Numerical Linear Algebra + Lean + AI

My Personal Experience

Haim Avron
Tel Aviv University

NLA + Lean Working Group
October 27, 2026

About This Talk

- Based on my **personal experience** trying Lean with my NLA work.

About This Talk

- Based on my **personal experience** trying Lean with my NLA work.
- My aim: making Lean part of my **daily research workflow**.

About This Talk

- Based on my **personal experience** trying Lean with my NLA work.
- My aim: making Lean part of my **daily research workflow**.
- Plan:
  1. Lean + Mathlib in a nutshell.
  2. Why Lean + AI is a great combination (and an important warning).
  3. My personal experience.
  4. My current workflow for formalizing papers.
  5. Looking forward: a proposal for our community.

About This Talk

- Based on my **personal experience** trying Lean with my NLA work.
- My aim: making Lean part of my **daily research workflow**.
- Plan:
  1. Lean + Mathlib in a nutshell.
  2. Why Lean + AI is a great combination (and an important warning).
  3. My personal experience.
  4. My current workflow for formalizing papers.
  5. Looking forward: a proposal for our community.

Disclaimer: I am not a Lean expert!

Lean in a Nutshell

- Lean: an open-source **functional programming language** and **interactive theorem prover**.
  Created by Leonardo de Moura (2013); today Lean 4, developed by the *Lean Focused Research Organization* since 2023.
- Based on **dependent type theory**: statements are types, and a proof of P is a term (element) of type P.
- Proofs are usually written using **tactics**; a small trusted **kernel** checks them.

```
theorem two_ab_le (a b : ℝ) : 2 * a * b ≤ a ^ 2 + b ^ 2 := by
  have h : 0 ≤ (a - b) ^ 2 := sq_nonneg (a - b)
  calc 2 * a * b = a ^ 2 + b ^ 2 - (a - b) ^ 2 := by ring
    _ ≤ a ^ 2 + b ^ 2 := by linarith
```

[2021] de Moura and Ullrich, "The Lean 4 Theorem Prover and Programming Language"

Lean in a Nutshell

- Lean: an open-source **functional programming language** and **interactive theorem prover**.
  Created by Leonardo de Moura (2013); today Lean 4, developed by the *Lean Focused Research Organization* since 2023.
- Based on **dependent type theory**: statements are types, and a proof of P is a term (element) of type P.
- Proofs are usually written using **tactics**; a small trusted **kernel** checks them.

```
theorem two_ab_le (a b : ℝ) : 2 * a * b ≤ a ^ 2 + b ^ 2 := by
  have h : 0 ≤ (a - b) ^ 2 := sq_nonneg (a - b)
  calc 2 * a * b = a ^ 2 + b ^ 2 - (a - b) ^ 2 := by ring
    _ ≤ a ^ 2 + b ^ 2 := by linarith
```

If it compiles, it is proved… modulo the axioms used, and modulo *whether the statement says what you think it says.*

[2021] de Moura and Ullrich, "The Lean 4 Theorem Prover and Programming Language"

Mathlib: The Lean Mathematical Library

- A **community-driven, unified** library of formalized mathematics.
- Oct. 2026: ≈ 2.3M lines of Lean, ≈ 240K theorems, ≈ 950 contributors.
- Algebra, analysis, topology, measure theory, probability, …, linear algebra.
- Designed for **generality** (“make assumptions as weak as possible”).
  E.g., a matrix is just a function: `Matrix m n α := m → n → α`
  (any index types `m`, `n`; any type of entries `α`).

```
-- trace(AB) = trace(BA) for rectangular A and B
example {m n : ℕ} (A : Matrix (Fin m) (Fin n) ℝ)
    (B : Matrix (Fin n) (Fin m) ℝ) :
    (A * B).trace = (B * A).trace :=
  Matrix.trace_mul_comm A B
```

[2020] The mathlib Community, “The Lean Mathematical Library”

Numerical Linear Algebra in Mathlib (Oct. 2026)

In Mathlib:

- ✔ Determinant, trace, rank, inverse
- ✔ Kronecker products, the vec trick
- ✔ Hermitian, positive (semi)definite
- ✔ Spectral theorem (Hermitian)
- ✔ $\mathbf{LDL}^*$, Gaussian elimination, Gershgorin
- ✔ Matrix norms (opt-in)
- ✔ Singular values of linear maps (2026)

Numerical Linear Algebra in Mathlib (Oct. 2026)

In Mathlib:

- ✔ Determinant, trace, rank, inverse
- ✔ Kronecker products, the vec trick
- ✔ Hermitian, positive (semi)definite
- ✔ Spectral theorem (Hermitian)
- ✔ $\mathbf{LDL}^*$, Gaussian elimination, Gershgorin
- ✔ Matrix norms (opt-in)
- ✔ Singular values of linear maps (2026)

Not (yet) in Mathlib:

- ✘ SVD as a factorization
- ✘ Eckart–Young
- ✘ QR, Householder, Givens
- ✘ Krylov subspace methods
- ✘ Condition numbers, Courant–Fischer, Weyl
- ✘ Pseudoinverse
- ✘ Rounding-error analysis

Much of the infrastructure we need for NLA has to be built.

Why Formalize? Benefits and Challenges

Benefits

- Higher confidence in the correctness of results.
- Option for a “continuous integration” mode (more on that later).

Why Formalize? Benefits and Challenges

Benefits	Challenges
◦ Higher confidence in the correctness of results. ◦ Option for a “continuous integration” mode (more on that later).	◦ Formalizing results in Lean is challenging (for a human). ◦ For important enough results: a research project on its own.

A Research Project on Its Own: Fermat's Last Theorem

Formalizing a proof of Fermat's Last Theorem (FLT) in Lean:

- Led by Kevin Buzzard (Imperial College London).
  It follows a modern variant of Wiles's proof, along a route planned by Richard Taylor (who, in 1994, helped Wiles fix the gap in the original proof).
- A five-year EPSRC fellowship (£0.93M, Oct. 2024 – Sep. 2029).
- Stated goal: **reduce** FLT to results known by the end of the 1980s.
- Organized around a blueprint (a dependency graph of definitions and lemmas); dozens of contributors.

[2025] Buzzard and Taylor, *FLT* (`github.com/ImperialCollegeLondon/FLT`)

A Research Project on Its Own: Fermat's Last Theorem

Formalizing a proof of Fermat's Last Theorem (FLT) in Lean:

- Led by Kevin Buzzard (Imperial College London).
  It follows a modern variant of Wiles's proof, along a route planned by Richard Taylor (who, in 1994, helped Wiles fix the gap in the original proof).
- A five-year EPSRC fellowship (£0.93M, Oct. 2024 – Sep. 2029).
- Stated goal: **reduce** FLT to results known by the end of the 1980s.
- Organized around a blueprint (a dependency graph of definitions and lemmas); dozens of contributors.

Sep. 2026: Anthropic announces a complete computer-checked proof of FLT

Claude agents, working largely autonomously for 11 days: ≈ 13M lines of Lean, building in part on the Imperial project.

"I only promised the EPSRC that I would reduce FLT to the 1980s; this repo proves the whole thing." — K. Buzzard (Xena blog, Sep. 2026)

[2025] Buzzard and Taylor, *FLT* (`github.com/ImperialCollegeLondon/FLT`); [Sept. 2026] Anthropic, *Formalizing Fermat's Last Theorem*

Lean + AI: A Great Combination

- **AI is very adept at writing Lean code.**
  It does not mind the complexity, or how time consuming it is.
  (Humans will not spend the time formalizing all their results.)

Lean + AI: A Great Combination

- **AI is very adept at writing Lean code.**
  It does not mind the complexity, or how time consuming it is.
  (Humans will not spend the time formalizing all their results.)
- **Lean in the feedback loop.**
  Formalization reveals mistakes in the AI's "proof", and the AI goes back and fixes it.

Lean + AI: A Great Combination

- **AI is very adept at writing Lean code.**
  It does not mind the complexity, or how time consuming it is.
  (Humans will not spend the time formalizing all their results.)
- **Lean in the feedback loop.**
  Formalization reveals mistakes in the AI's "proof", and the AI goes back and fixes it.
- **Reduces what needs to be checked.**
  If you mainly care about correctness: check that the *statements* were translated correctly.

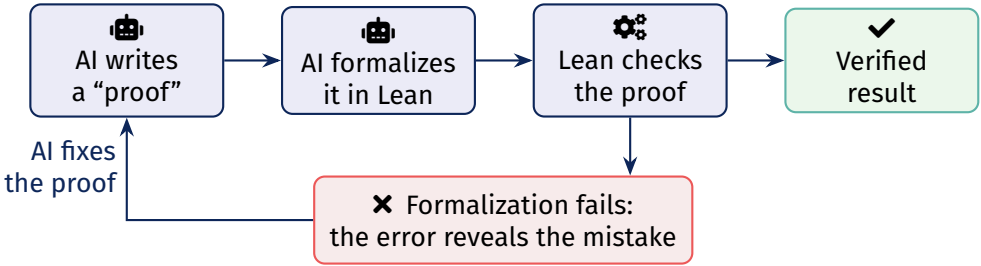
Lean + AI: A Great Combination

- **AI is very adept at writing Lean code.**
  It does not mind the complexity, or how time consuming it is.
  (Humans will not spend the time formalizing all their results.)
- **Lean in the feedback loop.**
  Formalization reveals mistakes in the AI's "proof", and the AI goes back and fixes it.
- **Reduces what needs to be checked.**
  If you mainly care about correctness: check that the *statements* were translated correctly.
- **Makes "continuous integration" realistic.**
  Results in an on-going project are formalized as soon as they are written (more on this later).

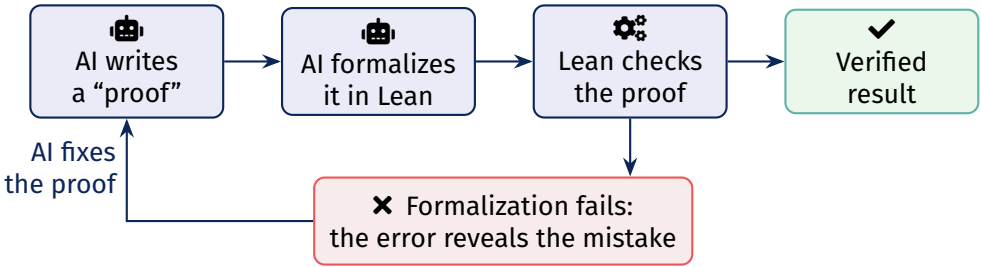
Lean + AI: A Great Combination

- **AI is very adept at writing Lean code.**
  It does not mind the complexity, or how time consuming it is.
  (Humans will not spend the time formalizing all their results.)
- **Lean in the feedback loop.**
  Formalization reveals mistakes in the AI's "proof", and the AI goes back and fixes it.
- **Reduces what needs to be checked.**
  If you mainly care about correctness: check that the *statements* were translated correctly.
- **Makes "continuous integration" realistic.**
  Results in an on-going project are formalized as soon as they are written (more on this later).
- **Allows: proof first in Lean, then translate to English** (more on this later).

Lean in the Feedback Loop



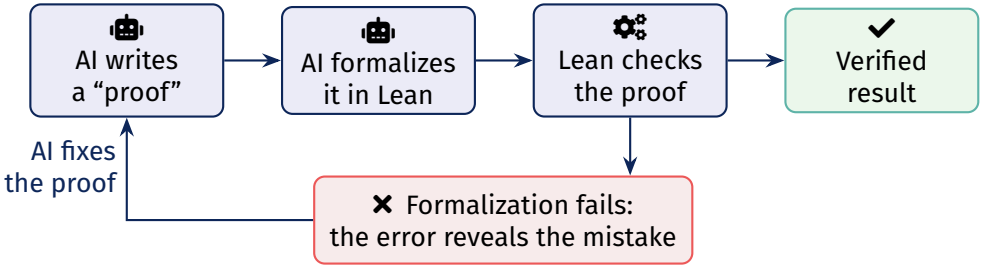
Lean in the Feedback Loop



A real example: comments Claude left inside a Lean proof (from one of my formalization projects, more later):

```
-- Hmm, maybe I have a sign error in the FOC. Let me reconsider.
...
-- Wait, that's POSITIVE 2 M_re Cuv, not negative! Let me recheck foc_v above.
...
-- The sign is wrong in foc_v! Let me fix it.
```

Lean in the Feedback Loop



A real example: comments Claude left inside a Lean proof (from one of my formalization projects, more later):

```
-- Hmm, maybe I have a sign error in the FOC. Let me reconsider.
...
-- Wait, that's POSITIVE 2 M_re Cuv, not negative! Let me recheck foc_v above.
...
-- The sign is wrong in foc_v! Let me fix it.
```

This is less of an issue with the latest models.

What Needs to Be Checked?

```
/-- `X = U S V^H` is a singular value decomposition. -/
def IsSVD (X : Matrix (Fin m) (Fin p) 𝕜) (U : Matrix (Fin m) (Fin m) 𝕜)
    (S : Matrix (Fin m) (Fin p) 𝕜) (V : Matrix (Fin p) (Fin p) 𝕜) : Prop :=
  U ∈ unitaryGroup (Fin m) 𝕜 ∧ V ∈ unitaryGroup (Fin p) 𝕜 ∧ IsSingularDiag S ∧
    X = U * S * Vᴴ

theorem eckart_young {X : Matrix (Fin m) (Fin p) 𝕜} {U S V} (h : IsSVD X U S V)
    (r : ℕ) (Y : Matrix (Fin m) (Fin p) 𝕜) (hY : Y.rank ≤ r) :
    frobSq (X - truncProd U S V r) ≤ frobSq (X - Y) := by
  ...
```

The Eckart-Young theorem, from one of my formalizations (more later).

What Needs to Be Checked?

```
/-- `X = U S V^H` is a singular value decomposition. -/
def IsSVD (X : Matrix (Fin m) (Fin p) 𝕜) (U : Matrix (Fin m) (Fin m) 𝕜)
    (S : Matrix (Fin m) (Fin p) 𝕜) (V : Matrix (Fin p) (Fin p) 𝕜) : Prop :=
  U ∈ unitaryGroup (Fin m) 𝕜 ∧ V ∈ unitaryGroup (Fin p) 𝕜 ∧ IsSingularDiag S ∧
    X = U * S * Vᴴ

theorem eckart_young {X : Matrix (Fin m) (Fin p) 𝕜} {U S V} (h : IsSVD X U S V)
    (r : ℕ) (Y : Matrix (Fin m) (Fin p) 𝕜) (hY : Y.rank ≤ r) :
    frobSq (X - truncProd U S V r) ≤ frobSq (X - Y) := by
  ...
```

The Eckart-Young theorem, from one of my formalizations (more later).

- **You check:** the statement and the definitions it uses (`IsSVD`, `frobSq`, `truncProd`, …): ≈ 20 lines.

What Needs to Be Checked?

```
/-- `X = U S V^H` is a singular value decomposition. -/
def IsSVD (X : Matrix (Fin m) (Fin p) 𝕜) (U : Matrix (Fin m) (Fin m) 𝕜)
    (S : Matrix (Fin m) (Fin p) 𝕜) (V : Matrix (Fin p) (Fin p) 𝕜) : Prop :=
  U ∈ unitaryGroup (Fin m) 𝕜 ∧ V ∈ unitaryGroup (Fin p) 𝕜 ∧ IsSingularDiag S ∧
    X = U * S * Vᴴ

theorem eckart_young {X : Matrix (Fin m) (Fin p) 𝕜} {U S V} (h : IsSVD X U S V)
    (r : ℕ) (Y : Matrix (Fin m) (Fin p) 𝕜) (hY : Y.rank ≤ r) :
    frobSq (X - truncProd U S V r) ≤ frobSq (X - Y) := by
  ...
```

The Eckart-Young theorem, from one of my formalizations (more later).

- **You check:** the statement and the definitions it uses (`IsSVD`, `frobSq`, `truncProd`, …): ≈ 20 lines.
- **Lean checks:** the proof, and the lemmas it relies on (the file has ≈ 700 lines).

What Needs to Be Checked?

```
/-- `X = U S V^H` is a singular value decomposition. -/
def IsSVD (X : Matrix (Fin m) (Fin p) 𝕜) (U : Matrix (Fin m) (Fin m) 𝕜)
    (S : Matrix (Fin m) (Fin p) 𝕜) (V : Matrix (Fin p) (Fin p) 𝕜) : Prop :=
  U ∈ unitaryGroup (Fin m) 𝕜 ∧ V ∈ unitaryGroup (Fin p) 𝕜 ∧ IsSingularDiag S ∧
    X = U * S * V^H

theorem eckart_young {X : Matrix (Fin m) (Fin p) 𝕜} {U S V} (h : IsSVD X U S V)
    (r : ℕ) (Y : Matrix (Fin m) (Fin p) 𝕜) (hY : Y.rank ≤ r) :
    frobSq (X - truncProd U S V r) ≤ frobSq (X - Y) := by
  ...
```

The Eckart-Young theorem, from one of my formalizations (more later).

- **You check:** the statement and the definitions it uses (`IsSVD`, `frobSq`, `truncProd`, …): ≈ 20 lines.
- **Lean checks:** the proof, and the lemmas it relies on (the file has ≈ 700 lines).

If you mainly care about correctness: check the statements, not the proofs.

```
/-- **Eckart–Young theorem** (Frobenius norm): the truncated SVD is an optimal rank-`r`
approximation. -/
theorem eckart_young {X : Matrix (Fin m) (Fin p) 𝕜} {U S V} (h : IsSVD X U S V) (r : ℕ)
    (Y : Matrix (Fin m) (Fin p) 𝕜) (hY : Y.rank ≤ r) :
    frobSq (X - truncProd U S V r) ≤ frobSq (X - Y) := by
  have hsvd := h
  obtain ⟨hU, hV, hS, hX⟩ := h
  obtain ⟨P, hPh, hPP, hPY, hPtr⟩ := SVD.exists_proj Y
  have hUU := SVD.unitary_star_mul hU
  have hUU' := SVD.unitary_mul_star hU
  have hVV := SVD.unitary_star_mul hV
  have hVV' := SVD.unitary_mul_star hV
  rw [hsvd.frobSq_sub_truncProd r]
  have key : frobSq (X - Y) = SVD.fsq (S - U^H * Y * V) := by
    rw [SVD.frobSq_eq_fsq
      ← SVD.fsq_mul_left_of_mem_unitary (by rw [conjTranspose_conjTranspose]; exact hU') (X - Y)
      ← SVD.fsq_mul_right (U^H * (X - Y)) V hVV']
    congr 1
    rw [hX, Matrix.mul_sub, Matrix.sub_mul]
    congr 1
    simp only [Matrix.mul_assoc]
    rw [hVV, Matrix.mul_one, ← Matrix.mul_assoc, hUU, Matrix.one_mul]
  rw [key]
  apply SVD.core hS r (U^H * P * U) ?_ ?_ ?_ (U^H * Y * V) ?_
  · simp only [conjTranspose_mul, conjTranspose_conjTranspose, hPh, Matrix.mul_assoc]
  · simp only [Matrix.mul_assoc]
    rw [← Matrix.mul_assoc U U^H, hUU', Matrix.one_mul, ← Matrix.mul_assoc P P, hPP]
  · rw [trace_mul_comm, ← Matrix.mul_assoc, hUU', Matrix.one_mul, hPtr]
    exact_mod_cast hY
  · simp only [Matrix.mul_assoc]
    rw [← Matrix.mul_assoc U U^H, hUU', Matrix.one_mul, ← Matrix.mul_assoc P Y, hPY]
```

Important Warning:
Not “Formalize and Forget”!

You Need to Validate the Output

1. The results are **stated correctly** in Lean:
   the theorem *statements* (and definitions), not the proofs themselves.

You Need to Validate the Output

1. The results are **stated correctly** in Lean:
   the theorem *statements* (and definitions), not the proofs themselves.
2. The formalization is **complete**, and did not skip anything.

You Need to Validate the Output

1. The results are **stated correctly** in Lean:
   the theorem *statements* (and definitions), not the proofs themselves.
2. The formalization is **complete**, and did not skip anything.
3. No `sorry` or **unwanted axioms** were used:

```
#print axioms two_ab_le   -- the proof from the "Lean in a Nutshell" slide
-- 'two_ab_le' depends on axioms: [propext, Classical.choice, Quot.sound]

#print axioms stuck       -- same statement, proved with `sorry`
-- 'stuck' depends on axioms: [propext, sorryAx, Classical.choice, Quot.sound]

#print axioms assumed     -- same statement, proved from an `axiom` we added
-- 'assumed' depends on axioms: [my_axiom, propext, Classical.choice, Quot.sound]
```

You Need to Validate the Output

1. The results are **stated correctly** in Lean:
   the theorem *statements* (and definitions), not the proofs themselves.
2. The formalization is **complete**, and did not skip anything.
3. No `sorry` or **unwanted axioms** were used:

```
#print axioms two_ab_le   -- the proof from the "Lean in a Nutshell" slide
-- 'two_ab_le' depends on axioms: [propext, Classical.choice, Quot.sound]

#print axioms stuck       -- same statement, proved with `sorry`
-- 'stuck' depends on axioms: [propext, sorryAx, Classical.choice, Quot.sound]

#print axioms assumed     -- same statement, proved from an `axiom` we added
-- 'assumed' depends on axioms: [my_axiom, propext, Classical.choice, Quot.sound]
```

A false axiom makes everything provable. (It happened to me: an axiom added by the AI turned out to be false.)

Case Study: Formalizing a Paper

Early 2026: I set out to formalize my paper

Haim Avron and Uria Mor (2025). “Demystifying Tubal Tensor Algebra”. In: arXiv: 2506.03311 [math.NA]

- Tubal tensors: third-order tensors viewed as matrices of *tubes* $\boldsymbol{a} \in \mathbb{R}^n$, multiplied via $\boldsymbol{a} \star_{\mathbf{M}} \boldsymbol{b} = \mathbf{M}^{-1}(\mathbf{M}\boldsymbol{a} \odot \mathbf{M}\boldsymbol{b})$.
- Main result: the $\star_{\mathbf{M}}$-product is the *only* way to define a tubal product with a few desired algebraic properties.

arXiv:2506.03311

Case Study: Formalizing a Paper

Early 2026: I set out to formalize my paper

Haim Avron and Uria Mor (2025). “Demystifying Tubal Tensor Algebra”. In: arXiv: 2506.03311 [math.NA]

- Tubal tensors: third-order tensors viewed as matrices of *tubes* $\boldsymbol{a} \in \mathbb{R}^n$, multiplied via $\boldsymbol{a} \star_{\mathbf{M}} \boldsymbol{b} = \mathbf{M}^{-1}(\mathbf{M}\boldsymbol{a} \odot \mathbf{M}\boldsymbol{b})$.
- Main result: the $\star_{\mathbf{M}}$-product is the *only* way to define a tubal product with a few desired algebraic properties.

arXiv:2506.03311

Why this paper?

- Mainly mathematical theorems, almost no algorithmic content.
- Many results: 71 numbered items (8 theorems, 9 propositions, 21 lemmas, 3 corollaries, 16 definitions, 12 examples, …).
- And also… because I really like this paper.

Take 1 (Feb.–Mar. 2026): Claude Opus 4.6

- **All 58 numbered results formalized** (examples were out of scope), including needed classical results that are not in Mathlib (e.g., the SVD and Eckart–Young).
  No `sorry`, no axioms.



github.com/haimav/
formalize-tubal-demyst-paper

Take 1 (Feb.–Mar. 2026): Claude Opus 4.6

- **All 58 numbered results formalized** (examples were out of scope), including needed classical results that are not in Mathlib (e.g., the SVD and Eckart–Young).
  No `sorry`, no axioms.
- Considerable effort: **almost two weeks**.
  It got stuck many times, and needed my help and pushing.

github.com/haimav/
formalize-tubal-demyst-paper

Take 1 (Feb.–Mar. 2026): Claude Opus 4.6

- **All 58 numbered results formalized** (examples were out of scope), including needed classical results that are not in Mathlib (e.g., the SVD and Eckart–Young).
  No `sorry`, no axioms.
- Considerable effort: **almost two weeks**.
  It got stuck many times, and needed my help and pushing.
- In some places it tried to prove only **weaker statements**.
  I discovered this by verifying the theorem statements.
  (E.g., the t-SVD without the ordering of the singular tubes.)

github.com/haimav/
formalize-tubal-demyst-paper

Take 1: Formalization Uncovered a Missing Assumption

- A central theorem (multirank Eckart–Young) was **missing an assumption**: the truncation multirank r must respect the conjugation pattern of $\mathbf{M}$.

```
def RespectsConjPairing (td : TubalData n) (r : Fin n → ℕ) : Prop :=
  ∀ k k', k ≠ k' → IsConjRow td.M k k' → r k = r k'

theorem tsvd_eckart_young_multirank
    (hsc : IsScaledUnitary td)
    {A : TubalMatrix m p n td} (svd : TSVD A)
    (B : TubalMatrix m p n td) (r : Fin n → ℕ)
    (hr : RespectsConjPairing td r)
    (hB : HasStarMMultirankAtMost B r) :
    tubalFrobNormSq (A - tsvdTruncMulti svd r hr) ≤ tubalFrobNormSq (A - B) := by
```

Take 1: Formalization Uncovered a Missing Assumption

- A central theorem (multirank Eckart–Young) was **missing an assumption**: the truncation multirank r must respect the conjugation pattern of $\mathbf{M}$.

```
def RespectsConjPairing (td : TubalData n) (r : Fin n → ℕ) : Prop :=
  ∀ k k', k ≠ k' → IsConjRow td.M k k' → r k = r k'

theorem tsvd_eckart_young_multirank
    (hsc : IsScaledUnitary td)
    {A : TubalMatrix m p n td} (svd : TSVD A)
    (B : TubalMatrix m p n td) (r : Fin n → ℕ)
    (hr : RespectsConjPairing td r)
    (hB : HasStarMMultirankAtMost B r) :
    tubalFrobNormSq (A - tsvdTruncMulti svd r hr) ≤ tubalFrobNormSq (A - B) := by
```

Take 1: Formalization Uncovered a Missing Assumption

- A central theorem (multirank Eckart–Young) was **missing an assumption**: the truncation multirank r must respect the conjugation pattern of $\mathbf{M}$.

```
def RespectsConjPairing (td : TubalData n) (r : Fin n → ℕ) : Prop :=
  ∀ k k', k ≠ k' → IsConjRow td.M k k' → r k = r k'

theorem tsvd_eckart_young_multirank
    (hsc : IsScaledUnitary td)
    {A : TubalMatrix m p n td} (svd : TSVD A)
    (B : TubalMatrix m p n td) (r : Fin n → ℕ)
    (hr : RespectsConjPairing td r)
    (hB : HasStarMMultirankAtMost B r) :
    tubalFrobNormSq (A - tsvdTruncMulti svd r hr) ≤ tubalFrobNormSq (A - B) := by
```

- We already knew about this omission from follow-up work[1], and planned to fix it in the next revision (done in the June 2026 revision).

[1] [2025] Mor, "Sufficient and Necessary Conditions for Eckart-Young like Result for Tubal Tensors"

Take 1: Formalization Uncovered a Missing Assumption

- A central theorem (multirank Eckart–Young) was **missing an assumption**: the truncation multirank r must respect the conjugation pattern of $\mathbf{M}$.

```
def RespectsConjPairing (td : TubalData n) (r : Fin n → ℕ) : Prop :=
  ∀ k k', k ≠ k' → IsConjRow td.M k k' → r k = r k'

theorem tsvd_eckart_young_multirank
    (hsc : IsScaledUnitary td)
    {A : TubalMatrix m p n td} (svd : TSVD A)
    (B : TubalMatrix m p n td) (r : Fin n → ℕ)
    (hr : RespectsConjPairing td r)
    (hB : HasStarMMultirankAtMost B r) :
    tubalFrobNormSq (A - tsvdTruncMulti svd r hr) ≤ tubalFrobNormSq (A - B) := by
```

- We already knew about this omission from follow-up work[1], and planned to fix it in the next revision (done in the June 2026 revision).
- It convinced me that auto-formalization of results in the style I work on is doable.

[1] [2025] Mor, "Sufficient and Necessary Conditions for Eckart-Young like Result for Tubal Tensors"

Prove First in Lean, Then Translate to English

- Proposition 13: the rings $\mathbb{K}_{\mathbf{M}}$ and $\mathbb{K}_{\mathbf{M}'}$ are isomorphic **if** they have the same realness (number of real rows of $\mathbf{M}$).
  In the paper, we proved only this **sufficient** condition.

Prove First in Lean, Then Translate to English

- Proposition 13: the rings $\mathbb{K}_{\mathbf{M}}$ and $\mathbb{K}_{\mathbf{M}'}$ are isomorphic **if** they have the same realness (number of real rows of $\mathbf{M}$).
  In the paper, we proved only this **sufficient** condition.
- I asked Claude to formalize that the condition is also **necessary**. It did:

```
/-- **Proposition 13 (iff)**: Two tubal rings are isomorphic iff they have
    the same realness. -/
theorem KM_algEquiv_iff_realness_eq (td₁ td₂ : TubalData n) :
    Nonempty (KM n td₁ ≃ₐ[ℝ] KM n td₂) ↔ realness td₁ = realness td₂ :=
  ⟨realness_of_algEquiv td₁ td₂, KM_algEquiv_of_realness_eq td₁ td₂⟩
```

Prove First in Lean, Then Translate to English

- Proposition 13: the rings $\mathbb{K}_{\mathbf{M}}$ and $\mathbb{K}_{\mathbf{M}'}$ are isomorphic **if** they have the same realness (number of real rows of $\mathbf{M}$).
  In the paper, we proved only this **sufficient** condition.
- I asked Claude to formalize that the condition is also **necessary**. It did:

```
/-- **Proposition 13 (iff)**: Two tubal rings are isomorphic iff they have
    the same realness. -/
theorem KM_algEquiv_iff_realness_eq (td₁ td₂ : TubalData n) :
    Nonempty (KM n td₁ ≃ₐ[ℝ] KM n td₂) ↔ realness td₁ = realness td₂ :=
  ⟨realness_of_algEquiv td₁ td₂, KM_algEquiv_of_realness_eq td₁ td₂⟩
```

- I then asked it to **translate the Lean proof** to natural language. It did.
  Key idea: the number of $\mathbb{R}$-algebra homomorphisms $\mathbb{K}_{\mathbf{M}} \to \mathbb{R}$ equals the realness, and it is an isomorphism invariant.

Prove First in Lean, Then Translate to English

- Proposition 13: the rings $\mathbb{K}_{\mathbf{M}}$ and $\mathbb{K}_{\mathbf{M}'}$ are isomorphic **if** they have the same realness (number of real rows of $\mathbf{M}$).
  In the paper, we proved only this **sufficient** condition.
- I asked Claude to formalize that the condition is also **necessary**. It did:

```
/-- **Proposition 13 (iff)**: Two tubal rings are isomorphic iff they have
    the same realness. -/
theorem KM_algEquiv_iff_realness_eq (td₁ td₂ : TubalData n) :
    Nonempty (KM n td₁ ≃ₐ[ℝ] KM n td₂) ↔ realness td₁ = realness td₂ :=
  ⟨realness_of_algEquiv td₁ td₂, KM_algEquiv_of_realness_eq td₁ td₂⟩
```

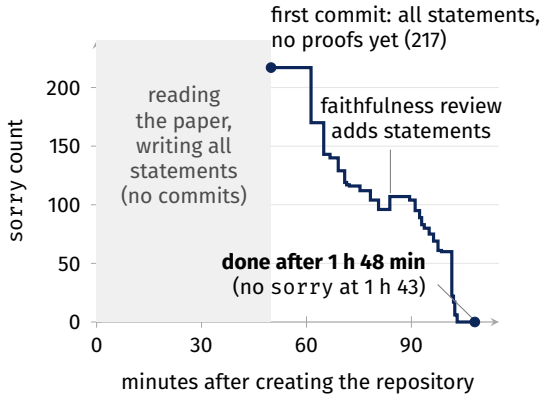
- I then asked it to **translate the Lean proof** to natural language. It did.
  Key idea: the number of $\mathbb{R}$-algebra homomorphisms $\mathbb{K}_{\mathbf{M}} \to \mathbb{R}$ equals the realness, and it is an isomorphism invariant.
- I verified the natural language proof.

Take 2 (Oct. 2026): Claude Opus 5.5

- In preparation for this talk, and to gauge progress, I redid the formalization from scratch.

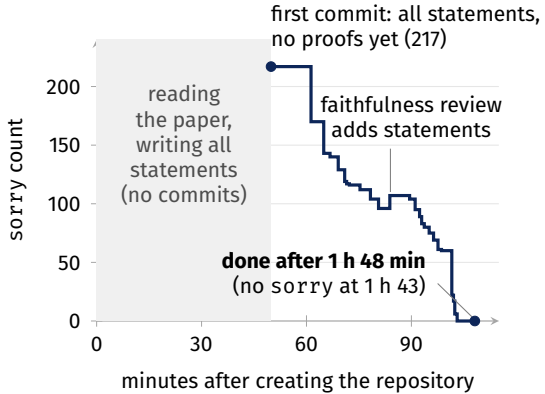
Take 2 (Oct. 2026): Claude Opus 5.5

- In preparation for this talk, and to gauge progress, I redid the formalization from scratch.
- Instead of two weeks with babysitting:
  under 2 hours, with no human intervention.



Take 2 (Oct. 2026): Claude Opus 5.5

- In preparation for this talk, and to gauge progress, I redid the formalization from scratch.
- Instead of two weeks with babysitting:
  under 2 hours, with no human intervention.
- A **single prompt** sufficed, run as a Claude workflow (parallel agents, plus independent faithfulness reviewers).



Take 2: The (Single) Prompt

Your goal is formalize in Lean the results of the attached paper. **Every numbered result** (theorem, proposition, corollary,...) needs to be formalized, and the statement **fully faithful to the paper**. You can assume as an axiom well known results such as Eckart-Young, but **do not just put in axioms or sorries just because you are stuck**. Continue until you are done. A few additional instructions:

1) First make sure you have a **working environment with lean and mathlib**. If you are having trouble setting this up, come back to me and I will help. Do not go on and blindly formalize without the ability to test it.
2) You should create a file called `mapping.md` that will map each result in the paper to the corresponding Lean theorem(s) (a result can be proven using multiple theorems). Use full paths (file and line number so I can easily find them). Add a column called **"Human Verified?"** left empty that I can fill.
3) **Be faithful to the statement in the paper** for the numbered results. Do not deviate from them, or add preconditions. Your formalization should reflect them exactly.
4) You do not have to follow the proof strategy in the paper. However you can view it as a recommendation.

Take 2: Outcomes

- The formalization completed **without my intervention**.
  No `sorry`, no axioms, not even for Eckart–Young (which the prompt allowed).

Take 2: Outcomes

- The formalization completed **without my intervention**.
  No `sorry`, no axioms, not even for Eckart–Young (which the prompt allowed).
- I then verified **every statement** (all rows of `mapping.md` are marked YES), and found **no errors**.
  Along the way, I asked questions to understand the statements, and asked for some clarifications and additional bridging lemmas.

Take 2: Outcomes

- The formalization completed **without my intervention**.
  No `sorry`, no axioms, not even for Eckart–Young (which the prompt allowed).
- I then verified **every statement** (all rows of `mapping.md` are marked YES), and found **no errors**.
  Along the way, I asked questions to understand the statements, and asked for some clarifications and additional bridging lemmas.
- **More complete** than Take 1: every numbered result (including all 12 examples), and every passing statement.

Take 2: Outcomes

- The formalization completed **without my intervention**.
  No `sorry`, no axioms, not even for Eckart–Young (which the prompt allowed).
- I then verified **every statement** (all rows of `mapping.md` are marked YES), and found **no errors**.
  Along the way, I asked questions to understand the statements, and asked for some clarifications and additional bridging lemmas.
- **More complete** than Take 1: every numbered result (including all 12 examples), and every passing statement.
- Uncovered several **tiny typos and missing assumptions**, which survived the original writing, two revisions, and the previous formalization effort.

Take 2: What It Found in the Paper (Examples)

Where	What the formalization revealed
Theorem 37	Error formula: $\lvert c\rvert^{-1}$ should be $\lvert c\rvert^{-2}$. Lean proves that the printed formula is false on the paper's own Example 68.
Lemma 38	"for all $\mathbf{A} \in \mathbb{C}^{n\times n}$" should be "for all $\mathbf{A} \in S$" (the literal statement is false).
Example 18	An index typo in the XOR product: the printed formula is not even commutative.
Proposition 71	Missing assumption $\mathbf{M} = \mathbf{DW}$ (false without it).

Numbering as in the June 2026 revision; all four issues were already in the original version (June 2025).

Take 2: What It Found in the Paper (Examples)

Where	What the formalization revealed
Theorem 37	Error formula: $\lvert c\rvert^{-1}$ should be $\lvert c\rvert^{-2}$. Lean proves that the printed formula is false on the paper's own Example 68.
Lemma 38	"for all $\mathbf{A} \in \mathbb{C}^{n\times n}$" should be "for all $\mathbf{A} \in S$" (the literal statement is false).
Example 18	An index typo in the XOR product: the printed formula is not even commutative.
Proposition 71	Missing assumption $\mathbf{M} = \mathbf{DW}$ (false without it).

```
theorem example68_literal_false :
    tnorm (A - B) ^ 2 ≠ (‖((Real.sqrt 2 : ℝ) : ℂ)‖)⁻¹ * (2 ^ 2 + 2 ^ 2) := by
  ...
```

Numbering as in the June 2026 revision; all four issues were already in the original version (June 2025).

Take 1 vs. Take 2

	Take 1 (Feb.–Mar. 2026)	**Take 2** (Oct. 2026)
Model	Claude Opus 4.6	Claude Opus 5.5
Time	≈ 2 weeks	Under 2 hours
Human intervention	Frequent: stuck, pushing, help	None during the run
Coverage	58 numbered results (examples out of scope)	All 71 numbered items (incl. examples), and passing statements
Statements	Some weaker than the paper's	All verified; no errors found
Found in the paper	A missing assumption (already known)	Several tiny typos and missing assumptions
Size	10.3K lines of Lean*	7.6K lines of Lean

*Excluding a 4K-line extension beyond the paper. Not a controlled experiment: model, tooling, and prompt all changed.

Take 1 vs. Take 2

	Take 1 (Feb.–Mar. 2026)	**Take 2** (Oct. 2026)
Model	Claude Opus 4.6	Claude Opus 5.5
Time	≈ 2 weeks	Under 2 hours
Human intervention	Frequent: stuck, pushing, help	None during the run
Coverage	58 numbered results (examples out of scope)	All 71 numbered items (incl. examples), and passing statements
Statements	Some weaker than the paper's	All verified; no errors found
Found in the paper	A missing assumption (already known)	Several tiny typos and missing assumptions
Size	10.3K lines of Lean*	7.6K lines of Lean

*Excluding a 4K-line extension beyond the paper. Not a controlled experiment: model, tooling, and prompt all changed.

Formalization helps uncover typos and missing assumptions.

	Take 1 (Feb.–Mar. 2026)	**Take 2** (Oct. 2026)
Model	Claude Opus 4.6	Claude Opus 5.5
Time	≈ 2 weeks	Under 2 hours
Human intervention	Frequent: stuck, needing help	None during the run
Coverage	20 numbered results (examples out of scope)	All 71 numbered items (incl. examples), and passing statements
Statements	Some weaker than the paper's	All verified; no errors found
Found in the paper	A missing assumption (already known)	Several tiny typos and missing assumptions
Size	10.3K lines of Lean*	7.6K lines of Lean

Clearly, Progress is Huge

Teaching: Formalized Lecture Notes

- Graduate course on tensor methods
  (Tel Aviv University, Spring 2026).

Teaching: Formalized Lecture Notes

- Graduate course on tensor methods (Tel Aviv University, Spring 2026).
- I used Claude to formalize the results of 6 of the 9 lectures: ≈ 17.6K lines of Lean, ≈ 540 theorems (9 classical facts taken as axioms).

Teaching: Formalized Lecture Notes

- Graduate course on tensor methods (Tel Aviv University, Spring 2026).
- I used Claude to formalize the results of 6 of the 9 lectures: ≈ 17.6K lines of Lean, ≈ 540 theorems (9 classical facts taken as axioms).
- The notes have a **clickable link** from each formalized result to its Lean proof.

Theorem. Let $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_D}$, and let $\mathbf{U}^{(1)} \in \mathbb{R}^{I_1 \times R_1}, \ldots, \mathbf{U}^{(D)} \in \mathbb{R}^{I_D \times R_D}$ be orthogonal matrices. Let $\mathcal{S} = \mathcal{X} \times_1 (\mathbf{U}^{(1)})^\mathsf{T} \times_2 \cdots \times_D (\mathbf{U}^{(D)})^\mathsf{T}$. Then, for any tensor $\mathcal{Z} \in \mathbb{R}^{R_1 \times \cdots \times R_D}$,

$$\left\| \mathcal{X} - [\![\mathcal{S} ; \{\mathbf{U}^{(d)}\}]\!] \right\|_F \leq \left\| \mathcal{X} - [\![\mathcal{Z} ; \{\mathbf{U}^{(d)}\}]\!] \right\|_F .$$

Proof (Lean: `TuckerDecomposition/OptimalCore.lean:optimal_core`). …

From Lecture 5 of the notes (Tucker decomposition); slightly abridged.

Teaching: Formalized Lecture Notes

- Graduate course on tensor methods (Tel Aviv University, Spring 2026).
- I used Claude to formalize the results of 6 of the 9 lectures: ≈ 17.6K lines of Lean, ≈ 540 theorems (9 classical facts taken as axioms).
- The notes have a **clickable link** from each formalized result to its Lean proof.
- The goal is **pedagogical**: introduce students to the concept of formalization.

Theorem. Let $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_D}$, and let $\mathbf{U}^{(1)} \in \mathbb{R}^{I_1 \times R_1}, \ldots, \mathbf{U}^{(D)} \in \mathbb{R}^{I_D \times R_D}$ be orthogonal matrices. Let $\mathcal{S} = \mathcal{X} \times_1 (\mathbf{U}^{(1)})^\mathsf{T} \times_2 \cdots \times_D (\mathbf{U}^{(D)})^\mathsf{T}$. Then, for any tensor $\mathcal{Z} \in \mathbb{R}^{R_1 \times \cdots \times R_D}$,

$$\left\| \mathcal{X} - [\![\mathcal{S} ; \{\mathbf{U}^{(d)}\}]\!] \right\|_F \leq \left\| \mathcal{X} - [\![\mathcal{Z} ; \{\mathbf{U}^{(d)}\}]\!] \right\|_F .$$

Proof (Lean: `TuckerDecomposition/OptimalCore.lean:optimal_core`). …

From Lecture 5 of the notes (Tucker decomposition); slightly abridged.

github.com/haimav/TensorsCourseFormalization

Continuous Integration: Formalize as You Write

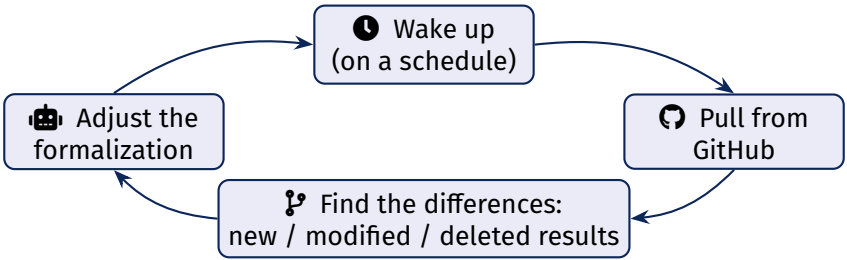
- An on-going project with a student: a folder for the formalization was set up early on.

Continuous Integration: Formalize as You Write

- An on-going project with a student: a folder for the formalization was set up early on.
- First, Claude formalized what we already had (not much).

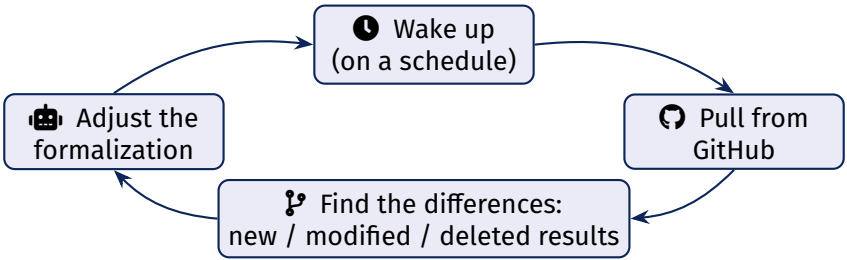
Continuous Integration: Formalize as You Write

- An on-going project with a student: a folder for the formalization was set up early on.
- First, Claude formalized what we already had (not much).
- Afterwards: Claude wakes up periodically (a scheduled routine), pulls from GitHub, and adjusts the formalization based on the differences.



Continuous Integration: Formalize as You Write

- An on-going project with a student: a folder for the formalization was set up early on.
- First, Claude formalized what we already had (not much).
- Afterwards: Claude wakes up periodically (a scheduled routine), pulls from GitHub, and adjusts the formalization based on the differences.



The formalization keeps pace with the paper, as it is being written.

MatrixLib: Bernstein's *Matrix Mathematics* in Lean

- Idea: a library aimed at formalizations in NLA, built by formalizing **all** the results in Bernstein's book.

[2009] Bernstein, *Matrix Mathematics: Theory, Facts, and Formulas*

MatrixLib: Bernstein's *Matrix Mathematics* in Lean

- Idea: a library aimed at formalizations in NLA, built by formalizing **all** the results in Bernstein's book.
- I worked on it for a couple of days (March 2026, Claude Opus 4.6), formalizing much of Chapter 2 ($\approx$ 6.7K lines of Lean). But then I was swamped with other work…

[2009] Bernstein, *Matrix Mathematics: Theory, Facts, and Formulas*

MatrixLib: Bernstein's *Matrix Mathematics* in Lean

- Idea: a library aimed at formalizations in NLA, built by formalizing **all** the results in Bernstein's book.
- I worked on it for a couple of days (March 2026, Claude Opus 4.6), formalizing much of Chapter 2 (≈ 6.7K lines of Lean). But then I was swamped with other work…
- Design: concrete (matrices over $\mathbb{R}$ or $\mathbb{C}$ only), following the book.

```
/-- If `A*B = 0` and `BA* = 0`, then `rank(A + B) = rank A + rank B`.

**Bernstein** Fact 2.10.29 -/
theorem rank_add_of_conjTranspose_mul_eq_zero (A B : FieldMatrix n m 𝕜)
    (h1 : Aᴴ * B = 0) (h2 : B * Aᴴ = 0) :
    rank (A + B) = rank A + rank B := by
```

[2009] Bernstein, *Matrix Mathematics: Theory, Facts, and Formulas*

MatrixLib: Bernstein's *Matrix Mathematics* in Lean

- Idea: a library aimed at formalizations in NLA, built by formalizing **all** the results in Bernstein's book.
- I worked on it for a couple of days (March 2026, Claude Opus 4.6), formalizing much of Chapter 2 (≈ 6.7K lines of Lean). But then I was swamped with other work…
- Design: concrete (matrices over $\mathbb{R}$ or $\mathbb{C}$ only), following the book.

```
/-- If `A*B = 0` and `BA* = 0`, then `rank(A + B) = rank A + rank B`.

**Bernstein** Fact 2.10.29 -/
theorem rank_add_of_conjTranspose_mul_eq_zero (A B : FieldMatrix n m 𝕜)
    (h1 : Aᴴ * B = 0) (h2 : B * Aᴴ = 0) :
    rank (A + B) = rank A + rank B := by
```

The idea is doable: it just needs some allocation of time.
(Done with an older model; probably faster with current models.)

[2009] Bernstein, *Matrix Mathematics: Theory, Facts, and Formulas*

Work in Progress: The Kronecker–Hutchinson Paper

- In preparation for this talk, I asked Claude (Opus 5.5) to formalize (from arXiv v2, Feb. 2025):

  Raphael A. Meyer and Haim Avron (2026). “Hutchinson’s Estimator is Bad at Kronecker-Trace-Estimation”. In: *SIAM Journal on Matrix Analysis and Applications* 47.1

Work in Progress: The Kronecker–Hutchinson Paper

- In preparation for this talk, I asked Claude (Opus 5.5) to formalize (from arXiv v2, Feb. 2025):

  Raphael A. Meyer and Haim Avron (2026). "Hutchinson's Estimator is Bad at Kronecker-Trace-Estimation". In: *SIAM Journal on Matrix Analysis and Applications* 47.1
- About 4 hours, unattended: ≈ 13K lines of Lean, no `sorry`, no axioms.

Work in Progress: The Kronecker–Hutchinson Paper

- In preparation for this talk, I asked Claude (Opus 5.5) to formalize (from arXiv v2, Feb. 2025):

  Raphael A. Meyer and Haim Avron (2026). "Hutchinson's Estimator is Bad at Kronecker-Trace-Estimation". In: *SIAM Journal on Matrix Analysis and Applications* 47.1
- About 4 hours, unattended: ≈ 13K lines of Lean, no `sorry`, no axioms.
- Claude flagged several statements in arXiv v2 as **false as literally written** (e.g., edge cases such as $d = 1$, a missing linear-independence assumption, a wrong constant), *but no fundamental error*, each with a formal counterexample and a *corrected version*.

Work in Progress: The Kronecker–Hutchinson Paper

- In preparation for this talk, I asked Claude (Opus 5.5) to formalize (from arXiv v2, Feb. 2025):

  Raphael A. Meyer and Haim Avron (2026). "Hutchinson's Estimator is Bad at Kronecker-Trace-Estimation". In: *SIAM Journal on Matrix Analysis and Applications* 47.1
- About 4 hours, unattended: ≈ 13K lines of Lean, no `sorry`, no axioms.
- Claude flagged several statements in arXiv v2 as **false as literally written** (e.g., edge cases such as $d = 1$, a missing linear-independence assumption, a wrong constant), *but no fundamental error*, each with a formal counterexample and a *corrected version*.
- Verifying the definitions led to a change: Claude had used its own closed-form formulas (e.g., for the partial trace). Now all results use the paper's **literal definitions**, with the equivalences proved (≈ 16K lines, still no `sorry`).

Work in Progress: The Kronecker–Hutchinson Paper

- In preparation for this talk, I asked Claude (Opus 5.5) to formalize (from arXiv v2, Feb. 2025):

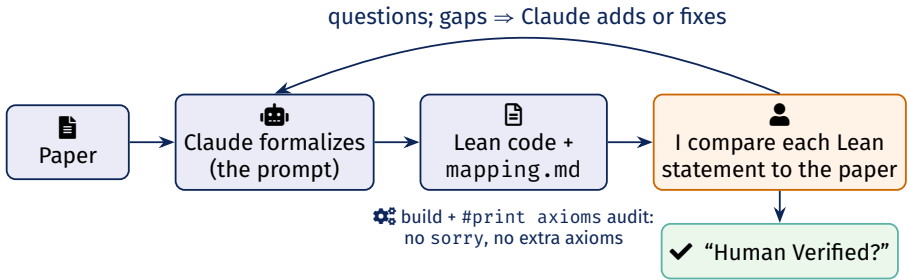
  Raphael A. Meyer and Haim Avron (2026). "Hutchinson's Estimator is Bad at Kronecker-Trace-Estimation". In: *SIAM Journal on Matrix Analysis and Applications* 47.1

- About 4 hours, unattended: ≈ 13K lines of Lean, no `sorry`, no axioms.
- Claude flagged several statements in arXiv v2 as **false as literally written** (e.g., edge cases such as $d = 1$, a missing linear-independence assumption, a wrong constant), *but no fundamental error*, each with a formal counterexample and a *corrected version*.
- Verifying the definitions led to a change: Claude had used its own closed-form formulas (e.g., for the partial trace). Now all results use the paper's **literal definitions**, with the equivalences proved (≈ 16K lines, still no `sorry`).

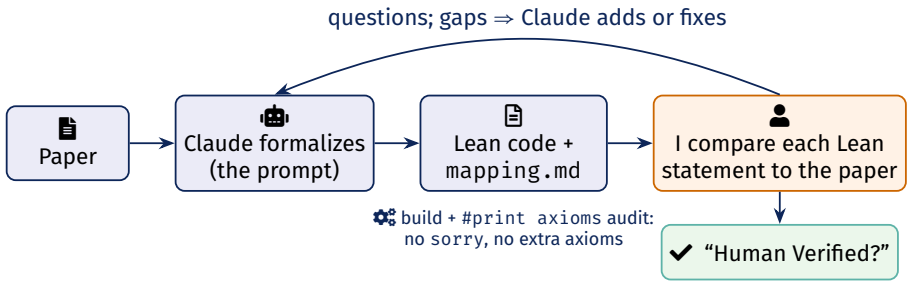
Status (Oct. 26): 50 of 56 definitions verified; the results (incl. the flagged ones) are next.

Until I verify them, the flags are Claude's claims, not mine.

My Current Workflow for Formalizing Papers

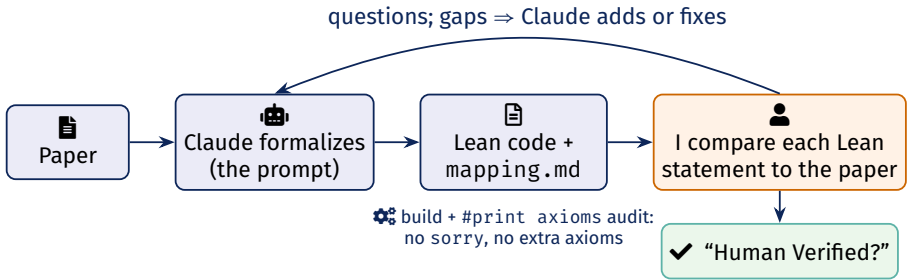


My Current Workflow for Formalizing Papers



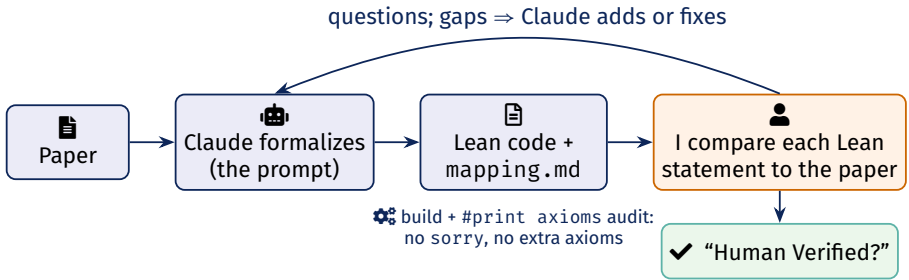
1. Ask Claude to formalize the paper, using the prompt shown earlier.

My Current Workflow for Formalizing Papers



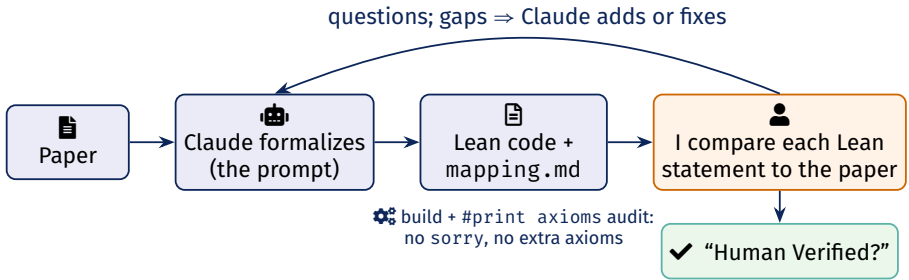
1. Ask Claude to formalize the paper, using the prompt shown earlier.
2. Go over `mapping.md` slowly: for every result, read the statement in Lean and compare it to the paper's.

My Current Workflow for Formalizing Papers



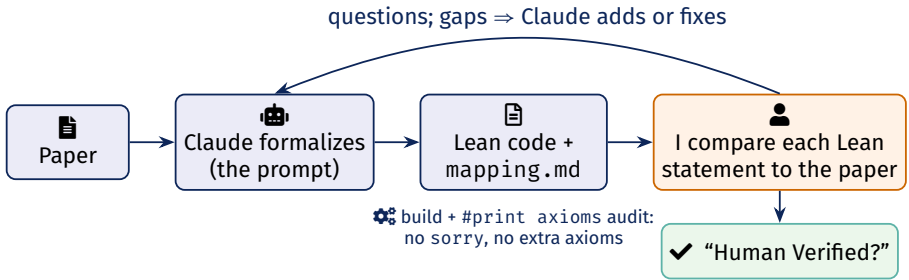
1. Ask Claude to formalize the paper, using the prompt shown earlier.
2. Go over `mapping.md` slowly: for every result, read the statement in Lean and compare it to the paper's.
3. This usually requires asking Claude questions, in order to understand what it did.

My Current Workflow for Formalizing Papers



1. Ask Claude to formalize the paper, using the prompt shown earlier.
2. Go over `mapping.md` slowly: for every result, read the statement in Lean and compare it to the paper's.
3. This usually requires asking Claude questions, in order to understand what it did.
4. If there are gaps, or things that need adding, I ask Claude to add them.

My Current Workflow for Formalizing Papers



1. Ask Claude to formalize the paper, using the prompt shown earlier.
2. Go over `mapping.md` slowly: for every result, read the statement in Lean and compare it to the paper's.
3. This usually requires asking Claude questions, in order to understand what it did.
4. If there are gaps, or things that need adding, I ask Claude to add them.
5. I do **not** verify the actual Lean proofs (this is next to impossible).

`mapping.md`: Where the Human Verification Happens

Excerpt from Take 2:

Paper result	Lean theorem	Location	Human Verified?
Theorem 11 (main result)	`theorem11`	`Section9/Main.lean:44`	✓
Proposition 13	`prop13`	`Canonical.lean:368`	✓
Theorem 30 (t-SVD)	`theorem30`	`TSVD.lean:172`	✓
Theorem 37 ⚠	`theorem37`	`EckartYoung.lean:457`	✓
Lemma 38 ⚠	`lemma38`	`Section9/SimulDiag.lean:148`	✓
Lemma 8	`lemma8`	`KMRing.lean:264`	✓

⚠ = deviation from the literal text of the paper (a typo), explained in `mapping.md`.

`mapping.md`: Where the Human Verification Happens

Excerpt from Take 2:

Paper result	Lean theorem	Location	Human Verified?
Theorem 11 (main result)	`theorem11`	`Section9/Main.lean:44`	✓
Proposition 13	`prop13`	`Canonical.lean:368`	✓
Theorem 30 (t-SVD)	`theorem30`	`TSVD.lean:172`	✓
Theorem 37 ⚠	`theorem37`	`EckartYoung.lean:457`	✓
Lemma 38 ⚠	`lemma38`	`Section9/SimulDiag.lean:148`	✓
Lemma 8	`lemma8`	`KMRing.lean:264`	✓

⚠ = deviation from the literal text of the paper (a typo), explained in `mapping.md`.

Status: **all 109 rows verified**. Every Lean name links to its file and line.

Verifying Statements Requires Knowing Lean

Proposition 6 of the paper ($\mathbb{K}_{\mathbf{M}}$ is a $*$-ring), as stated in Take 2, where `KM M` is $\mathbb{K}_{\mathbf{M}}$:

```
theorem prop6 (M : Matrix (Fin n) (Fin n) ℂ) [Fact (Admissible M)] :
    (∀ a : KM M, star (star a) = a) ∧
    (∀ a b : KM M, star (a + b) = star a + star b) ∧
    (∀ a b : KM M, star (a * b) = star b * star a) ∧
    star (1 : KM M) = 1 := by
  ...
```

Verifying Statements Requires Knowing Lean

Proposition 6 of the paper ($\mathbb{K}_{\mathbf{M}}$ is a $*$-ring), as stated in Take 2, where `KM M` is $\mathbb{K}_{\mathbf{M}}$:

```
theorem prop6 (M : Matrix (Fin n) (Fin n) ℂ) [Fact (Admissible M)] :
    (∀ a : KM M, star (star a) = a) ∧
    (∀ a b : KM M, star (a + b) = star a + star b) ∧
    (∀ a b : KM M, star (a * b) = star b * star a) ∧
    star (1 : KM M) = 1 := by
  ...
```

- `[Fact (Admissible M)]` is a **type-class** argument. Here it is a hypothesis: the paper's standing assumption ($\mathbf{M}$ invertible, plus the row condition).

Verifying Statements Requires Knowing Lean

Proposition 6 of the paper ($\mathbb{K}_{\mathbf{M}}$ is a $*$-ring), as stated in Take 2, where `KM M` is $\mathbb{K}_{\mathbf{M}}$:

```
theorem prop6 (M : Matrix (Fin n) (Fin n) ℂ) [Fact (Admissible M)] :
    (∀ a : KM M, star (star a) = a) ∧
    (∀ a b : KM M, star (a + b) = star a + star b) ∧
    (∀ a b : KM M, star (a * b) = star b * star a) ∧
    star (1 : KM M) = 1 := by
  ...
```

- `[Fact (Admissible M)]` is a **type-class** argument. Here it is a hypothesis: the paper's standing assumption ($\mathbf{M}$ invertible, plus the row condition).
- What do `*` and `star` mean on `KM M`? Type-class **instances**, defined elsewhere:

```
instance : Mul (KM M) :=
  ⟨fun a b => ⟨fun i => (starMC M (toC a.vec) (toC b.vec) i).re⟩⟩
```

Verifying Statements Requires Knowing Lean

Proposition 6 of the paper ($\mathbb{K}_{\mathbf{M}}$ is a $*$-ring), as stated in Take 2, where `KM M` is $\mathbb{K}_{\mathbf{M}}$:

```
theorem prop6 (M : Matrix (Fin n) (Fin n) ℂ) [Fact (Admissible M)] :
    (∀ a : KM M, star (star a) = a) ∧
    (∀ a b : KM M, star (a + b) = star a + star b) ∧
    (∀ a b : KM M, star (a * b) = star b * star a) ∧
    star (1 : KM M) = 1 := by
  ...
```

- `[Fact (Admissible M)]` is a **type-class** argument. Here it is a hypothesis: the paper's standing assumption ($\mathbf{M}$ invertible, plus the row condition).
- What do `*` and `star` mean on `KM M`? Type-class **instances**, defined elsewhere:

```
instance : Mul (KM M) :=
  ⟨fun a b => ⟨fun i => (starMC M (toC a.vec) (toC b.vec) i).re⟩⟩
```

Real part! It equals $*_{\mathbf{M}}$ only because the hypothesis makes the product real (Lemma 2).

Verifying Statements Requires Knowing Lean

Proposition 6 of the paper ($\mathbb{K}_{\mathbf{M}}$ is a $*$-ring), as stated in Take 2, where `KM M` is $\mathbb{K}_{\mathbf{M}}$:

```
theorem prop6 (M : Matrix (Fin n) (Fin n) ℂ) [Fact (Admissible M)] :
    (∀ a : KM M, star (star a) = a) ∧
    (∀ a b : KM M, star (a + b) = star a + star b) ∧
    (∀ a b : KM M, star (a * b) = star b * star a) ∧
    star (1 : KM M) = 1 := by
  ...
```

- `[Fact (Admissible M)]` is a **type-class** argument. Here it is a hypothesis: the paper's standing assumption ($\mathbf{M}$ invertible, plus the row condition).
- What do `*` and `star` mean on `KM M`? Type-class **instances**, defined elsewhere:

```
instance : Mul (KM M) :=
  ⟨fun a b => ⟨fun i => (starMC M (toC a.vec) (toC b.vec) i).re⟩⟩
```

Real part! It equals $*_{\mathbf{M}}$ only because the hypothesis makes the product real (Lemma 2).

Checking a statement requires familiarity with Lean concepts, such as type classes.

Looking Forward: A Proposal for Our Community

- My view is that mathematics and computer science should adopt a standard: **each paper has an accompanying Lean formalization**. At the very least, this is the standard I plan to uphold in my future papers.

Looking Forward: A Proposal for Our Community

- My view is that mathematics and computer science should adopt a standard: **each paper has an accompanying Lean formalization**. At the very least, this is the standard I plan to uphold in my future papers.
- We should have a **firm library of infrastructure for NLA**: like Mathlib, but focused on what we want and need in NLA.
  - Perhaps more **concrete** than Mathlib (which is very abstract): e.g., matrices over $\mathbb{R}$ and $\mathbb{C}$ rather than general rings or fields.

Looking Forward: A Proposal for Our Community

- My view is that mathematics and computer science should adopt a standard: **each paper has an accompanying Lean formalization**. At the very least, this is the standard I plan to uphold in my future papers.
- We should have a **firm library of infrastructure for NLA**: like Mathlib, but focused on what we want and need in NLA.
  - Perhaps more **concrete** than Mathlib (which is very abstract): e.g., matrices over $\mathbb{R}$ and $\mathbb{C}$ rather than general rings or fields.
- **NLA-lib** by Tyler Chen: looks very exciting and promising.
  - Concrete by design: scalars in $\mathbb{R}$ or $\mathbb{C}$.
  - Take 2 was done without NLA-lib. Using it does make sense: it would have saved about 10% of the lines.

nla-lib.org

Conclusions

- Lean + AI makes formalizing NLA papers practical: AI writes the Lean, Lean checks the proofs, and I check the statements.
- Progress is fast: $\approx$ 2 weeks with babysitting (Feb.–Mar. 2026) $\rightarrow$ under 2 hours unattended (Oct. 2026).
- Formalization uncovers typos and missing assumptions, even after two revisions.
- But it is not “formalize and forget”: check the statements, the completeness, and the axioms.
- Proposal: papers with accompanying Lean formalizations, and a firm NLA library (e.g., NLA-lib).

Conclusions

Formalizations discussed in this talk (public):

Take 1: *Demystifying Tubal Tensor Algebra* (with U. Mor).
`github.com/haimav/formalize-tubal-demyst-paper`



Take 2: the same paper, redone with Claude Opus 5.5.
`github.com/haimav/demyst-formalization-take2`



Lecture notes: *Tensor Computations and Analyzing High Dimensional Data* (TAU).
`github.com/haimav/TensorsCourseFormalization`



NLA-lib: a Lean 4 library for numerical linear algebra (T. Chen).
`nla-lib.org`

