

Evaluating Program Semantics Reasoning with Type Inference in System F

Yifeng He¹, Luning Yang², Chris Gonzalo¹, Hao Chen¹

¹University of California, Davis; ²University of Hong Kong



Background

Design of TF-Bench

Experimental results

References

Backup slides

Test-time compute: let's think step by step

- Chain-of-Thought prompting [Wei+22] enables *reasoning* in LLMs.
- CoT decomposes multi-step problems into intermediate (reasoning) steps.
- CoT becomes a *training paradigm* [Ope24; Dee+25; Mue+25].

How many r in raspberry? **Question**

Let's break down the process of counting the letter 'r' in the word "raspberry" ...
* First letter: 'r' - This is an 'r', count = 1.
* Second letter: 'a' - Not an 'r', count remains 1 ...
* Sixth letter: 'e' - Not an 'r', count remains 1.
* Seventh letter: 'r' - This is an 'r', count = 2.
* Eighth letter: 'y' - Not an 'r', count remains 2 ...
The number of 'r's in "raspberry" is 2.
Wait, let's re-read the question carefully. It asks "How many r in raspberry?" ... * r - a - s - p - b - e - r - r - y ... * First 'r' ... *
Second 'r' ... * Third 'r' ... Count = 3 ... **Reasoning trace**

My initial answer of 2 was incorrect due to a quick reading of the word. **Final Answer:** The final answer is **3** **Response**

- Pre-trained knowledge: rules
- In-context knowledge + prompt: assumptions/hypotheses
- CoT: reasoning
 - step 1: lemma 1
 - step 2: lemma 2
 - ...
- Response: a proposition

Figure 1: TTC example [Mue+25].

The evaluation gap

Reasoning LLMs are evaluated on math and coding:

- Mathematical reasoning
 - AIME 2024: high school math competition
 - MATH-500: math problem solving
- Code generation
 - LiveCodeBench [Jai+25]: LeetCode, online coding contests
 - SWE-Bench [Jim+24]: patch generation

What is missing in the idea of *reasoning* for code?

- *program-centric* deductive system
- \implies *reason* about structural logic behind programs

Propositions as types

- Types in programming \cong propositions in logic [Wad15].
- A function type $A \rightarrow B \cong$ the proposition $A \implies B$ [Cur34].
- Type inference is natural deduction.

```
1 not :: Bool -> Bool
2 (.) :: (b -> c) -> (a -> b) -> a -> c
3 span :: (a -> Bool) -> [a] -> ([a], [a])
4
5 break p = span (not . p)
6 -- complete the following type signature
   for `break`
7 break :: (a -> Bool) -> [a] -> ([a], [a])
```

Listing 1: Example task for the break function

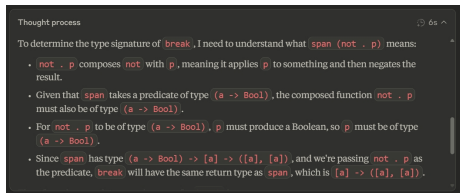


Figure 2: Claude 3.7' extended thinking mode on the task in Listing 1, ✓.

But wait, are LLMs really reasoning about *program semantics*?

Input transformations lead to a significant performance drop:

- Semantic-preserving code transformations [AD22; Yan+22; Liu+23].
- Perturbations on math problems [Mir+24; Jia+24; Gul+24].

```
1 f2 :: (t1 -> T1) -> [t1] -> ([t1], [t1])
2 f3 :: T1 -> T1
3 f4 :: (t1 -> t2) -> (t3 -> t1) -> t3->t2
4
5 f1 p = f2 (f3 `f4` p)
6 -- complete the following type signature
   for `f1`
7 f1 :: (t1 -> T1) -> [t1] -> ([t1], [t1])
```

Listing 2: The task in 1 after alpha-rewrite

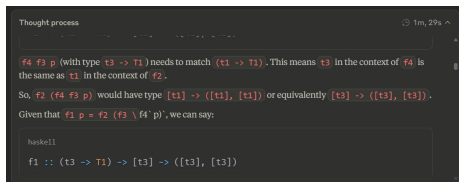


Figure 3: Claude 3.7' extended thinking mode on the task in Listing 2, ✓.

Background

Design of TF-Bench

Experimental results

References

Backup slides

Benchmark construction

We use the Haskell Prelude [Jono3] for the following reasons:

1. A formal deductive type system: System F (and System $F_{<^1}$).
2. The type signature is concise and is decoupled from the function body.
3. Type equivalence is formally verifiable (w/ signature only).
4. Haskell is the most popular language that meets these conditions.

```
1 map :: (a -> b) -> [a] -> [b]
2 map f [] = []
3 map f (x:xs) = f x : map f xs
```

Listing 3: Parametric polymorphic

```
1 (==) :: Eq a => a -> a -> Bool
2 x == y = not (x /= y)
```

Listing 4: Ad-hoc polymorphism

¹for bounded quantification

Removing natural language from tasks

We design three *alpha-rewrite* operators to remove NL from the tasks:

- NL-Type: `Int`, `Char`, `Bool`, `Eq`, `Ord` ... \rightarrow T1, T2, T3, T4, T5 ...
- Type-Var: `a`, `b`, `c`, `d`, `e` ... \rightarrow t1, t2, t3, t4, t5 ...
- Binding: `map`, `not`, `foldl`, `(+)` ... \rightarrow f1, f2, f3, f4 ...

Implementation details:

- Operators have type `Task -> Either Task Error`.
- Operators are commutative and associative under composition [Kle65].
- Transformed tasks are *alpha-equivalent* to the original tasks.

Construction pipeline and statistics

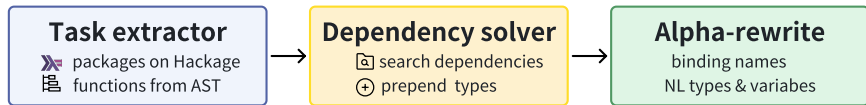


Figure 4: Pipeline to construct TF-Bench.

In total, TF-Bench has 188 tasks,

- 26.6% are monomorphic functions,
- 32.4% are parametric polymorphisms,
- and 41.0% are ad-hoc polymorphisms.

TF-Bench_{pure} is the NL-free version of TF-Bench.

Evaluation methodology

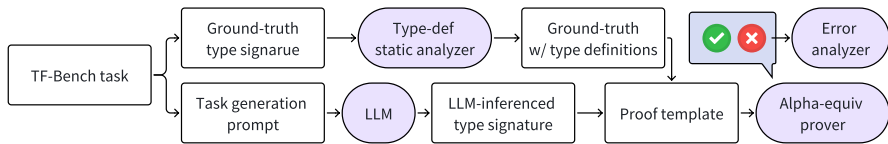


Figure 5: Pipeline to evaluate LLMs on TF-Bench.

1. We use the ground-truth signature to add definitions for new types.
2. We prompt the LLMs with tasks to generate a type signature.
3. We construct a proof using the ground-truth and generated signatures.
4. If the proof compiles, the two signatures are equivalent.
5. Otherwise we analyze what the error is.

Background

Design of TF-Bench

Experimental results

References

Backup slides

Research questions

We ask the following questions around *program semantics reasoning*:

1. What is the performance gap of LLMs on TF-Bench and TF-Bench_{pure}?
2. How effective is TTC of models after reinforcement learning?
3. Can fine-tuning on math/code improve reasoning?

RQ1. The performance gap

Model	Version	TTC	Acc	Acc _{pure}	RS
Claude-3.5-sonnet	2024-06-20	✗	85.46	48.97	57.3
Claude-3.7-sonnet	2025-02-19	✓	<u>90.42</u>	55.85	<u>61.77</u>
GPT-O3-mini	2025-01-31	✓	90.43	48.40	53.52
GPT-O3	2025-04-16	✓	81.91	<u>52.66</u>	64.29
DeepSeek-V3	2025-03-25	✗	83.51	43.62	52.23
DeepSeek-R1	2025-01-20	✓	86.70	44.15	50.92
Qwen3	30B-A3B	✓	81.38	40.43	49.68
	32B	✓	87.94	43.09	49.00
	235B-A22B-FP8	✓	85.11	44.15	51.87

Table 1: Main evaluation results. $RS(m) = \text{Acc}_{\text{pure}}(m)/\text{Acc}(m)$.

RQ2. The effectiveness of TTC

Model	TTC	Acc	Acc _{pure}	RE
Qwen3-235B-FP8	✗	80.49	35.64	1.37
	✓	86.70	44.15	
Claude-3.7-sonnet	✗	87.77	46.81	3.41
	✓	90.42	55.85	
Gemini-2.5-flash	✗	78.19	30.32	3.90
	✓	83.51	51.06	

Table 2: Reasoning effectiveness of top LLMs.

$$\text{RE}(m_{\text{ttc}}, m) = \frac{\text{Acc}_{\text{pure}}(m_{\text{ttc}}) - \text{Acc}_{\text{pure}}(m)}{\text{Acc}(m_{\text{ttc}}) - \text{Acc}(m)} = \frac{\Delta_{\text{pure}}}{\Delta}.$$

RQ3: Fine-tuning on math/code I

FT Corpus	Base Model (FT Model)	Size	Acc	FT Acc	Δ	Acc _{pure}	FT Acc _{pure}	Δ_{pure}
Code	Gemma (CodeGemma)	7B	48.94	53.19	+ 4.25	7.45	12.23	+ 4.78
	DeepSeek-V2 (-Coder)	16B	29.79	55.32	+ 25.53	7.98	15.96	+ 7.98
		236B	38.30	80.85	+ 42.55	11.17	36.70	+ 25.53
	Mistral (Codestral)	22B	61.17	63.30	+ 2.13	19.68	11.17	- 8.51
	Qwen2.5 (-Coder)	1.5B	30.32	36.70	+ 6.38	6.91	9.04	+ 2.13
		7B	65.96	61.17	- 4.79	21.28	21.28	0.00
		32B	74.47	82.45	+ 7.98	36.17	31.91	- 4.26
Math	Mistral (Mathstral)	7B	45.21	47.34	+ 2.13	7.99	15.43	+ 7.44
	Qwen2 (-Math)	7B	40.43	43.09	+ 2.66	3.19	10.64	+ 7.45
		72B	63.83	71.28	+ 7.45	21.81	33.51	+ 11.7

Table 3: Result comparison of fine-tuning. FT Corpus: the corresponding fine-tuning corpus. $\Delta, \Delta_{\text{pure}}$: absolute increase in accuracy after fine-tuning.

RQ3: Fine-tuning on math/code II

- fine-tuning on code sometimes leads to a - **decline** in performance,
- fine-tuning on math consistently results in performance + **gains**,
- fine-tuning on code exhibit smaller or negative improvements on $\text{TF-Bench}_{\text{pure}}$, i.e. $\text{RE} < 0$,
- the *same models* fine-tuned on math demonstrate greater improvements on $\text{TF-Bench}_{\text{pure}}$, i.e. $\eta > 1$, although not as significant as TTC.

Observation: Fine-tuning on math might enhance the models' reasoning ability, which also translates effectively tasks related to code.

Background

Design of TF-Bench

Experimental results

References

Backup slides

- [Wei+22] Jason Wei et al. “Chain of Thought Prompting Elicits Reasoning in Large Language Models”. In: *Advances in Neural Information Processing Systems*. 2022. URL: https://openreview.net/forum?id=_VjQlMeSB_J.
- [Ope24] OpenAI. *OpenAI o1 System Card*. 2024. URL: <https://cdn.openai.com/o1-system-card.pdf>.
- [Dee+25] DeepSeek-AI et al. *DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning*. 2025. arXiv: 2501.12948 [cs.CL]. URL: <https://arxiv.org/abs/2501.12948>.
- [Mue+25] Niklas Muennighoff et al. “s1: Simple test-time scaling”. In: *arXiv preprint arXiv:2501.19393* (2025).
- [Jai+25] Naman Jain et al. “LiveCodeBench: Holistic and Contamination Free Evaluation of Large Language Models for Code”. In: *The Thirteenth International Conference on Learning Representations*. 2025. URL: <https://openreview.net/forum?id=chfJJYC3iL>.

- [Jim+24] Carlos E Jimenez et al. “SWE-bench: Can Language Models Resolve Real-world Github Issues?” In: *The Twelfth International Conference on Learning Representations*. 2024. URL: <https://openreview.net/forum?id=VTF8yNQM66>.
- [Wad15] Philip Wadler. “Propositions as types”. In: *Commun. ACM* 58.12 (Nov. 2015), pp. 75–84. ISSN: 0001-0782. DOI: 10.1145/2699407. URL: <https://doi.org/10.1145/2699407>.
- [Cur34] H. B. Curry. “Functionality in Combinatory Logic”. In: *Proceedings of the National Academy of Sciences* 20.11 (1934), pp. 584–590. DOI: 10.1073/pnas.20.11.584. eprint: <https://www.pnas.org/doi/pdf/10.1073/pnas.20.11.584>. URL: <https://www.pnas.org/doi/abs/10.1073/pnas.20.11.584>.
- [AD22] Toufique Ahmed and Premkumar Devanbu. “Multilingual training for software engineering”. In: *Proceedings of the 44th International Conference on Software Engineering*. ICSE ’22. ACM, May 2022. DOI: 10.1145/3510003.3510049. URL: <http://dx.doi.org/10.1145/3510003.3510049>.

- [Yan+22] Zhou Yang, Jieke Shi, Junda He, and David Lo. “Natural attack for pre-trained models of code”. In: *Proceedings of the 44th International Conference on Software Engineering*. ICSE ’22. Pittsburgh, Pennsylvania: Association for Computing Machinery, 2022, pp. 1482–1493. ISBN: 9781450392211. DOI: 10.1145/3510003.3510146. URL: <https://doi.org/10.1145/3510003.3510146>.
- [Liu+23] Shangqing Liu, Bozhi Wu, Xiaofei Xie, Guozhu Meng, and Yang Liu. “Contrabert: Enhancing code pre-trained models via contrastive learning”. In: *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE. 2023, pp. 2476–2487.
- [Mir+24] Iman Mirzadeh et al. *GSM-Symbolic: Understanding the Limitations of Mathematical Reasoning in Large Language Models*. 2024. arXiv: 2410.05229 [cs.LG]. URL: <https://arxiv.org/abs/2410.05229>.

- [Jia+24] Bowen Jiang et al. “A Peek into Token Bias: Large Language Models Are Not Yet Genuine Reasoners”. In: *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. Miami, Florida, USA: Association for Computational Linguistics, Nov. 2024, pp. 4722–4756. DOI: 10.18653/v1/2024.emnlp-main.272. URL: <https://aclanthology.org/2024.emnlp-main.272/>.
- [Gul+24] Aryan Gulati et al. “Putnam-AXIOM: A Functional and Static Benchmark for Measuring Higher Level Mathematical Reasoning”. In: *The 4th Workshop on Mathematical Reasoning and AI at NeurIPS’24*. 2024. URL: <https://openreview.net/forum?id=YXnwlZe0yf>.
- [Jono3] Simon Peyton Jones. *Haskell 98 language and libraries: the revised report*. Cambridge University Press, 2003.
- [Kle65] H. Kleisli. “Proc. Amer. Math. Soc. 16 (1965), 544-546”. In: *Proceedings of the American Mathematical Society* 16 (1965), pp. 544–546. ISSN: 0002-9939. DOI: 10.1090/S0002-9939-1965-0177024-4.

Background

Design of TF-Bench

Experimental results

References

Backup slides

Formal reasoning in deductive systems

Definition

A deductive system (or inference system) is specified by ²

- a collection of judgments/assertions/validations,
- a collection of steps (inference rules) that move from validation to validation, and finally to the proposition.

Natural deduction is a deductive system that *reason from assumptions*.

²This is not a completely standard definition, but is an illustrative description.

Effects of different rewrite operators

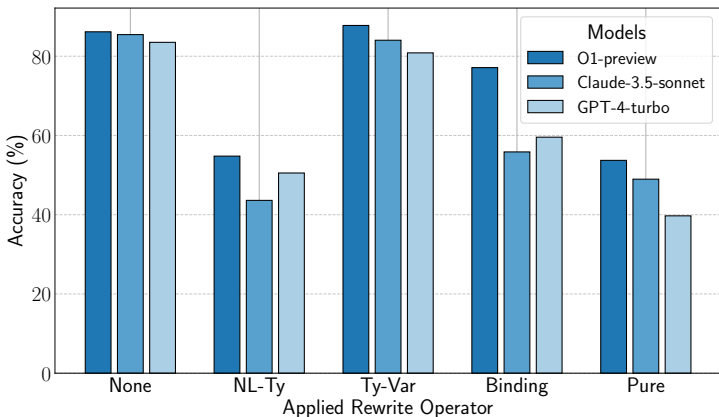


Figure 6: Accuracy on TF-Bench with different rewrite operators. None: the original TF-Bench. NL-Ty: rewriting NL types. Ty-Var: rewriting type variables. Binding: rewriting binding names. Pure: $\text{TF-Bench}_{\text{pure}}$.