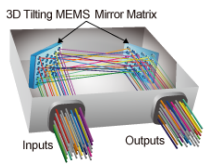
LLM code better when they think functionally

Dean Foster, Amazon

October 7, 2025



My background: Before LLMs

- Professor of Statistics
  - Regression: variable selection, “big data”
  - NLP: parsing, eigenwords before word2vec
  - Game theory: Calibration, fairness
- Joined Amazon 10 years ago to do forecasting
  - My goal was to avoid the NN craze of NLP

My background: Before LLMs

- Professor of Statistics
  - Regression: variable selection, “big data”
  - NLP: parsing, eigenwords before word2vec
  - Game theory: Calibration, fairness
- Joined Amazon 10 years ago to do forecasting
  - My goal was to avoid the NN craze of NLP
  - We started using NNs for forecasting 1 year later
  - We grew the NYC team to 30 scientists
- Started an RL team in NYC
  - Created systems to control buying, cross docks and placement

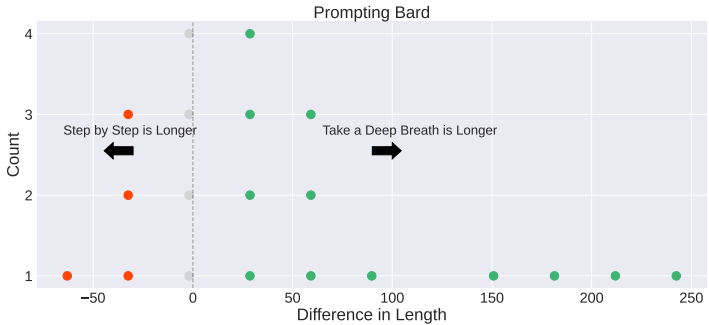
My background: Before LLMs

- Professor of Statistics
  - Regression: variable selection, “big data”
  - NLP: parsing, eigenwords before word2vec
  - Game theory: Calibration, fairness
- Joined Amazon 10 years ago to do forecasting
  - My goal was to avoid the NN craze of NLP
  - We started using NNs for forecasting 1 year later
  - We grew the NYC team to 30 scientists
- Started an RL team in NYC
  - Created systems to control buying, cross docks and placement
  - While the supply chain caught up with RL, my team started working on LLMs as a side project

My background: After LLMs

- What I've been doing for the past 2 years:
  - Engineering: fighting communication bottlenecks (LLMs require bandwidth)
  - Alignment: defending LLMs using game theory ([1])
  - Optimization: picking the batch size ([2])
  - Applications: tutoring children ([3], [4])
  - Theory: Chain of thought lifts LLMs from TC^0 to PSPACE ([5])
  - RL: generate and test for self improvement ([6])
- What I'll talk about today:
  - Automatic reasoning, programming and Lean
  - Lean: getting LLMs to think differently

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.

Contrasting native LLMs vs Chain of Thought

Theorem (Merrill and Sabharwal 2023)

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

Theorem (F. and Madeka 2023)

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

Contrasting native LLMs vs Chain of Thought

Theorem (Merrill and Sabharwal 2023)

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

Theorem (F. and Madeka 2023)

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

Other versions:

Theorem (Malach 2023)

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

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

Looped Transformers are general computers.

What should LLM’s think about?

What should LLM’s think about?

Lean!

What is *Lean*?

- 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 is *Lean*?

- 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 (But is hard)

What is *Lean*?

- Lean is a functional programming language:
  - Lean is a feature-complete functional programming language
  - It is a pure language: No side effects at all
  - This makes it easy to prove theorems about the code you write

What does is *Lean* math look like?

The square root of a prime is irrational with the start of its proof 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*?

- How is mathematics connected to programming?
  - “lambda calculus” was in mathematics before it was Lisp
  - Turing machines are mathematics and programs
  - Lean implements another form of logic based on type theory

What is *Lean*?

- How is mathematics connected to programming?
  - “lambda calculus” was in mathematics before it was Lisp
  - Turing machines are mathematics and programs
  - Lean implements another form of logic based on type theory
- A theorem in Lean is a type
- A proof in Lean is an example of that type

What *Lean* isn't:

Lean is not a theorem prover

- Everyone knows 3SAT can represent any logic problem
- But no one writes 3SAT to describe their problems
- SMT solvers are generic tool
- Lean is not an SMT
  - It is a proof assistant
  - Humans write the proofs, and Lean checks it

What *Lean* isn't:

Lean is not axiomatic mathematics

- The axioms need to be added (called mathlib)
- So Lean is very small and has been proven to be a correct engine
- Lean doesn't make errors–axioms might!

Amazon's connection to *Lean*

- We have the Leo de Moura (the creator of Lean) on our Automatic reasoning team
- But we have 100 other scientist doing automatic reasoning (largest in the world)
  - They find bugs in hardware
  - They Find security leaks
  - They create provable correct translations crypto code
  - They create the trust that AWS users require

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

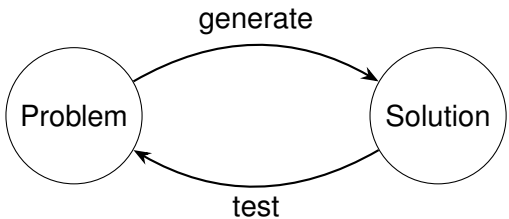
```
def qsort.F {α} (lt : α → α → bool) : Π (x : list α),
  (Π (y : list α), length y < length x → list α) → list α
| []     IH := []
| (h::t) IH := begin
    induction e : partition (λ x, lt h x = tt) t with large small,
    have : length small < length (h::t) ∧ length large < length (h::t),
    { rw partition_eq_filter_filter at e,
      injection e,
      subst large, subst small,
      constructor;
      exact nat.succ_le_succ (length_le_of_sublist (filter_sublist _)) },
    exact IH small this.left ++ h :: IH large this.right
  end
```

Writing lean code is hard;

checking it is easy

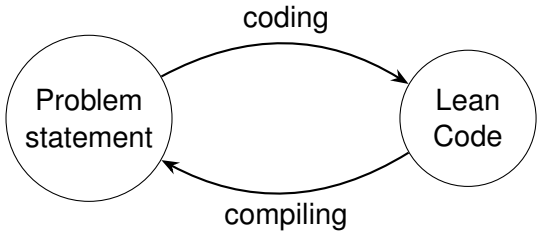
Generate and test

- Classic idea:
  - Generation is easier than testing
  - The gap can be HUGE:
    - Suppose NP $\approx$ EXP
    - Generation is takes exponential time
    - Testing takes polynomial times
- Nicely fits into mathematics
  - Problem to solution is generation
  - Solution to problem is testing

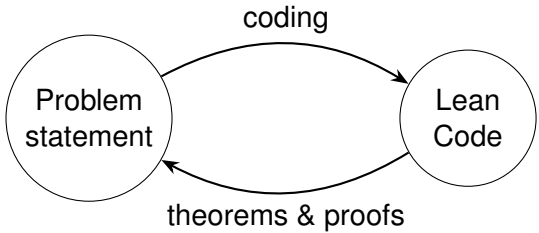

generate
Problem
Solution
test

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

Only checks for termination.

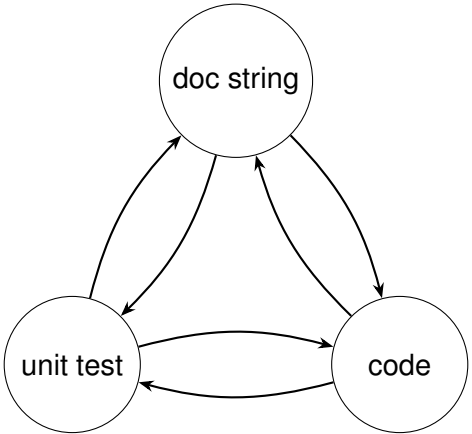

coding
Problem statement
Lean Code
theorems & proofs

Both directions require LLMs.

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

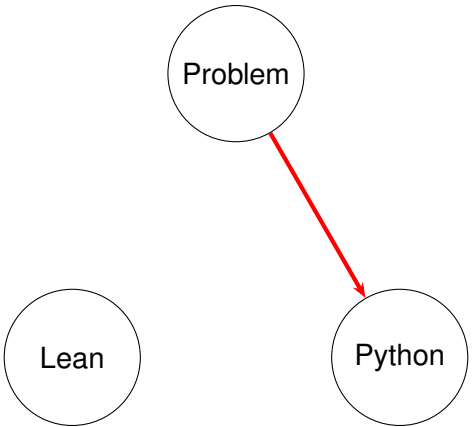
typical coding model



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



I call this thinking in Lean

I call this thinking in Lean

But why should it help?

Sapir-Whorf Hypothesis

- Some of my coauthors think in Mathematics

Sapir-Whorf Hypothesis

- Some of my coauthors think in Mathematics
- I don't!
  - I think in heuristics
  - I treat mathematics like an empirical science
- We think differently

Sapir-Whorf Hypothesis

- Sapir-Whorf hypothesis says that the language we use influences the way we think.
  - Example claim by Frank Boas: “Eskimos have 200 words for snow so they understand it better.”

Sapir-Whorf Hypothesis

- Sapir-Whorf hypothesis says that the language we use influences the way we think.
  - Example claim by Frank Boas: “Eskimos have 200 words for snow so they understand it better.”
  - This is wrong in at least 4 ways (not Eskimos, inaccurate quote, 10 not 200, no change in thinking)
- Generally considered discredited

Sapir-Whorf Hypothesis

- Sapir-Whorf hypothesis says that the language we use influences the way we think.
  - Example claim by Frank Boas: “Eskimos have 200 words for snow so they understand it better.”
  - This is wrong in at least 4 ways (not Eskimos, inaccurate quote, 10 not 200, no change in thinking)
- Generally considered discredited
- Great SF though:
  - Early: Samuel R. Delany’s *Babel 17*
  - Hopeful: Suzette Haden Elgin’s *Native Tongue*
  - Less extreme: Janet Kagan’s *Hellspark*

Sapir-Whorf Hypothesis

- Changing English to Chinese doesn't matter

Sapir-Whorf Hypothesis

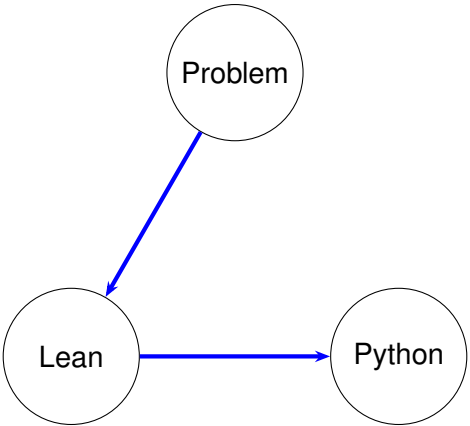
- Changing English to Chinese doesn't matter
- Changing English to Logic is a big change

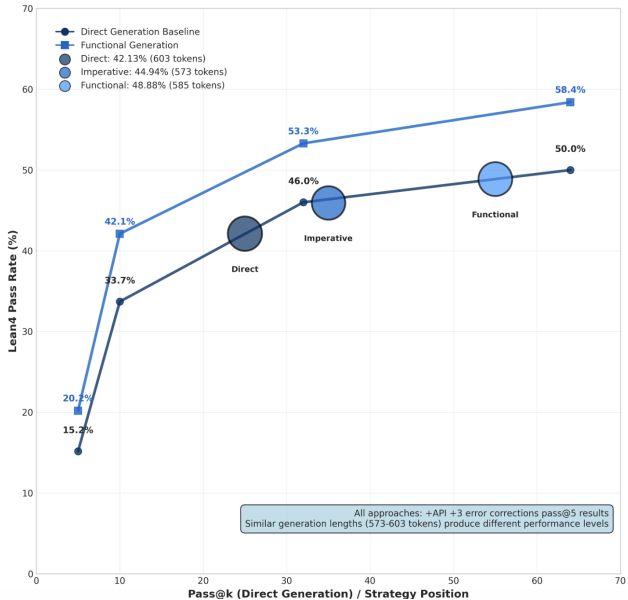
Sapir-Whorf Hypothesis

- Changing English to Chinese doesn't matter
- Changing English to Logic is a big change
- Changing imperative coding to functional coding is a big change

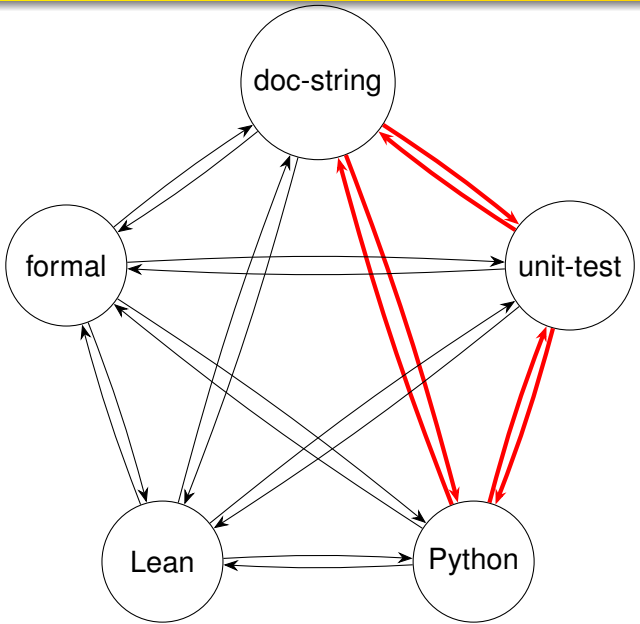
What if we do chain-of-though in Lean?

Lean short cut

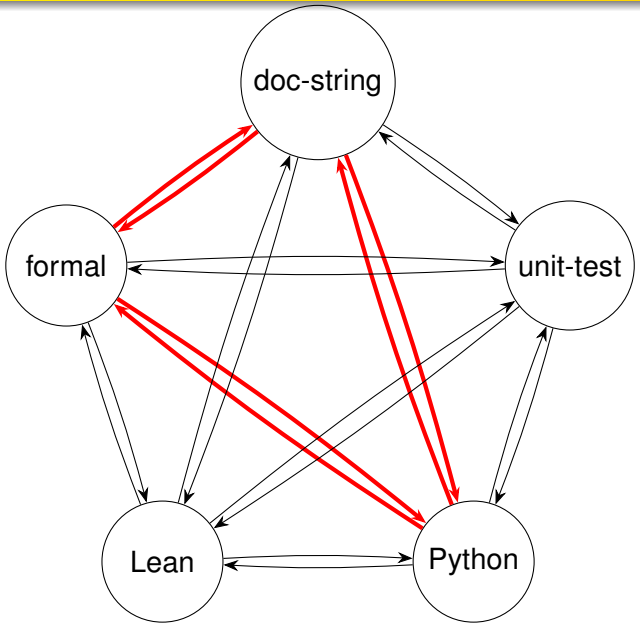



Performance vs Generation Length: Multi-Step Reasoning Shows Similar Lengths, Different Result
Direct Generation Baseline
Functional Generation
Direct: 42.13% (603 tokens)
Imperative: 44.94% (573 tokens)
Functional: 48.88% (585 tokens)
58.4%
53.3%
50.0%
46.0%
42.1%
33.7%
20.2%
15.2%
Direct
Imperative
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

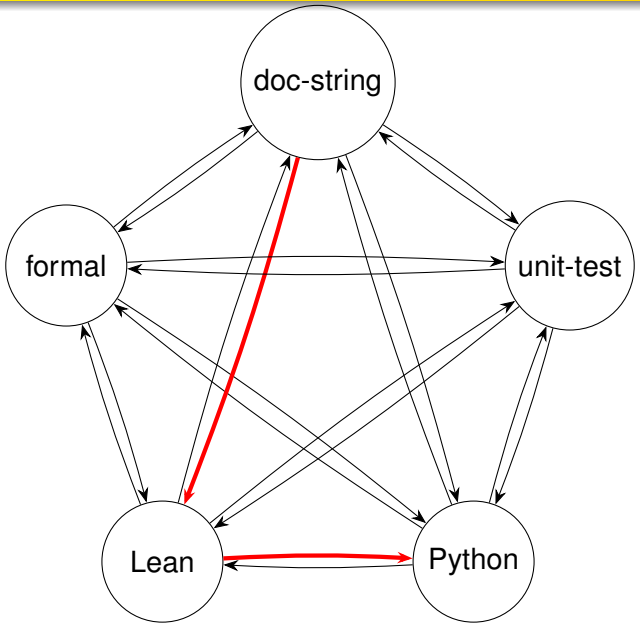
Starting point



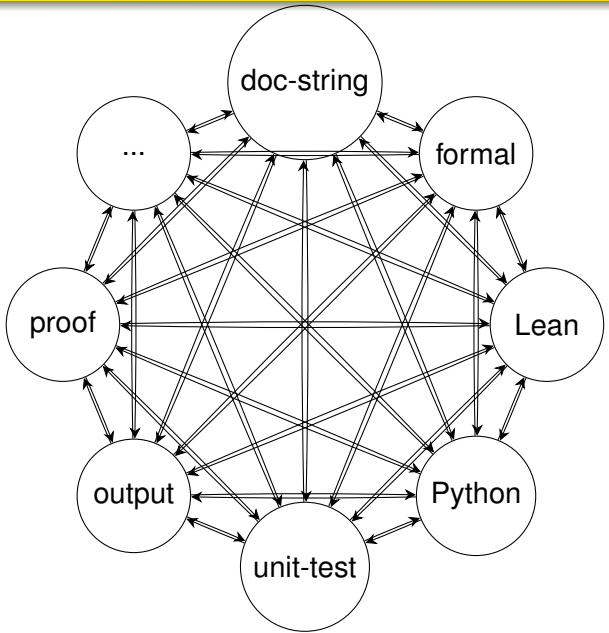
CLOVER



Our work



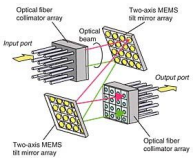
The dream



Close LLM Colaborators:

- Robert Joseph [intern and Cal Tech] Lean
- Carson Eisenach [NYC] RLMF, Lean
- Udaya Guha [NYC $\rightarrow$ AWS] Lean
- Dhruv Madeka [NYC $\rightarrow$ GDM] communication
- Omer Gottesman [NYC] education
- Riccardo Savorgnan [NYC $\rightarrow$ NYU] Lean
- Sham Kakade [Amazon Scholar and Harvard] Batch size and education
- Alex (Hyunji) Nam [intern and Stanford] Education
- Yuda Song [intern and CMU] mathematics
- Dominique Perrault-Joncas [Seattle] Computer and human eval
- Kari Torkkola [Seattle] Fine tuning
- Joao Sedoc [NYU] Human evaluation and theory (see him Friday!)
- Lyle Ungar [U Penn], Emma Brunskill [Stanford], Amy Zhang [UT]: Education
- . . .

THANKS!


Optical fiber collimator array
Two-axis MEMS tilt mirror array
Optical beam
Input port
Output port
Two-axis MEMS tilt mirror array
Optical fiber collimator array

Citations (slides at deanfoster.net and amazon.science)

"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, Gottesman, Zhang, —, Brunskill, Ungar. ([4])

"A TCS look at LLMs," — at MTI-LLM. ([5])

"Mind the Gap: Examining the Self-Improvement Capabilities of Large Language Models Song, Zhang, Eisenach, Kakade, —, Ghai. ([6])

"Clover: Closed-Loop Verifiable Code Generation," Chuyue Sun, Ying Sheng, Oded Padon, Clark Barrett. (CLOVER)

"Progressive Formalization: A Multi-Representation Framework for Automated Verificatio Ceisen, George, —. Today's talk.