (2024) – Lean Copilot: LLMs for Theorem Proving. *NeurIPS*

   AI-assisted formal verification; 74% proof step automation.

Lecture Outline: Why Math? $\rightarrow$ Matrix Multiplication $\rightarrow$ AlphaTensor $\rightarrow$ AlphaEvolve $\rightarrow$ Proof Assistants $\rightarrow$ Synthesis

Introduction

AI Breaks 50-Year-Old Records

The Matrix Multiplication Story

Year	Event	Multiplications
Standard	Naive algorithm for 4×4	64
1969	Strassen discovers 7-multiply for 2×2	$\to O(n^{2.81})$
1969-2022	No improvement for 4×4	49 (Strassen^2)
2022	AlphaTensor (RL on tensors)	**47** ($\mathbb{Z}_2$), 48 (standard)
2024	AlphaEvolve (LLM evolution)	**48** (complex)

Two Complementary Approaches

- **AlphaTensor**: Exploits tensor structure via RL game
- **AlphaEvolve**: Flexible code evolution via LLMs

Neither is "better"—different tools for different contexts

AI broke 50-year records using complementary strategies

What Makes AI Promising for Mathematics?

> "Intelligence means having a goal." — Richard Sutton

Mathematics Provides Clear, Verifiable Goals

- **Matrix multiplication**: Count scalar multiplications, verify correctness numerically/symbolically
- **Theorem proving**: Proof type-checks $\Rightarrow$ guaranteed correct
- **Algorithm discovery**: Objective function is unambiguous

Contrast with Open-Ended Natural Language Tasks

- What is funny? Humor is culturally dependent
- Creative writing: What is "good"?
- Art generation: Success is subjective

clear objectives enable AI breakthroughs in mathematics

Two Modes of AI in Mathematics

Discovery (Inductive)

- Search vast spaces
- Propose novel algorithms/constructions
- Methods: RL, code evolution, heuristic search
- Systems: AlphaTensor, AlphaEvolve

Fast but uncertain

Verification (Deductive)

- Prove correctness
- Formalize mathematics
- Methods: Proof assistants, formal methods
- Systems: Lean 4, LeanCopilot

Slow but certain

Problem-Specific Choice: Some problems need discovery, some need proofs

Different examples for each mode are natural—they don't always meet

discovery and verification are complementary tools

Matrix Multiplication

Why Matrix Multiplication Matters

Matrix Multiplication is Everywhere

- Neural network forward/backward passes
- Graphics and game engines
- Scientific simulations
- Cryptography

The Complexity Question

- Naive algorithm: n^3 multiplications for $n \times n$ matrices
- **Can we do better?** How many multiplies are *actually* necessary?

Why 4×4 Specifically

- Building block for larger matrices (recursive algorithms)
- Hardware-relevant size (SIMD, tensor cores)
- Small enough to search, large enough to matter

small improvements in matrix multiplication compound at scale

Algorithm Discovery as Discrete Optimization

Problem Formulation

Let $\mathcal{A}$ be the space of all algorithms for a task. Find:

$$a^* = \arg\min_{a \in \mathcal{A}} L(a)$$

where $L : \mathcal{A} \to \mathbb{R}$ measures cost (e.g., operation count, runtime)

Key Design Choice: How do we *represent* candidate algorithms?

- As **tensors**: AlphaTensor (exploits mathematical structure)
- As **code**: AlphaEvolve (flexible, domain-agnostic)

Why Is This Hard?

- $\mathcal{A}$ is **discrete, combinatorial, and huge**
- No gradient information available
- Random search is hopeless

Classical Local Search: $a_{k+1} \in \mathcal{N}(a_k)$ with random neighbors

Problem: Random perturbations rarely produce valid/improved algorithms

The Tensor Decomposition View

Mathematical Formulation

Matrix multiplication $(A, B) \mapsto C = AB$ defines a **trilinear form**:

$$C_{ij} = \sum_k A_{ik} B_{kj}$$

This encodes into a 3-way tensor $\mathcal{T} \in \mathbb{R}^{n^2 \times n^2 \times n^2}$ with decomposition:

$$\mathcal{T} = \sum_{r=1}^{R} \mathbf{u}_r \otimes \mathbf{v}_r \otimes \mathbf{w}_r$$

Key Correspondence Kolda and Bader 2009

- Each rank-1 term $\mathbf{u}_r \otimes \mathbf{v}_r \otimes \mathbf{w}_r \leftrightarrow$ one scalar multiplication
- **Tensor rank** R: Minimum number of rank-1 terms needed
- **Finding minimal R = finding most efficient algorithm**

The Search Problem

Given multiplication tensor $\mathcal{T}$, find vectors $\{\mathbf{u}_r, \mathbf{v}_r, \mathbf{w}_r\}_{r=1}^{R}$ minimizing R

tensor rank = number of scalar multiplications

2×2 Example: Naive Algorithm (Rank 8)

The Naive Decomposition

For $C = AB$ with 2×2 matrices, naive algorithm uses 8 multiplications:

$$C_{ij} = \sum_{k=1}^{2} A_{ik} B_{kj}$$

Explicit Rank-1 Terms (vectorizing A, B, C as length-4 vectors)

r	$\mathbf{u}_r$ (selects from A)	$\mathbf{v}_r$ (selects from B)	$\mathbf{w}_r$ (contributes to C)	Computes
1	$(1,0,0,0)$	$(1,0,0,0)$	$(1,0,0,0)$	$a_{11}b_{11} \to c_{11}$
2	$(0,1,0,0)$	$(0,0,1,0)$	$(1,0,0,0)$	$a_{12}b_{21} \to c_{11}$
3	$(1,0,0,0)$	$(0,1,0,0)$	$(0,1,0,0)$	$a_{11}b_{12} \to c_{12}$
4	$(0,1,0,0)$	$(0,0,0,1)$	$(0,1,0,0)$	$a_{12}b_{22} \to c_{12}$
5	$(0,0,1,0)$	$(1,0,0,0)$	$(0,0,1,0)$	$a_{21}b_{11} \to c_{21}$
6	$(0,0,0,1)$	$(0,0,1,0)$	$(0,0,1,0)$	$a_{22}b_{21} \to c_{21}$
7	$(0,0,1,0)$	$(0,1,0,0)$	$(0,0,0,1)$	$a_{21}b_{12} \to c_{22}$
8	$(0,0,0,1)$	$(0,0,0,1)$	$(0,0,0,1)$	$a_{22}b_{22} \to c_{22}$

Each row = one scalar multiplication. Total: $R = 8$

2×2 Example: Strassen's Algorithm (Rank 7)

Strassen's Insight (1969): Trade multiplications for additions
Define 7 intermediate products $M_1, \ldots, M_7$:

$$M_1 = (a_{11} + a_{22})(b_{11} + b_{22})$$
$$M_2 = (a_{21} + a_{22})b_{11}$$
$$M_3 = a_{11}(b_{12} - b_{22})$$
$$M_4 = a_{22}(b_{21} - b_{11})$$
$$M_5 = (a_{11} + a_{12})b_{22}$$
$$M_6 = (a_{21} - a_{11})(b_{11} + b_{12})$$
$$M_7 = (a_{12} - a_{22})(b_{21} + b_{22})$$

Reconstruction

$$c_{11} = M_1 + M_4 - M_5 + M_7$$
$$c_{12} = M_3 + M_5$$
$$c_{21} = M_2 + M_4$$
$$c_{22} = M_1 - M_2 + M_3 + M_6$$

Result: 7 multiplications, 18 additions (vs. 8 mult, 4 add for naive)

Strassen proved rank ≥ 7, so this is optimal for 2×2

Strassen as Tensor Decomposition

The 7 Rank-1 Terms (non-zero entries only, using ± 1 coefficients)

r	$\mathbf{u}_r$ (selects from A)	$\mathbf{v}_r$ (selects from B)	$\mathbf{w}_r$ (contributes to C)	Computes
M_1	$(1,0,0,1)$	$(1,0,0,1)$	$(1,0,0,1)$	$(a_{11}+a_{22})(b_{11}+b_{22})$
M_2	$(0,0,1,1)$	$(1,0,0,0)$	$(0,0,1,-1)$	$(a_{21}+a_{22})b_{11}$
M_3	$(1,0,0,0)$	$(0,1,0,-1)$	$(0,1,0,1)$	$a_{11}(b_{12}-b_{22})$
M_4	$(0,0,0,1)$	$(-1,0,1,0)$	$(1,0,1,0)$	$a_{22}(b_{21}-b_{11})$
M_5	$(1,1,0,0)$	$(0,0,0,1)$	$(-1,1,0,0)$	$(a_{11}+a_{12})b_{22}$
M_6	$(-1,0,1,0)$	$(1,1,0,0)$	$(0,0,0,1)$	$(a_{21}-a_{11})(b_{11}+b_{12})$
M_7	$(0,1,0,-1)$	$(0,0,1,1)$	$(1,0,0,0)$	$(a_{12}-a_{22})(b_{21}+b_{22})$

Validity Constraint: For any decomposition $\mathcal{T} = \sum_r \mathbf{u}_r \otimes \mathbf{v}_r \otimes \mathbf{w}_r$

$$\text{Valid} \iff \forall A, B: \quad \sum_r (\mathbf{u}_r^\top \text{vec}(A))(\mathbf{v}_r^\top \text{vec}(B))\mathbf{w}_r = \text{vec}(AB)$$

Tensor $\mathcal{T}$ encodes the multiplication table: decomposition must reconstruct it exactly

finding better algorithms = finding lower-rank decompositions

Historical Breakthroughs and the 4×4 Gap

The Strassen Revolution (1969) Strassen 1969

- 2×2: Rank 7 (optimal, proven)
- Recursive application: $O(n^{\log_2 7}) \approx O(n^{2.807})$
- Coppersmith-Winograd (1990): $O(n^{2.376})$ theoretical bound

The 4×4 Stagnation

Method	Multiplications	Notes
Naive	64	4^3
Strassen^2 (recursive)	49	7^2
Best known (pre-2022)	49	No improvement!
Theoretical minimum	≥ 19	Lower bound

50+ years with no practical improvement for 4×4!

next: how to discover multiplication algorithms with AI

AlphaTensor: RL on Tensor Structure

AlphaTensor: Tensor Decomposition as a Game

AlphaTensor's Key Innovation Fawzi et al. 2022
Reframe tensor decomposition as a **single-player game**:

Why Not Standard Tensor Decomposition?

- We want factors with **small integer entries** $\{-2, -1, 0, 1, 2\}$ (exact arithmetic)
- Standard methods (ALS, gradient descent) find real-valued decompositions
- Integer constraint $\Rightarrow$ discrete combinatorial search $\Rightarrow$ RL

TensorGame

- **State**: Residual tensor S_t (initialized: $S_0 = \mathcal{T}$, the multiplication tensor)
- **Action**: $S_{t+1} = S_t - \mathbf{u} \otimes \mathbf{v} \otimes \mathbf{w}$ where $\mathbf{u}, \mathbf{v}, \mathbf{w}$ have small integer entries
- **Goal**: $S_R = 0$ (complete decomposition)
- **Reward**: $-R$ (minimize number of rank-1 terms)

RL Architecture (AlphaZero-style)

$$\text{State (tensor)} \rightarrow \text{Transformer} \rightarrow \text{Value + Policy (next action)}$$

Training

- Self-play from scratch (no human algorithm examples)
- Exploits symmetries of multiplication tensor

AlphaTensor Results

Breakthrough Achievement (2022) Fawzi et al. 2022

Matrix Size	Previous Best	AlphaTensor	Arithmetic
4×4	49	**48**	Standard ($\mathbb{R}$)
4×4	49	**47**	Modular ($\mathbb{Z}_2$)
5×5	98	**96**	Modular ($\mathbb{Z}_2$)

What is Modular Arithmetic ($\mathbb{Z}_2$)?

- Arithmetic modulo 2: only values 0 and 1, where $1 + 1 = 0$
- Used in cryptography, coding theory, computer science
- Simpler structure $\Rightarrow$ sometimes allows lower rank
- AlphaTensor's 47-multiplication algorithm works *only* in $\mathbb{Z}_2$

Scale of Discovery

Discovered **14,000+ distinct algorithms** for 4×4 alone!

exploiting tensor structure via RL breaks long-standing records

AlphaEvolve: LLM Code Evolution

LLMs as Informed Proposal Distributions

Analogy to MCMC

In Markov Chain Monte Carlo, better proposal $q(x'|x) \Rightarrow$ faster convergence

For Algorithm Search

Define a **proposal distribution** $q(a'|a)$ over algorithm modifications:

- **Uniform random**: $q(a'|a) \propto \mathbf{1}_{a' \in \mathcal{N}(a)}$ (inefficient)
- **Informed**: $q(a'|a)$ concentrates on "sensible" modifications

The LLM as Proposal Distribution

Large language models define an *informed* proposal:

$$q_{\text{LLM}}(a'|a) = P(\text{modified code } a' \mid \text{original code } a)$$

- Trained on billions of lines of code $\Rightarrow$ learns valid transformations
- Proposes *structurally valid* algorithms (not random bit flips)
- Sample complexity: $\sim$150 evaluations vs. millions

Hope: LLMs provide intelligent navigation of algorithm space

The Iterative Search Algorithm

Algorithm (LLM-Guided Evolutionary Search) Novikov, Fawzi, et al. 2025

Input: Initial algorithm a_0, objective L, archive cells $\{C_j\}$
Initialize: Archive $\mathcal{B} \leftarrow \{a_0\}$
For $k = 1, \ldots, K$:

1. **Select**: Sample a from archive $\mathcal{B}$ (quality-weighted)
2. **Propose**: Generate $a' \sim q_{\text{LLM}}(\cdot|a)$ via language model
3. **Evaluate**: Compute $L(a')$ and verify correctness
4. **Update**: If $L(a') < L(\mathcal{B}[j])$, set $\mathcal{B}[j] \leftarrow a'$

Return: Archive $\mathcal{B}$ of best algorithms

Key Properties

- Typically $K \sim 150$ iterations (not millions!)
- Discovers *interpretable* algorithms (executable code)
- Maintains diversity across trade-off space

AlphaEvolve Results

Matrix Multiplication Achievement Novikov, Fawzi, et al. 2025

- 4×4 **complex-valued** matrices: **48 scalar multiplications**
- First improvement over Strassen (49) for complex matrices

Broader Discoveries

Problem Class	Match Known	Exceed Known
Combinatorial optimization	75%	20%
Discrete mathematics	80%	15%
Algorithm design	70%	25%

AlphaTensor vs. AlphaEvolve: Complementary Approaches

	AlphaTensor	AlphaEvolve
Representation	Tensor decomposition	Executable code
Search method	RL (AlphaZero)	LLM evolution
Structure use	Exploits tensor structure	Domain-agnostic
Best 4×4	47 ($\mathbb{Z}_2$), 48 ($\mathbb{R}$)	48 ($\mathbb{C}$)

Neither is “better”—different strengths for different contexts

The Discovery-Verification Gap

What Discovery Systems Provide

- ✓ Candidate algorithms
- ✓ Numerical verification on test cases
- ✓ Efficient implementations
- ✓ Clear objective: multiplication count

What's Still Missing

- × Formal proof of correctness for *all* inputs
- × Rigorous complexity analysis
- × Numerical stability guarantees

The Goal Advantage Again

Matrix multiplication has a **clear verification criterion**: numerical correctness
But for mathematical theorems, we want **absolute certainty**...

next: proof assistants provide the "ultimate truth"

Proof Assistants

What Are Proof Assistants?

Definition: Interactive theorem provers that convert informal mathematics into machine-checkable formal proofs

Key Property: **Absolute certainty** — if the proof type-checks, it's correct

The Goal is Crystal Clear

Sutton's Principle Applied

Proof type-checks = correct. No ambiguity. Ultimate truth.

The Lean 4 Ecosystem Moura and Ullrich 2021

- Modern proof assistant (Microsoft Research $\rightarrow$ open source)
- **Mathlib**: 210,000+ theorems, 100,000+ definitions
- Active community including Terence Tao

The Formalization Process

Informal theorem $\rightarrow$ Formal statement $\rightarrow$ Proof tactics $\rightarrow$ Verified theorem

The Formalization Challenge

What Makes Formalization Hard

1. **Expertise barrier**: Learning Lean syntax and tactics
2. **Library knowledge**: Which of 210,000 theorems are relevant?
3. **Granularity gap**: Informal "obvious" $\rightarrow$ formal 50-step proof
4. **Time investment**: 10-100$\times$ longer than informal proof

The Verification Bottleneck

- Discovery systems produce candidates faster than we can verify
- Human formalization effort is the limiting factor
- AI assistance is essential for scaling

verification is the bottleneck, not discovery

Example: Gershgorin Circle Theorem

The Formalization Gap: Same theorem, two representations

LaTeX (Informal)

"Every eigenvalue λ of A lies in some Gershgorin disc:"

$$D_i = \{z : |z - a_{ii}| \leq r_i\}$$

where $r_i = \sum_{j \neq i} |a_{ij}|$

Human-readable, ambiguous types

Lean 4 (Formal)

```
def radius (A : Matrix n n C)
    (i : n) : R :=
  sum j, |A i j|  -- j != i

theorem gershgorin
    (h : e in spectrum C A) :
    exists i, e in disc A i := by
  sorry  -- proof needed!
```

Machine-checkable, explicit types

The Gap: Informal "obvious" $\rightarrow$ formal 50+ tactics

AI assistance bridges the informal-formal gap

LeanCopilot: AI-Assisted Proving

LeanCopilot (2024) Song et al. 2024

Three Core Capabilities

1. Tactic Suggestion

- Input: Current proof state + goal
- Output: Next tactic to apply (like GitHub Copilot for proofs)

2. Proof Search

- Automated multi-step proof completion
- Combines neural guidance with symbolic search

3. Premise Selection

- Which theorems from Mathlib are relevant?
- LLM learns mathematical relevance patterns

Empirical Results

- **74.2%** of proof steps automated (vs. 40.1% with aesop alone)
- Average **2.08** manual steps needed (vs. 3.86 without AI)

Human-AI Collaboration Workflow

The Collaborative Pattern

Human: Strategy → AI: Fill steps → Human: Verify → Lean: Check

Where AI Helps Most

- Boilerplate tactics
- Library search (premise selection)
- Completing routine sub-goals
- Interactive feedback loop

Where Humans Remain Essential

- Overall proof strategy
- Creative problem decomposition
- Handling edge cases
- Mathematical insight

Notable Achievements

- **AlphaProof (2024)**: Silver medal at International Mathematical Olympiad
- **Terence Tao**: Formalized Polynomial Freiman-Ruzsa (PFR) conjecture in Lean

Summary

Two Modes of AI in Mathematics: Summary

Discovery and Verification are Complementary Tools

	Discovery	Verification
Goal	Find new algorithms/objects	Prove correctness
Methods	RL, LLM evolution, search	Proof assistants, tactics
Examples	AlphaTensor, AlphaEvolve	Lean 4, LeanCopilot
Strength	Fast exploration	Absolute certainty
Weakness	No formal guarantees	Slow, labor-intensive

Problem-Specific Choice

- Matrix multiplication $\rightarrow$ Discovery (clear numerical objective)
- Formalizing theorems $\rightarrow$ Verification (need certainty)
- Some problems need both, some need only one

The Common Thread: Both succeed because math provides **clear goals**

choose the right tool for each mathematical problem

Σ: Five Key Takeaways

1. **Clear Goals Enable AI Success**
   - Mathematics provides unambiguous objectives (Sutton's principle)
2. **Complementary Discovery Methods**
   - AlphaTensor (tensor structure + RL) and AlphaEvolve (code + LLM) both work
   - Neither is universally better—different tools for different contexts
3. **Verification Provides Certainty**
   - Proof assistants + AI: absolute correctness guarantees
4. **Discovery and Verification are Problem-Specific**
   - Some problems need discovery, some need proofs, some need both
5. **AI Augments Mathematicians**
   - Human insight + AI automation = powerful combination

Research Opportunities

For Computational Mathematicians

- Learn basics of Lean (accessible formal methods)
- Experiment with evolution frameworks for your domain
- Collaborate with AI on formalization projects

For AI Researchers

- Explore mathematical domains beyond images/text
- Integrate symbolic reasoning with neural methods
- Contribute to formal-informal translation

Open Research Directions

1. Automated discovery $\rightarrow$ verification pipelines
2. Proof-guided evolution systems
3. Verified neural solvers for PDEs
4. Mathematical intuition extraction from AI discoveries

Course Structure: 10 Lectures, 3 Modules

Module 1: Crash Course

L1: ML Overview
- Learning tasks
- Double descent

L2: Learning Problems
- MLPs, GNNs, Transformers
- ResNets, Neural ODEs
- Loss functions

L3: Optimization
- Empirical vs. expected risk

Module 2: CM $\rightarrow$ AI

L4: Stochastic Optimization
- Convergence
- Implicit regularization

L5: Loss Landscapes
- Adaptive methods
- Modern optimization

L6: Generative Modeling
- PDEs, optimal transport
- Diffusion, flow matching

Module 3: CM $\leftarrow$ AI

L7: Scientific ML
- PINNs, neural operators
- learned solvers

L8: High-Dim PDEs
- Curse of dimensionality
- Deep BSDE, FBSDE, HJB

L9: Inverse Problems
- Simulation based inference
- Diffusion priors

L10: Math Discovery
- Evolutionary coding
- Proof assistants

Course Philosophy and Expectations

What this course IS:

- Illustrative: Representative examples from different topics
- Bidirectional: CompMath $\leftrightarrow$ AI synergy
- Hands-on: Numerical experiments and computational demos
- Research-oriented: Active frontiers, open problems

What this course is NOT:

- Comprehensive: 10 lectures cannot cover everything
- Pure theory: Balance rigor with intuition
- Software engineering: Concepts over production code
- Latest & greatest: Field evolves faster than curricula

Our approach:

- Pick characteristic issues from each research direction
- Guide you into the field, not exhaustive coverage
- Complement with workshop research talks
- Equip you to read papers and start your own projects

goal: Mathematical foundations + computational tools for CM+AI research

Closing Reflection

Where We Started (Lecture 1)

- Machine learning as function approximation
- Neural networks as computational tools
- Optimization algorithms for training

Where We Ended (Lecture 10)

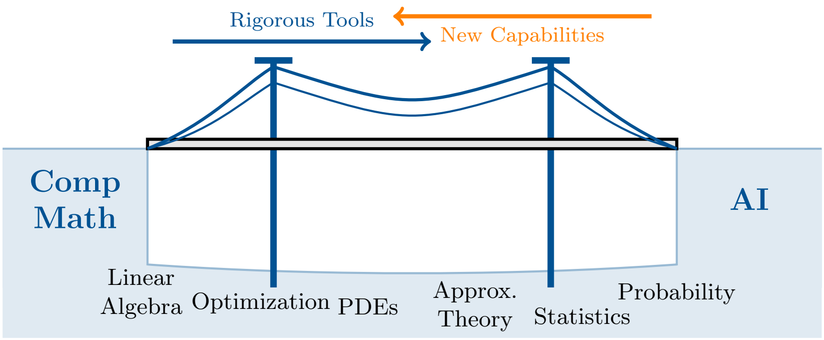
- AI discovering mathematical algorithms
- AI assisting formal theorem proving
- Computational mathematics and AI fully intertwined

The Meta-Lesson

The bidirectional relationship between computational mathematics and AI is not just pedagogical—it's the frontier of both fields. Progress in one advances the other.

The Complete Course Journey

Three Modules of Computational Mathematics and AI



Module	Lectures	Theme
ML Crash Course	1-3	Architectures, optimization, generalization
ApplMath for ML	4-6	Theory, regularization, PDEs
ML for ApplMath	7-10	Operators, inverse problems, discovery

Thanks to the organizers, audience, and NSF and UH for this workshop!

References I

- Fawzi, A. et al. (2022). "Discovering faster matrix multiplication algorithms with reinforcement learning". In: *Nature* 610.7930, pp. 47–53.
- Kolda, T. G. and B. W. Bader (2009). "Tensor decompositions and applications". In: *SIAM Review* 51.3, pp. 455–500.
- Moura, L. de and S. Ullrich (2021). "The Lean 4 Theorem Prover and Programming Language". In: *International Conference on Automated Deduction (CADE)*. Springer, pp. 625–635.
- Novikov, A., A. Fawzi, et al. (2025). "AlphaEvolve: A coding agent for scientific and algorithmic discovery". In: *Google DeepMind Technical Report*.
- Song, P. et al. (2024). "Lean Copilot: Large Language Models as Copilots for Theorem Proving in Lean". In: *Advances in Neural Information Processing Systems (NeurIPS)*.
- Strassen, V. (1969). "Gaussian elimination is not optimal". In: *Numerische Mathematik* 13.4, pp. 354–356.