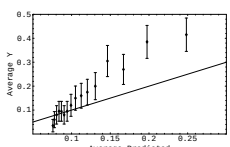
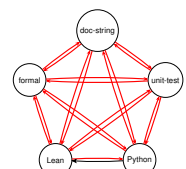
Modeling LLM statements and dialogs

Dean Foster

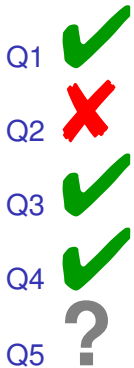
Amazon & U Penn

June 19, 2026





How often is an LLM correct?



Easy case

If all the questions are from GSM8K it is easy

Hard case

- Which is the best Lift, Waydoo or Fliteboard?
- My hand falls asleep when I'm biking, what is going on?
- Does M-convexity imply submodularity?
- . . .

Hard case

- Which is the best Lift, Waydoo or Fliteboard?
- My hand falls asleep when I'm biking, what is going on?
- Does M-convexity imply submodularity?
- . . .

What does accuracy even mean?

Hard case

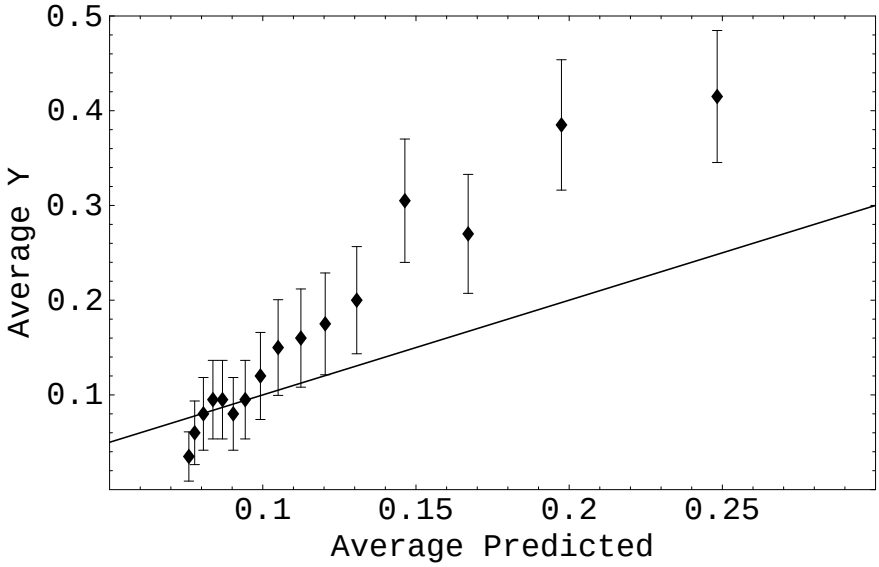
- Which is the best Lift, Waydoo or Fliteboard?
- My hand falls asleep when I'm biking, what is going on?
- Does M-convexity imply submodularity?
- . . .

What does accuracy even mean?
I'll argue "calibration" is the right interpretation.

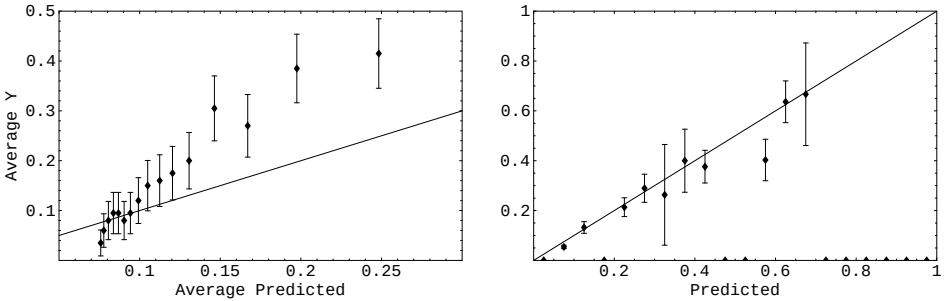
Calibration

- I been working on calibration since **1988**.
- Back then it was a niche game-theory question.
- Today it's the right way to answer "How often is an LLM correct?"

This is not calibrated



Anything easily fixed isn't calibrated



Fix the obvious problems!

Calibration is achievable

Theorem

A calibrated forecast exists.

Calibration is achievable

Theorem

A calibrated forecast exists.

PROOF:

Apply the minimax theorem. QED

Calibration is achievable

Theorem

A calibrated forecast exists.

PROOF:

Apply the minimax theorem. QED
(We'll give a different proof later, via online regression.)

Calibration limits LLM mis-alignment

Calibration is a low bar, so one all systems should achieve.

Reducing calibration to no-regret

- Many results in no-regret learning theory say we can make decisions in real time as well as we could in hindsight.
- We'll consider the least squares case.

Crazy calibration variable

Y	X_1	X_2	X_3	X_4
Y_1	X_{11}	X_{12}	X_{13}	X_{14}
Y_2	X_{21}	X_{22}	X_{23}	X_{24}
Y_3	X_{31}	X_{32}	X_{33}	X_{34}
Y_4	X_{41}	X_{42}	X_{43}	X_{44}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
Y_t	X_{t1}	X_{t2}	X_{t3}	X_{t4}

Starting with our data that we observed up to time t

Crazy calibration variable

Y	X_1	X_2	X_3	X_4
Y_1	X_{11}	X_{12}	X_{13}	X_{14}
Y_2	X_{21}	X_{22}	X_{23}	X_{24}
Y_3	X_{31}	X_{32}	X_{33}	X_{34}
Y_4	X_{41}	X_{42}	X_{43}	X_{44}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
Y_t	X_{t1}	X_{t2}	X_{t3}	X_{t4}

$$\hat{\beta}_t = \arg\min_\beta \sum_{i=1}^{t} (Y_i - \beta' X_i)^2$$

We can fit $\hat{\beta}_t$ on everything up to time t

Crazy calibration variable

Y	X_1	X_2	X_3	X_4
Y_1	X_{11}	X_{12}	X_{13}	X_{14}
Y_2	X_{21}	X_{22}	X_{23}	X_{24}
Y_3	X_{31}	X_{32}	X_{33}	X_{34}
Y_4	X_{41}	X_{42}	X_{43}	X_{44}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
Y_t	X_{t1}	X_{t2}	X_{t3}	X_{t4}
	$X_{t+1,1}$	$X_{t+1,2}$	$X_{t+1,3}$	$X_{t+1,4}$

$\hat{\beta}_t$

$$\hat{Y}_{t+1} = \hat{\beta}_t' X_{t+1}$$

From a new X_{t+1} we can compute $\hat{Y}_{t+1}$

Crazy calibration variable

Y	X_1	X_2	X_3	X_4	$\hat{\beta}$
Y_1	X_{11}	X_{12}	X_{13}	X_{14}	0
Y_2	X_{21}	X_{22}	X_{23}	X_{24}	$\hat{\beta}_1$
Y_3	X_{31}	X_{32}	X_{33}	X_{34}	$\hat{\beta}_2$
Y_4	X_{41}	X_{42}	X_{43}	X_{44}	$\hat{\beta}_3$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
Y_t	X_{t1}	X_{t2}	X_{t3}	X_{t4}	$\hat{\beta}_{t-1}$

Looking at only the first part of the data, we can generate:

$$\hat{\beta}_0, \quad \hat{\beta}_1, \quad \hat{\beta}_2, \quad \hat{\beta}_3, \quad \hat{\beta}_4, \quad \ldots, \quad \hat{\beta}_{t-1}$$

Crazy calibration variable

Y	X_1	X_2	X_3	X_4	$\hat{\beta}$	$\hat{Y}$
Y_1	X_{11}	X_{12}	X_{13}	X_{14}	0	$\hat{Y}_1 = 0$
Y_2	X_{21}	X_{22}	X_{23}	X_{24}	$\hat{\beta}_1$	$\hat{Y}_2 = \hat{\beta}_1' X_2$
Y_3	X_{31}	X_{32}	X_{33}	X_{34}	$\hat{\beta}_2$	$\hat{Y}_3 = \hat{\beta}_2' X_3$
Y_4	X_{41}	X_{42}	X_{43}	X_{44}	$\hat{\beta}_3$	$\hat{Y}_4 = \hat{\beta}_3' X_4$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
Y_t	X_{t1}	X_{t2}	X_{t3}	X_{t4}	$\hat{\beta}_{t-1}$	$\hat{Y}_t = \hat{\beta}_{t-1}' X_t$

Each of these leads to a next round

$$\hat{Y}_1, \quad \hat{Y}_2, \quad \hat{Y}_3, \quad \hat{Y}_4, \quad \ldots, \quad \hat{Y}_t$$

Crazy calibration variable

Y	X_1	X_2	X_3	X_4	$\hat{\beta}$	$\hat{Y}$
Y_1	X_{11}	X_{12}	X_{13}	X_{14}	0	$\hat{Y}_1 = 0$
Y_2	X_{21}	X_{22}	X_{23}	X_{24}	$\hat{\beta}_1$	$\hat{Y}_2 = \hat{\beta}_1' X_2$
Y_3	X_{31}	X_{32}	X_{33}	X_{34}	$\hat{\beta}_2$	$\hat{Y}_3 = \hat{\beta}_2' X_3$
Y_4	X_{41}	X_{42}	X_{43}	X_{44}	$\hat{\beta}_3$	$\hat{Y}_4 = \hat{\beta}_3' X_4$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
Y_t	X_{t1}	X_{t2}	X_{t3}	X_{t4}	$\hat{\beta}_{t-1}$	$\hat{Y}_t = \hat{\beta}_{t-1}' X_t$

Theorem (Foster 1991, Forster 1999)

Such an on-line least squares forecast generates low regret:

$$\sum_{t=1}^{T} (Y_t - \hat{Y}_t)^2 - \min_{\beta} \sum_{t=1}^{T} (Y_t - \beta' X_t)^2 \leq O(\log(T))$$

Crazy calibration variable

Y	X_1	X_2	X_3	X_4	$\hat{\beta}$	$\hat{Y}$
Y_1	X_{11}	X_{12}	X_{13}	X_{14}	0	$\hat{Y}_1 = 0$
Y_2	X_{21}	X_{22}	X_{23}	X_{24}	$\hat{\beta}_1$	$\hat{Y}_2 = \hat{\beta}_1' X_2$
Y_3	X_{31}	X_{32}	X_{33}	X_{34}	$\hat{\beta}_2$	$\hat{Y}_3 = \hat{\beta}_2' X_3$
Y_4	X_{41}	X_{42}	X_{43}	X_{44}	$\hat{\beta}_3$	$\hat{Y}_4 = \hat{\beta}_3' X_4$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
Y_t	X_{t1}	X_{t2}	X_{t3}	X_{t4}	$\hat{\beta}_{t-1}$	$\hat{Y}_t = \hat{\beta}_{t-1}' X_t$

Works no matter what the X's are.

Crazy calibration variable

Y	X_1	X_2	X_3	X_4	$\hat{\beta}$	$\hat{Y}$
Y_1	X_{11}	X_{12}	$\hat{Y}_1$	X_{14}	0	$\hat{Y}_1 = 0$
Y_2	X_{21}	X_{22}	$\hat{Y}_2$	X_{24}	$\hat{\beta}_1$	$\hat{Y}_2 = \hat{\beta}_1' X_2$
Y_3	X_{31}	X_{32}	$\hat{Y}_3$	X_{34}	$\hat{\beta}_2$	$\hat{Y}_3 = \hat{\beta}_2' X_3$
Y_4	X_{41}	X_{42}	$\hat{Y}_4$	X_{44}	$\hat{\beta}_3$	$\hat{Y}_4 = \hat{\beta}_3' X_4$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
Y_t	X_{t1}	X_{t2}	$\hat{Y}_t$	X_{t4}	$\hat{\beta}_{t-1}$	$\hat{Y}_t = \hat{\beta}_{t-1}' X_t$

Even if one of the X's were $\hat{Y}$!

Crazy calibration variable

Y	X_1	X_2	X_3	X_4	$\hat{\beta}$	$\hat{Y}$
Y_1	X_{11}	X_{12}	$\hat{Y}_1$	X_{14}	0	$\hat{Y}_1 = 0$
Y_2	X_{21}	X_{22}	$\hat{Y}_2$	X_{24}	$\hat{\beta}_1$	$\hat{Y}_2 = \hat{\beta}_1' X_2$
Y_3	X_{31}	X_{32}	$\hat{Y}_3$	X_{34}	$\hat{\beta}_2$	$\hat{Y}_3 = \hat{\beta}_2' X_3$
Y_4	X_{41}	X_{42}	$\hat{Y}_4$	X_{44}	$\hat{\beta}_3$	$\hat{Y}_4 = \hat{\beta}_3' X_4$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
Y_t	X_{t1}	X_{t2}	$\hat{Y}_t$	X_{t4}	$\hat{\beta}_{t-1}$	$\hat{Y}_t = \hat{\beta}_{t-1}' X_t$

Theorem ($\Longrightarrow$ Foster and Kakade 2008, Foster and Hart 2018)

Adding the crazy calibration variable gives multi-calibration:

$$(\forall i) \quad \sum_{t=1}^{T} X_{t,i}(Y_t - \hat{Y}_t) = O(\sqrt{T \log(T)})$$

Progressive con (Sometimes called foot-in-the-door)

- $2 + 2 = 4$ ✔
- The sky is blue. ✔
- Another trivial statement. ✔
- . . . ✔
- . . . ✔
- . . . ✔ . . . ✔ . . . ✔

Progressive con (Sometimes called foot-in-the-door)

- 2 + 2 = 4 ✔
- The sky is blue. ✔
- Another trivial statement. ✔
- . . . ✔
- . . . ✔
- . . . ✔ . . . ✔ . . . ✔
- LLMs will not take your job.

Progressive con (Sometimes called foot-in-the-door)

- $2 + 2 = 4$ ✔
- The sky is blue. ✔
- Another trivial statement. ✔
- . . . ✔
- . . . ✔
- . . . ✔ . . . ✔ . . . ✔
- LLMs will not take your job. ✘
  - Calibration says “Yes”
  - Multi-calibration says “no”

Multi-calibration is the protection we need

If we use multi-calibration, we are calibrated on:

- mathematics ✔
- history ✔
- factual questions ✔
- theorems ✔
- trivial questions ✔
- meaningful questions ✔

Multi-calibration is the protection we need

If we use multi-calibration, we are calibrated on:

- mathematics ✔
- history ✔
- factual questions ✔
- theorems ✔
- trivial questions ✔
- meaningful questions ✔

(But not necessarily on historical$\times$factual$\times$trivial questions)

. . . and it is what lets an agent act

An agent acts on many *surfaces*:

- `bash`, `git`, file edits, the web, . . .
- different projects, different scales

. . . and it is what lets an agent act

An agent acts on many *surfaces*:

- `bash`, `git`, file edits, the web, . . .
- different projects, different scales

We want it calibrated on *each one separately*:

- being reliable at `git` must *not* inflate its confidence on `bash`.
- no borrowing strength across surfaces.

. . . and it is what lets an agent act

An agent acts on many *surfaces*:

- `bash`, `git`, file edits, the web, . . .
- different projects, different scales

We want it calibrated on *each one separately*:

- being reliable at `git` must *not* inflate its confidence on `bash`.
- no borrowing strength across surfaces.

Multi-calibration is exactly that guarantee — so we can let the agent loose and still trust each kind of action.

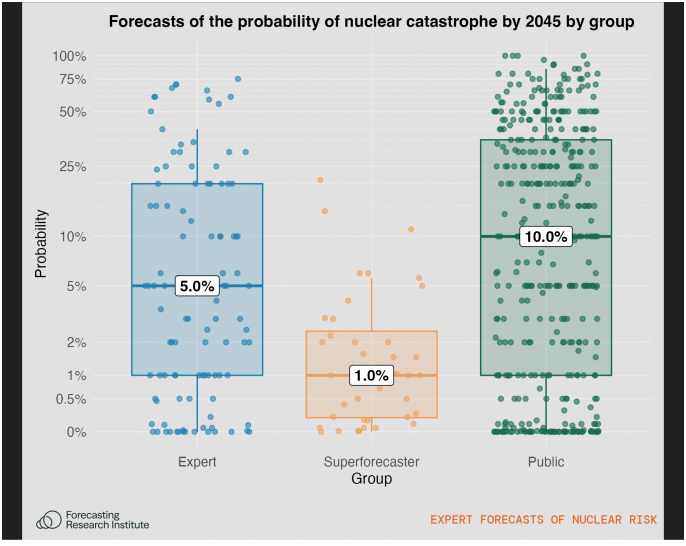
Feeding LLMs back and forth

- Feed one LLM's forecast into another.
- This generates a "dialog."
- Hart and I call this calibeating
- Each round the forecasts get better and better (Collina et al).
- Is there a limit?

Agreeing to disagree

- This back and forth converges:
  - Theorem by Bob Aumann (Nobel prize, “Agreeing to disagree.”)
  - Scott Aaronson did it for computers in 2001
  - LLMs proved it last week using Lean!
- Claim: We all agree at the end

We all agreement in the end: or maybe not



A typical dialog

- These aren't very interesting conversations:
  - He said: .9
  - She said: .7
  - He said: .7
  - She said: .6
  - ...
  - both say: .55
- Let's turn to more interesting dialogs

Calibration: how often it's right.

Calibration: how often it's right.

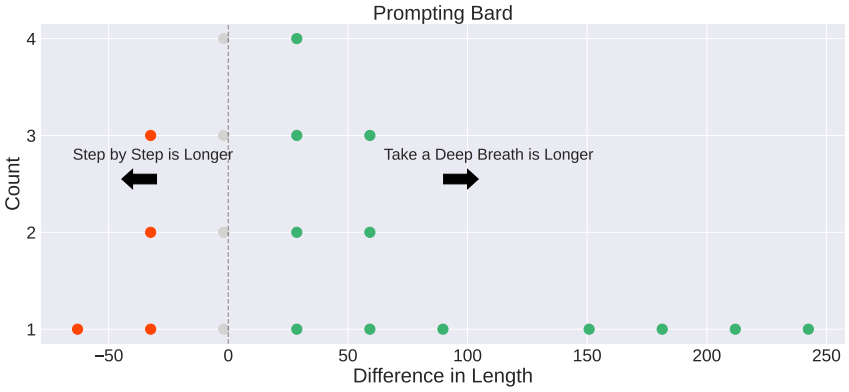
Thinking: How to make it right.

Calibration: how often it's right.

Thinking: How to make it right.

Checking: Is *this* answer right?

LLMs do better with more thinking



Contrasting native LLMs vs Chain of Thought

Theorem (Merrill and Sabharwal 2023)

(rephrased) An LLM can not answer questions in PSPACE. [arXiv]

Theorem (F. and Madeka 2023)

Using chain of thought reasoning, an LLM can solve any problem in PSPACE.

[slides]

Contrasting native LLMs vs Chain of Thought

Theorem (Merrill and Sabharwal 2023)

(rephrased) An LLM can not answer questions in PSPACE. [arXiv]

Theorem (F. and Madeka 2023)

Using chain of thought reasoning, an LLM can solve any problem in PSPACE. [slides]

Other versions:

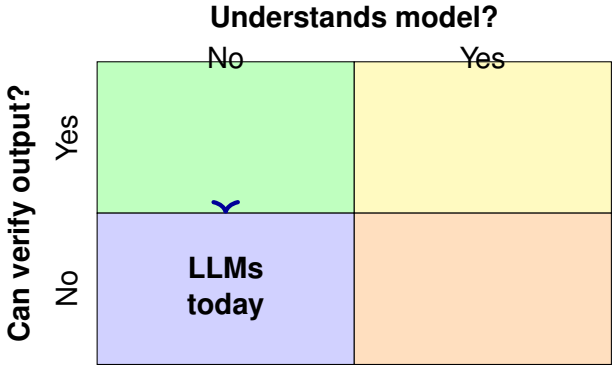
Theorem (Malach 2023)

A linear LLM can be trained to mimic a Turing machine using chain-of-thought. [arXiv]

Theorem (Giannou, Rajput, Sohn, Lee, Lee, and Papailiopoulos 2023)

Looped Transformers are general computers. [arXiv]

We can't open the box — but we can challenge it



Interactive proofs move LLMs *up*: verifiable without being transparent.

IP: a better kind of dialog

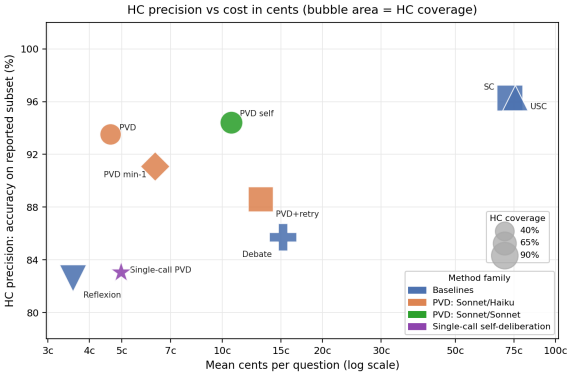
- “He said .9, she said .7” converges, but isn’t informative.
- Interactive proofs (Goldwasser–Micali–Rackoff): a bounded *verifier* interrogates a powerful *prover*.
- Model the human as a poly-time verifier, the LLM as a (near) exponential-time prover.

IP: a better kind of dialog

- “He said .9, she said .7” converges, but isn’t informative.
- Interactive proofs (Goldwasser–Micali–Rackoff): a bounded *verifier* interrogates a powerful *prover*.
- Model the human as a poly-time verifier, the LLM as a (near) exponential-time prover.
- The deliberation history *is* the proof: each round checks a partial claim.
- We don’t need a “faithful” explanation — we need *honesty*: claims stay consistent across rounds.
- Zero-knowledge: the model’s billions of parameters (the witness) are never revealed.

Prover–Verifier Deliberation

- Prover defends an answer through checkable sub-claims; verifier returns ACCEPT, CHALLENGE, or REJECT.
- "Accept + No Change" = the high-confidence subset: report it, abstain otherwise.
- ∼30pp precision gap between the confident and unsure answers.



A real IP dialog: “is this code leak-free?”

Claim: this repository has no memory leaks.

Verifier: check Package 1 — any leak source?

A real IP dialog: “is this code leak-free?”

Claim: this repository has no memory leaks.

Verifier: check Package 1 — any leak source?

Prover: Function A’s alloc/free are matched; Function B uses RAII; here is the path . . .

A real IP dialog: “is this code leak-free?”

Claim: this repository has no memory leaks.

Verifier: check Package 1 — any leak source?

Prover: Function A’s alloc/free are matched; Function B uses RAII; here is the path . . .

Verifier: show me the *allocation site and the deallocation site* for Function A.

A real IP dialog: “is this code leak-free?”

Claim: this repository has no memory leaks.

Verifier: check Package 1 — any leak source?

Prover: Function A’s alloc/free are matched; Function B uses RAII; here is the path . . .

Verifier: show me the *allocation site and the deallocation site* for Function A.

Each round forces a *checkable artifact* — file, line, test, or counterexample. The process *is* the proof

It knows when it can't be trusted

- On GPQA: ANC answers are 84–98% precise, $\sim$30pp above the rest — at $\sim$3 calls/question (vs. 8 for self-consistency).

It knows when it can't be trusted

- On GPQA: ANC answers are 84–98% precise, $\sim$30pp above the rest — at $\sim$3 calls/question (vs. 8 for self-consistency).
- Put the verifier *outside* its competence (Humanity's Last Exam, weak verifier): the gap *inverts* to $-$7pp.
  - It accepts exactly the questions it cannot evaluate.

It knows when it can't be trusted

- On GPQA: ANC answers are 84–98% precise, $\sim$30pp above the rest — at $\sim$3 calls/question (vs. 8 for self-consistency).
- Put the verifier *outside* its competence (Humanity's Last Exam, weak verifier): the gap *inverts* to -7pp.
  - It accepts exactly the questions it cannot evaluate.
- The collapse is itself the signal: we learn *when to stop trusting the protocol.* (Calibration, again.)

Alignment: IP and calibration

- P can check PSPACE
  - So if they outsmart us, we can still check them
- RP can be calibrated against anyone!
  - So no matter how far they outsmart us, we can still count and compute their calibration accuracy

What should LLM’s think about?

What should LLM’s think about?

Lean!

What you might have heard about *Lean* already

- Lean is used for formalizing mathematics:
  - Terry Tao is fascinated by Lean
  - He did a math project with 100s of people
  - They could all write Lean proofs
  - The proofs were “checked” by simply being type checked in Lean

What you might have heard about *Lean* already

- Lean is used for formalizing mathematics:
  - Terry Tao is fascinated by Lean
  - He did a math project with 100s of people
  - They could all write Lean proofs
  - The proofs were “checked” by simply being type checked in Lean
- The entire Cambridge undergraduate mathematics has been formalized in Lean
  - extensive library (mathlib) exists
- Formalization of latex to lean can be done
  - Formalization is a great LLM application

What does *Lean* math look like?

Here is "$\sqrt{2}$ is irrational" in Lean:

```
example {m n p : ℕ} (nnz : n ≠ 0) (prime_p : p.Prime) :
    m^2 ≠ p * n^2 := by
  intro sqr_eq
  have nsqr_nez : n^2 ≠ 0 := by simpa
  have eq1 : Nat.factorization (m^2) p
           = 2 * m.factorization p := by
  -- ...
```

What is *Lean*?

- Lean is a pure functional programming language — no side effects at all.
  - This makes it easy to prove theorems about the code you write.

What is *Lean*?

- Lean is a pure functional programming language — no side effects at all.
  - This makes it easy to prove theorems about the code you write.
- It is also logic, based on type theory:
  - A theorem is a *type*; a proof is a *term* of that type.
  - (Lambda calculus and Turing machines were mathematics before they were code.)

What does *Lean* code look like? (here is Quick Sort)

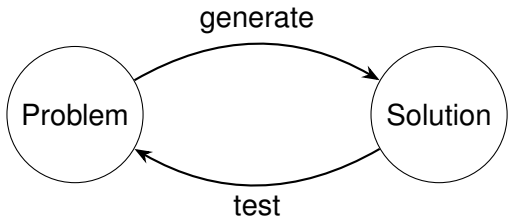
```
def qsort {α} (lt : α → α → Bool) : List α → List α
  | []     => []
  | h :: t =>
    let small := t.filter (fun x =>   lt x h)
    let large := t.filter (fun x => ! lt x h)
    qsort lt small ++ h :: qsort lt large
termination_by l => l.length
decreasing_by
  all_goals simp_wf
  all_goals
    apply Nat.lt_succ_of_le
    apply Nat.le_trans (List.length_filter_le _ _)
    simp
```

Writing Lean code is hard;

checking it is easy

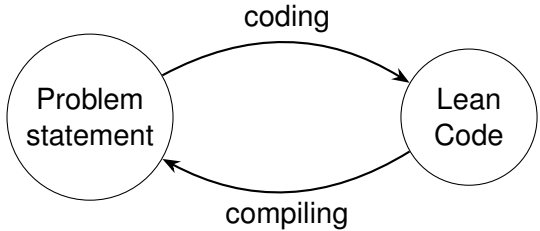
Generate and test: Mathematics

- CS:
  - Generation is easier than testing (P $\neq$ NP)
- mathematics:
  - Problem $\rightarrow$ solution: generation
  - Solution $\rightarrow$ problem: testing


generate
Problem
Solution
test

Generate and test

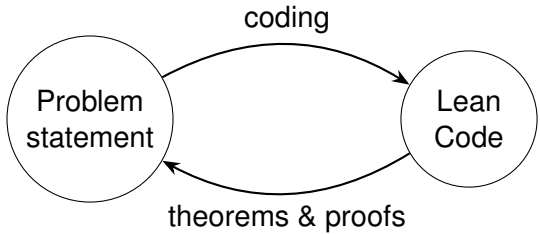
- Fits nicely into LLMs
  - Have one LLM generate a solution
  - Have a different one test the solution
  - Improve the generation
- We found disappointing results ([6])
  - Saturates after a few rounds
  - Effectively doubling the data size
  - Far from a polynomial – exponential split
  - Better models have a bigger gap, so the future might still be rosy for this approach


coding
Problem statement
Lean Code
compiling

Compiling only checks *termination*.

Compiling only checks *termination*.

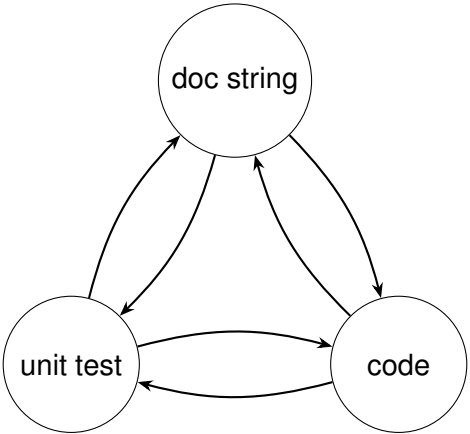
Type-checking $\neq$ correctness — that's the next arrow.


coding
Problem statement
Lean Code
theorems & proofs

Give LLMs problems they can more easily solve

- Have LLMs:
  - Write the goal of the program (aka doc string)
  - Write unit tests
  - Write code
- Have them check consistency between them
- No compiling necessary! All done by LLMs

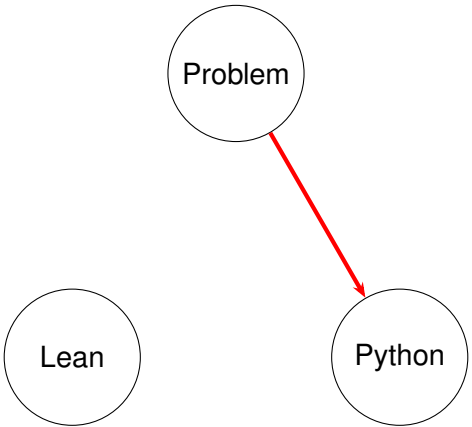
typical coding model



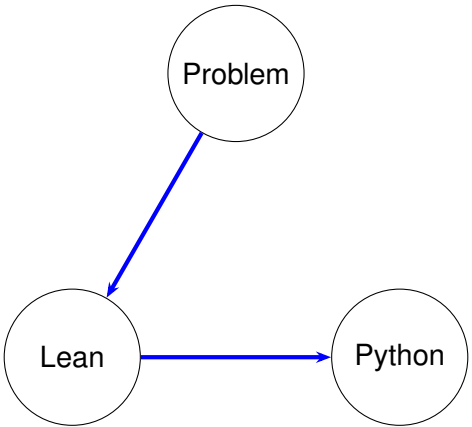
CLOVER: Ask an LLM interesting questions

- Do these unit tests match the doc string?
- Does this doc string summarize the code?
- . . .
- 6 questions in all
- Key idea in CLOVER by Sun, Sheng, Padon and Barrett.
  (They actually used formalization instead of unit tests)

Lean short cut



Lean short cut



I call this thinking in Lean

I call this thinking in Lean

But why should it help?

Sapir-Whorf Hypothesis

- My coauthors think in logic / math; I think in heuristics. We think differently.

Sapir-Whorf Hypothesis

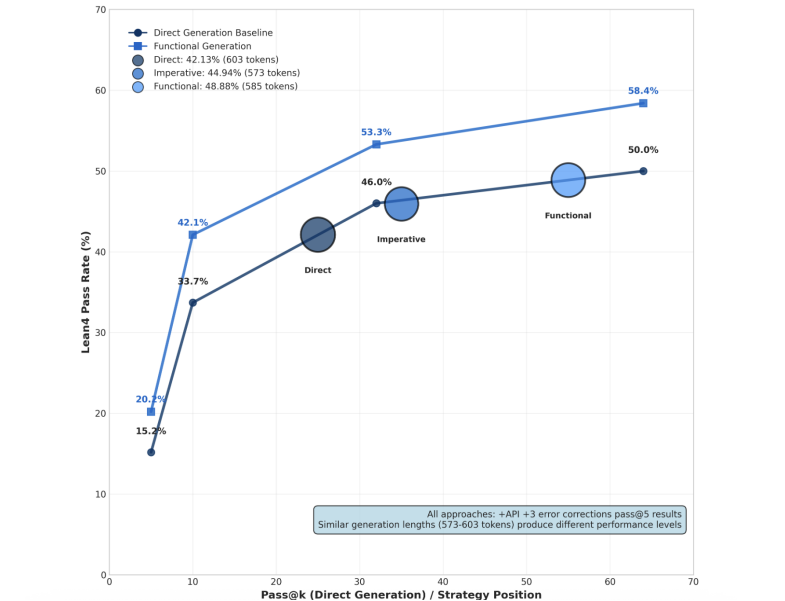
- My coauthors think in logic / math; I think in heuristics. We think differently.
- Sapir-Whorf: the language you use shapes how you think.
  - Famous (and wrong) example: “Eskimos have 200 words for snow” — considered discredited.

Sapir-Whorf Hypothesis

- My coauthors think in logic / math; I think in heuristics. We think differently.
- Sapir-Whorf: the language you use shapes how you think.
  - Famous (and wrong) example: “Eskimos have 200 words for snow” — considered discredited.
- Sapir-Whorf-*Foster*: English vs. Chinese doesn’t matter — but English vs. *logic* is a big change. So is imperative vs. functional.

Great SF on the idea: Delany’s Babel-17*, Elgin’s* Native Tongue*, Kagan’s* Hellspark*.*

What if we do chain-of-though in Lean?


Performance vs Generation Length: Multi-Step Reasoning Shows Similar Lengths, Different Result
Direct Generation Baseline
Functional Generation
Direct: 42.13% (603 tokens)
Imperative: 44.94% (573 tokens)
Functional: 48.88% (585 tokens)
58.4%
53.3%
50.0%
46.0%
42.1%
33.7%
20.2%
15.2%
Direct
Imperative
Functional
All approaches: +API +3 error corrections pass@5 results
Similar generation lengths (573-603 tokens) produce different performance levels
Lean4 Pass Rate (%)
Pass@k (Direct Generation) / Strategy Position
0
10
20
30
40
50
60
70

Thinking functionally wins — for every model

- *BRIDGE*: 178 problems, 5 LLMs.
- Functional reasoning vs. direct Python generation:
  - $\sim$1.5$\times$ higher Lean correctness (pass@5) ✔
  - up to 2$\times$ more sample-efficient
- Holds across Claude, DeepSeek, Llama, Qwen.

Thinking functionally wins — for every model

- *BRIDGE*: 178 problems, 5 LLMs.
- Functional reasoning vs. direct Python generation:
  - $\sim$1.5$\times$ higher Lean correctness (pass@5) ✔
  - up to 2$\times$ more sample-efficient
- Holds across Claude, DeepSeek, Llama, Qwen.
- And it's *learnable*: fine-tuning on functional reasoning traces beats code-only fine-tuning by $\sim$1.5$\times$.

Why functional helps

Functional prompting fixes exactly the errors Lean checks:

- Syntax errors: 45% $\rightarrow$ 12%
- Termination / totality failures: 15% $\rightarrow$ 8%
- Fewer type errors

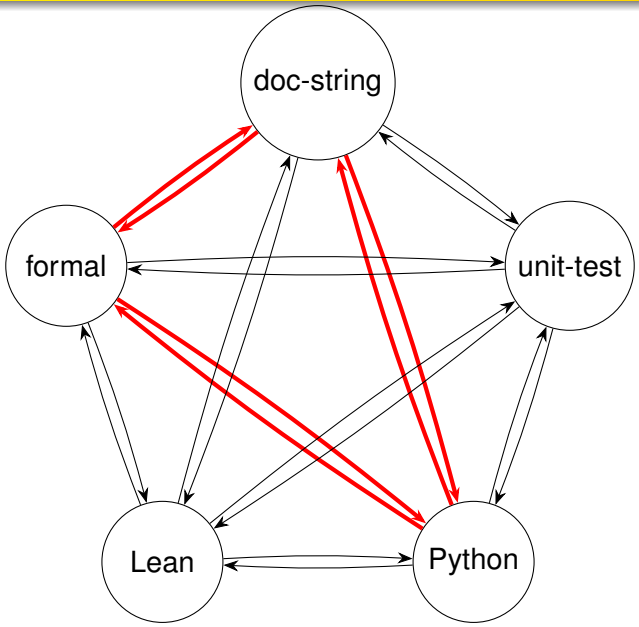
Why functional helps

Functional prompting fixes exactly the errors Lean checks:

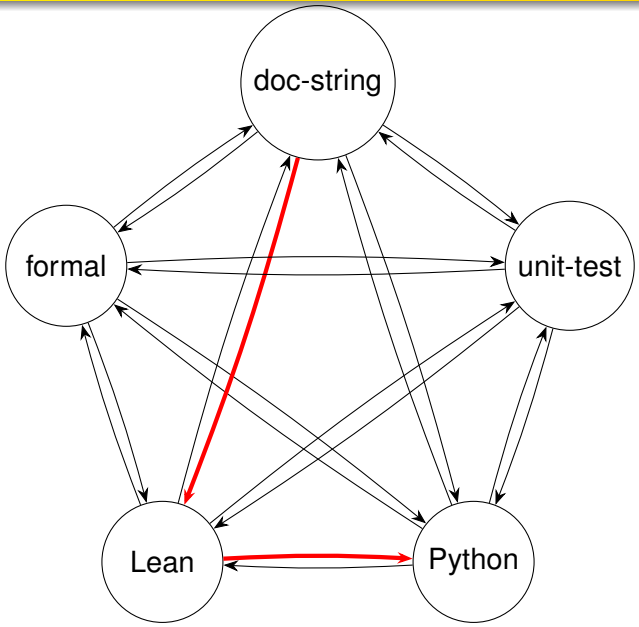
- Syntax errors: 45% $\rightarrow$ 12%
- Termination / totality failures: 15% $\rightarrow$ 8%
- Fewer type errors

Not just more tokens: a length-matched “Double Lean” control (same budget) still loses, 15.2% vs. 20.2%. The win is the *representation*, not the verbosity.

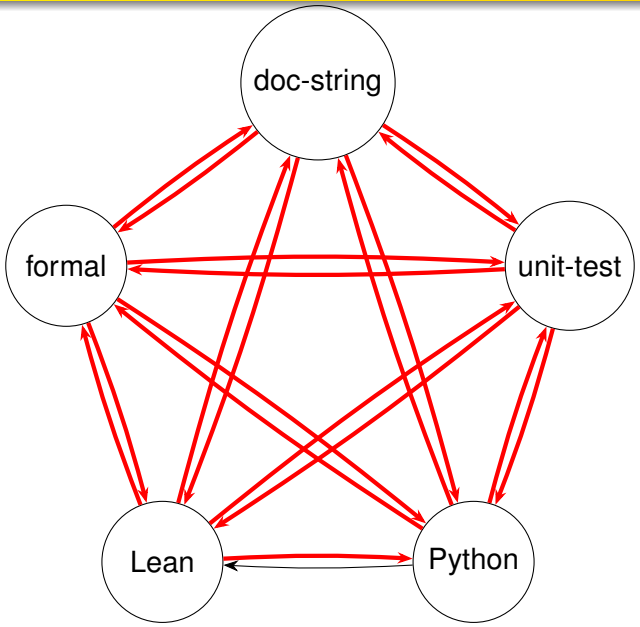
CLOVER: the starting point



Our work



The Dream: Putting them all together in one system



Manifest destiny: an agent that thinks in Lean

- Problem to solve? *Code.*
- What should it talk about? *Theorems.*
- Use Lean as the programming language:
  - so functional it won't call a function a "function" until you prove it isn't partial.

Manifest destiny: an agent that thinks in Lean

- Problem to solve? *Code.*
- What should it talk about? *Theorems.*
- Use Lean as the programming language:
  - so functional it won't call a function a "function" until you prove it isn't partial.
- So I built one: **l3m** — a Claude-CLI-shaped agent, written in Lean, cloned in two weeks.
- Its trusted computing base is the Lean kernel, not a prompt.

l3m: safety you can *prove*

In **tame** mode, these are kernel-checked theorems, not promises:

- ✔ Never writes outside its own directory.
- ✔ Leaks at most 1000 bits to the web per session.
- ✔ Every action it takes can be undone.
- ✔ High-entropy strings (API_KEYs) are redacted on input.

l3m: safety you can *prove*

In **tame** mode, these are kernel-checked theorems, not promises:

- ✔ Never writes outside its own directory.
- ✔ Leaks at most 1000 bits to the web per session.
- ✔ Every action it takes can be undone.
- ✔ High-entropy strings (API_KEYs) are redacted on input.

The trust report is a build artifact: every claim is a ProvenTheorem, a ManifestAxiom, or an explicit WorldClaim about the OS.
[manifests]

When `bash` is needed: back to IP

- Raw shell escapes the typed, provable surface — so the agent must *argue* for it.
- It runs an interactive proof (with a proxy verifier) for why `bash` is necessary.
  - Often it argues itself into a simpler, tame call instead.
  - If it can't decide, it throws the question to me.
  - I rarely have to answer.

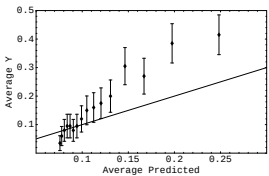
When `bash` is needed: back to IP

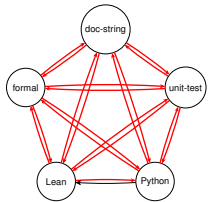
- Raw shell escapes the typed, provable surface — so the agent must *argue* for it.
- It runs an interactive proof (with a proxy verifier) for why `bash` is necessary.
  - Often it argues itself into a simpler, tame call instead.
  - If it can't decide, it throws the question to me.
  - I rarely have to answer.
- Calibration let it *know what it doesn't know*; IP let it *prove what it does*; Lean let it *think in logic*.
- Each surface — `bash`, `git`, web, every project and scale — stays *separately* calibrated (multi-calibration).
- I now live in tame mode.

Close LLM Collaborators:

- Joao Sedoc [NYU]: Human evaluation and IP
- Carson Eisenach [Amazon]: RLMF, Lean
- Robert Joseph [Cal Tech], Udaya Guha [Amazon]: Lean
- Yuda Song [CMU]: mathematics
- Dominique Perrault-Joncas [Amazon]: Computer and human eval
- Omer Gottesman [Amazon], Lyle Ungar [U Penn], Emma Brunskill [Stanford], Amy Zhang [UT]: Education

THANKS!





(slides are at deanfoster.net)

Citations (slides at deanfoster.net)

"Principal / agent theory for LLMs and alignment," —. [1]

"How Does Critical Batch Size Scale in Pre-training?" Zhang, Morwani, Vyas, Wu, Zou, Ghai, —, Kakade. [2]

"What is the Value of Human-Level AI to Education?," Madeka, —, Kakade. [3]

"Efficient RL for optimizing conversation level outcomes with an LLM-based tutor," Nam, et al. [4]

"A TCS look at LLMs." [5]

"Mind the Gap: Examining the Self-Improvement Capabilities of Large Language Models" Song, et al. [6]

"Clover: Closed-Loop Verifiable Code Generation," Sun, Sheng, Padon, Barrett. CLOVER

"BRIDGE: Building Representations In Domain Guided Program Synthesis," George, et al. arxiv.

"Trust but Verify: Prover–Verifier Deliberation for Selective LLM Prediction," Sedoc, Zhang, —. [7]

"Human–LLM Deliberation as an Interactive Zero-Knowledge Proof Protocol," Sedoc et al.

"Calibeating: Beating Forecasters at Their Own Game," — and Hart. [8]

"Tractable Agreement Protocols," Collina, Goel, Gupta, Roth. [arXiv]

"l3m: a verified coding agent in Lean," —. lean4.ai/l3m

"Lean Manifests: compiler-enforced evidence levels," —. lean4.ai/lean-manifests