Three Lessons from Evaluating Language Models on Reasoning Tasks

Alex Gu, MIT CSAIL
gua@mit.edu

1. Focusing on the Right Metrics Without Bias is Crucial

2. Looking Beyond the Score Reveals Hidden Nuances

3. What We Measures Shapes What We Build

Timeline

June 2020 *- GPT-3*

Timeline

June 2020 *- GPT-3*

…then Codex, Copilot, InstructGPT

InstructGPT

User: *Tell me about when Christopher Columbus came to the US in 2015*

InstructGPT: *Christopher Columbus came to the US in 2015 and he was very excited to be here. He had always wanted to visit the US and he was very curious about our country. He was very impressed with our country and he enjoyed his time here.*

November 2022 - ChatGPT

User: *Tell me about when Christopher Columbus came to the US in 2015*

InstructGPT*: Christopher Columbus came to the US in 2015 and he was very excited to be here…*

ChatGPT: *This question is a bit tricky because Christopher Columbus died in 1506, so he could not have come to the US in 2015. But let's pretend for a moment that he did! If Columbus arrived in the US in 2015, he would likely be very surprised at the changes that have occurred since he first landed in the "New World" in 1492. For one, he would probably be shocked to find out that the land he "discovered" was actually already…*

Timeline

June 2020 *- GPT-3*

…then Codex, Copilot, InstructGPT

November 2022 *- ChatGPT (GPT-3.5)*

March 2023 *- GPT-4*

only 4 months!

Sparks of Artificial General Intelligence: Early experiments with GPT-4

Sébastien Bubeck Varun Chandrasekaran Ronen Eldan Johannes Gehrke
Eric Horvitz Ece Kamar Peter Lee Yin Tat Lee Yuanzhi Li Scott Lundberg
Harsha Nori Hamid Palangi Marco Tulio Ribeiro Yi Zhang

Microsoft Research

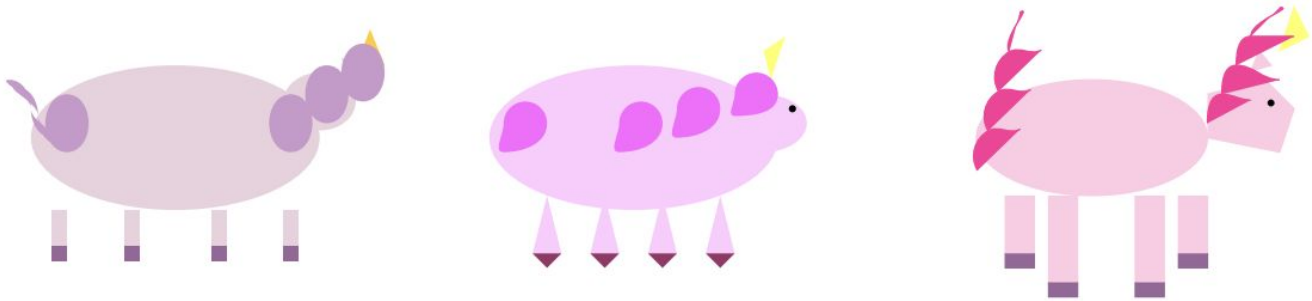


Figure 1.3: We queried GPT-4 three times, at roughly equal time intervals over the span of a month while the system was being refined, with the prompt "Draw a unicorn in TikZ". We can see a clear evolution in the sophistication of GPT-4's drawings.

Benchmark	GPT-4 Evaluated few-shot	GPT-3.5 Evaluated few-shot
MMLU Multiple-choice questions in 57 subjects (professional & academic)	86.4%	70.0%
HellaSwag Commonsense reasoning around everyday events	95.3%	85.5%
AI2 Reasoning Challenge (ARC) Grade-school multiple choice science questions. Challenge-set.	96.3%	85.2%
WinoGrande Commonsense reasoning around pronoun resolution	87.5%	81.6%
HumanEval Python coding tasks	67.0%	48.1%
DROP (f1 score) Reading comprehension & arithmetic.	80.9	64.1

1. Focusing on the Right Metrics Without Bias is Crucial

2. Looking Beyond the Score Reveals Hidden Nuances

3. What We Measures Shapes What We Build

1. Focusing on the Right Metrics Without Bias is Crucial

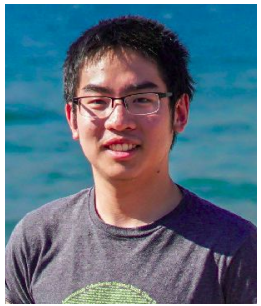
2. Looking Beyond the Score Reveals Hidden Nuances

3. What We Measures Shapes What We Build

CRUXEval

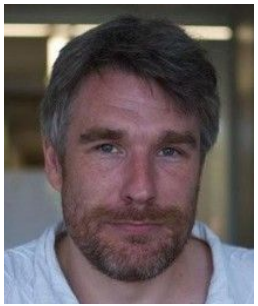
Code Reasoning, Understanding, and Execution Evaluation

Alex Gu



Baptiste Rozière

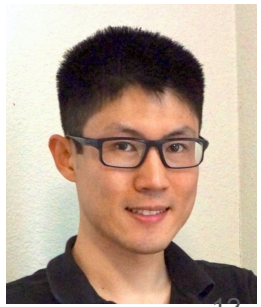
Hugh Leather



Armando Solar-Lezama

Gabriel Synnaeve

Sida Wang



CRUXEval

Input Prediction (I)

Find an input producing a given output

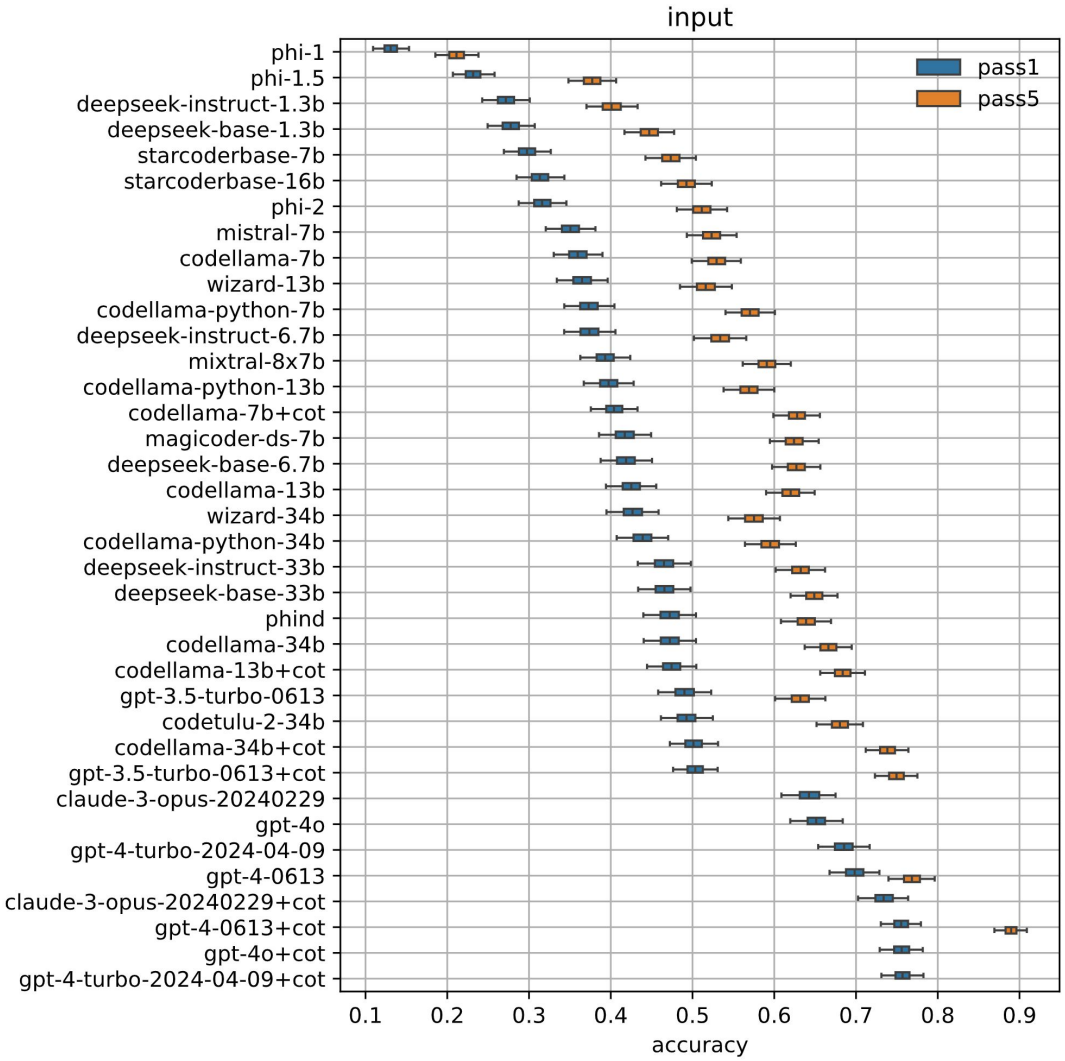
(code understanding + reasoning)

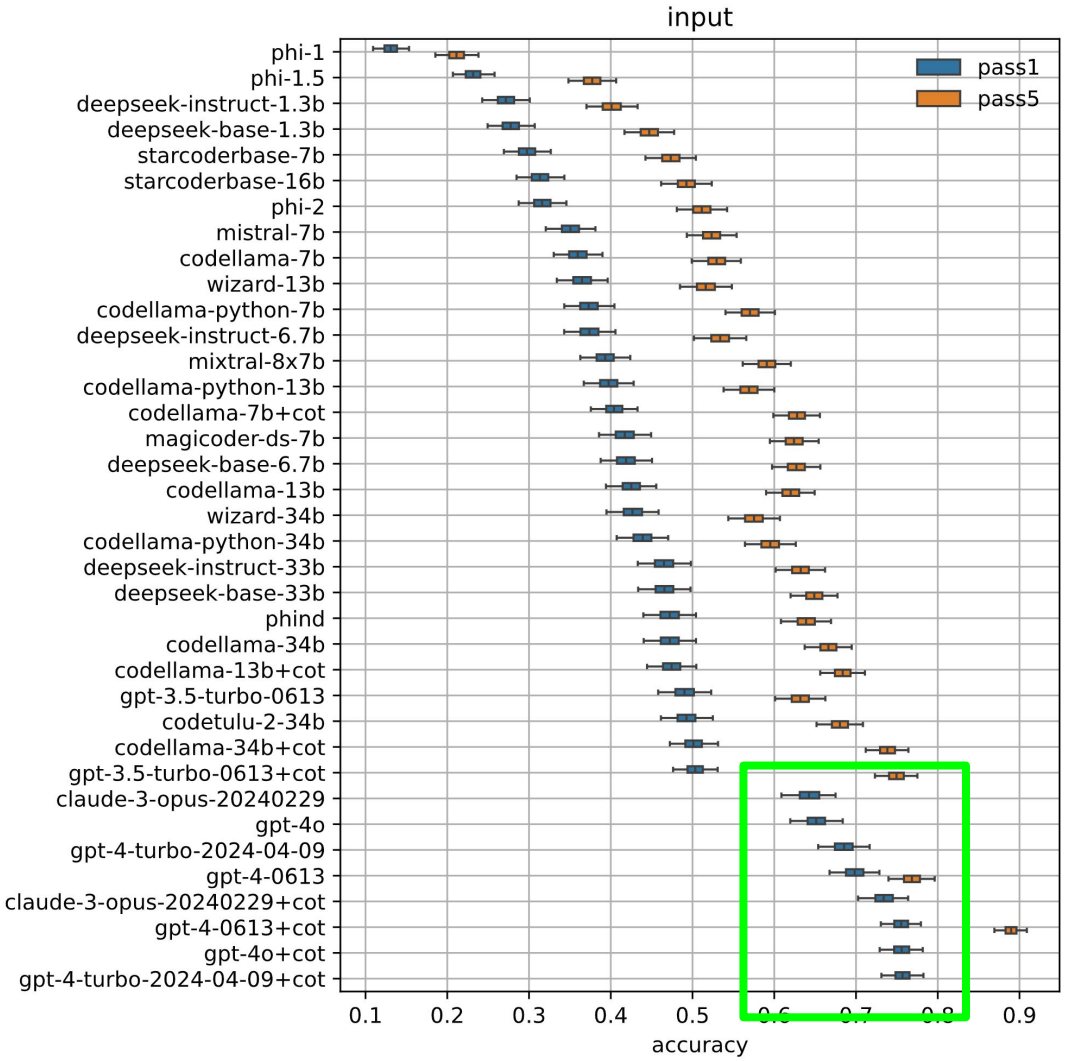
Output Prediction (O)

Execute the function on a given input

(code execution)

```
def f(string):
    string_x = string.rstrip("a")
    string = string_x.rstrip("e")
    return string

# output prediction, CRUXEval-O
assert f("xxxxaaee") == ??
## GPT4: "xxxx", incorrect

# input prediction, CRUXEval-I
assert f(??) == "xxxxaa"
## GPT4: "xxxxaae", correct
```


input
phi-1
phi-1.5
deepseek-instruct-1.3b
deepseek-base-1.3b
starcoderbase-7b
starcoderbase-16b
phi-2
mistral-7b
codellama-7b
wizard-13b
codellama-python-7b
deepseek-instruct-6.7b
mixtral-8x7b
codellama-python-13b
codellama-7b+cot
magicoder-ds-7b
deepseek-base-6.7b
codellama-13b
wizard-34b
codellama-python-34b
deepseek-instruct-33b
deepseek-base-33b
phind
codellama-34b
codellama-13b+cot
gpt-3.5-turbo-0613
codetulu-2-34b
codellama-34b+cot
gpt-3.5-turbo-0613+cot
claude-3-opus-20240229
gpt-4o
gpt-4-turbo-2024-04-09
gpt-4-0613
claude-3-opus-20240229+cot
gpt-4-0613+cot
gpt-4o+cot
gpt-4-turbo-2024-04-09+cot
pass1
pass5
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
accuracy


input
pass1
pass5
phi-1
phi-1.5
deepseek-instruct-1.3b
deepseek-base-1.3b
starcoderbase-7b
starcoderbase-16b
phi-2
mistral-7b
codellama-7b
wizard-13b
codellama-python-7b
deepseek-instruct-6.7b
mixtral-8x7b
codellama-python-13b
codellama-7b+cot
magicoder-ds-7b
deepseek-base-6.7b
codellama-13b
wizard-34b
codellama-python-34b
deepseek-instruct-33b
deepseek-base-33b
phind
codellama-34b
codellama-13b+cot
gpt-3.5-turbo-0613
codetulu-2-34b
codellama-34b+cot
gpt-3.5-turbo-0613+cot
claude-3-opus-20240229
gpt-4o
gpt-4-turbo-2024-04-09
gpt-4-0613
claude-3-opus-20240229+cot
gpt-4-0613+cot
gpt-4o+cot
gpt-4-turbo-2024-04-09+cot
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
accuracy

GPT-4 + CoT Failures

```
def f(text, suffix):
    if suffix == '':
        suffix = None
    return text.endswith(suffix)
assert f('uMeGndkGh', 'kG') == ??
# GPT-4 CoT: True
# should be False
```

```
def f(num):
    if 0 < num < 1000 and num != 6174:
        return 'Half Life'
    return 'Not found'
assert f(6173) == ??
# GPT-4 CoT: 'Half Life'
# should be 'Not found'
```

```
def f(text, repl):
    trans = str.maketrans(text.lower(), repl.
    ↪ lower())
    return text.translate(trans)
assert f(??) == 'lwwer case'

# GPT4 CoT: 'lower case', 'ow'
# could be 'lower case', 'lwwer case'
```

```
def f(text):
    string = ''
    for char in text:
        string += char + char.lower()
    return string
assert f(??) == 'llaallaakk'
# GPT-4 CoT: 'LAK'
# should be 'lalak'
```

Function Anonymization

Original

```
def f(s):
    nums = ''.join(filter(lambda c:c.isdecimal(), s))
    if nums == '': return 'none'
    m = max([int(num) for num in nums.split(',')])
    return str(m)
assert f('01,001') == '1001'
```

Anonymized

```
def f(x0):
    x1 = ''.join(filter(lambda x2: x2.isdecimal(), x0))
    if x1 == '':
        return 'none'
    x3 = max([int(x4) for x4 in x1.split(',')])
    return str(x3)
assert f('01,001') == '1001'
```

No significant drop from anonymizing variable names!

Model	Anonymized	Input Prediction		Output Prediction	
		Pass@1	Pass@5	Pass@1	Pass@5
CodeLlama 7B	✗	36.6%	48.0%	36.4%	43.5%
	✓	37.5%	53.3%	34.0%	46.9%
	Δ	+0.9%	+5.3%	-2.4%	+3.4%
CodeLlama 13B	✗	39.0%	50.2%	38.3%	44.7%
	✓	40.0%	55.8%	36.1%	50.6%
	Δ	+1.0%	+5.6%	-2.2%	+5.9%
CodeLlama 34B	✗	46.5%	57.4%	41.1%	47.5%
	✓	48.0%	63.8%	39.1%	54.0%
	Δ	+1.5%	+6.4%	-2.0%	+6.5%

Focusing on the Right Metrics Without Bias is Crucial

Previous Metric

Function-level code generation (HumanEval, MBPP)

New Metric

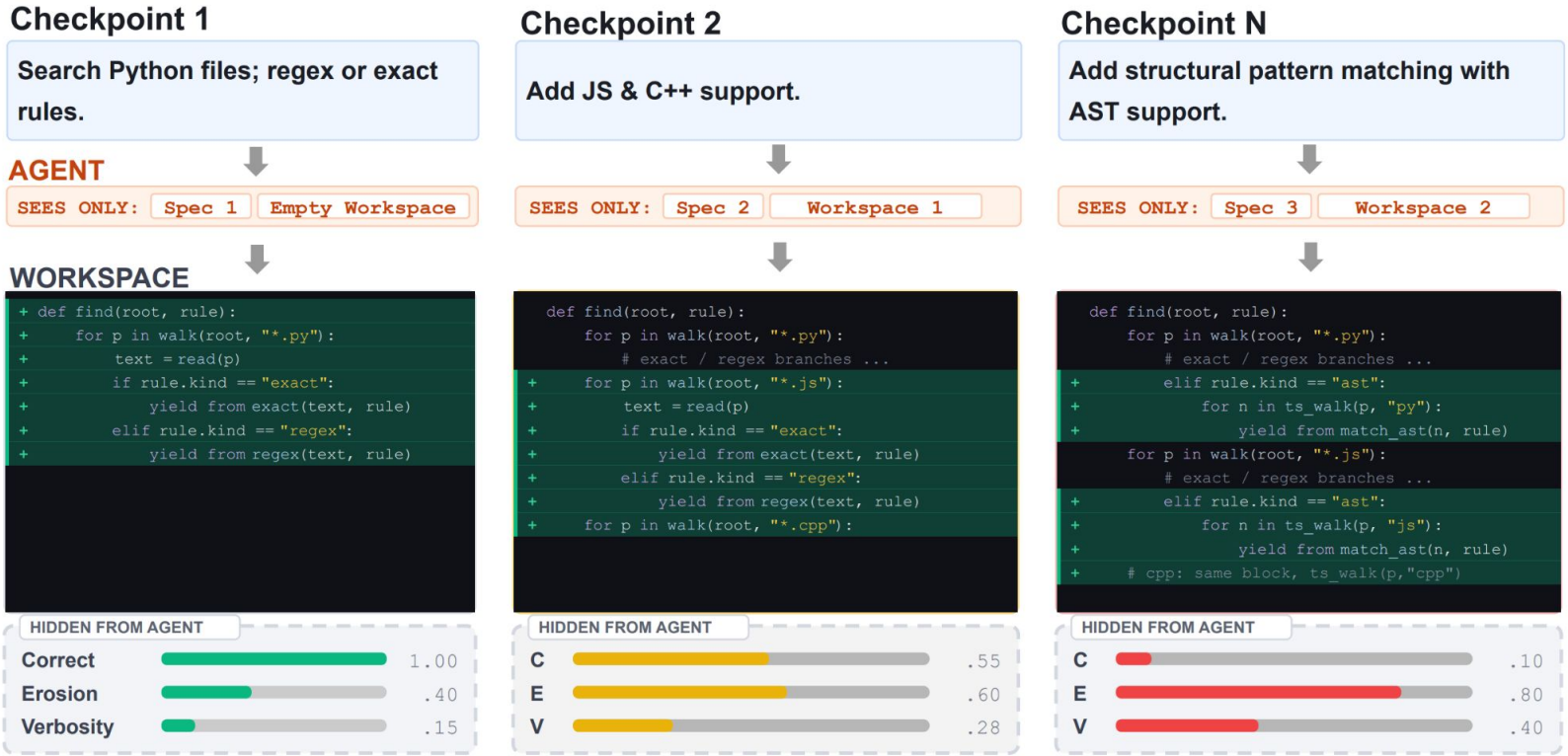
Code execution and reasoning

SlopCodeBench

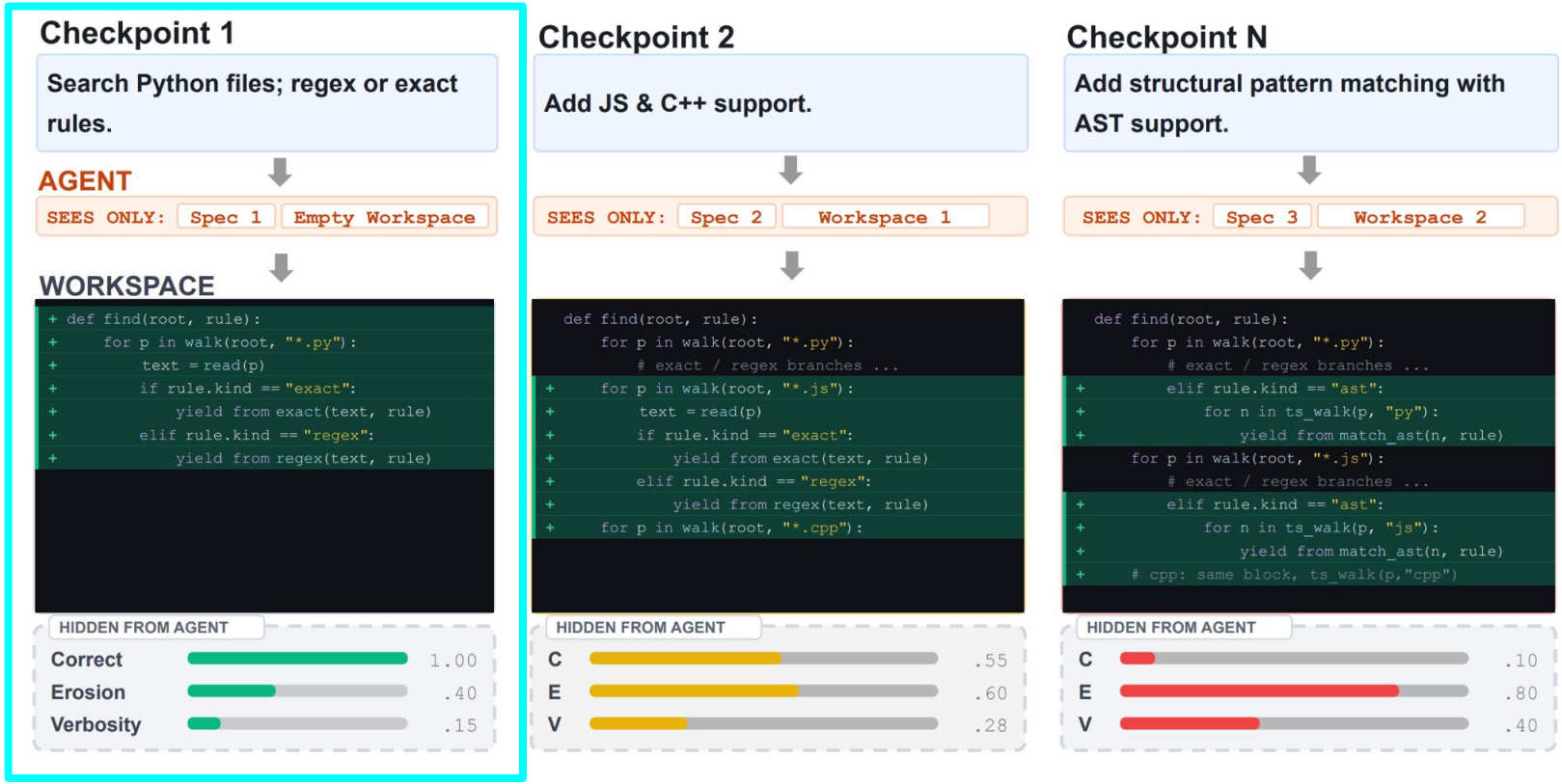
Benchmarking How Coding Agents Degrade Over Long-Horizon Iterative Tasks

Gabriel Orlanski, Devjeet Roy, Alexander Yun, Changho Shin, Alex Gu, Albert Ge, Dyah Adila, Nicholas Roberts, Frederic Sala, Aws Albarghouthi

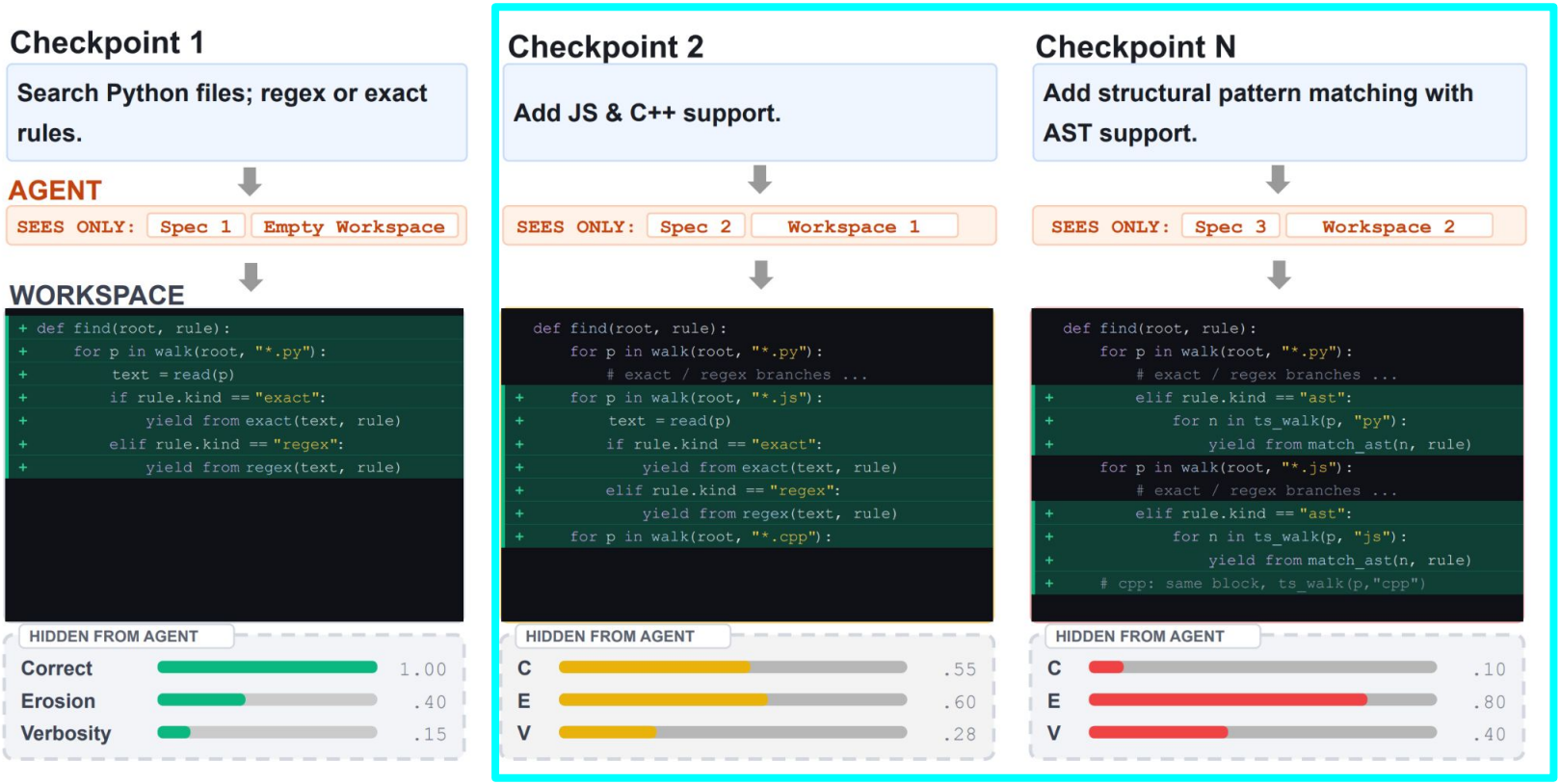
Iterative Evaluation



Iterative Evaluation



Iterative Evaluation



Task Example

Running example: `code_search`. The agent is tasked with building a CLI tool for semantic source-file search, inspired by ast-grep, across five checkpoints:

C_1 - Python files only exact and regex matching. Establishes the core CLI contract.

C_2 - Support for Javascript and C++.

C_3 - AST-based pattern matching with metavariable capture.

C_4 - Selector rules and auto-fix functionality.

C_5 - Add support for Go, Rust, and Java.

An agent that hardcodes language-specific logic at C_1 faces cascading rewrites at C_2 and C_5; one that builds an extensible parser interface does not.

Code Quality Metrics

Erosion: fraction of a codebase that consists of high cyclomatic complexity functions

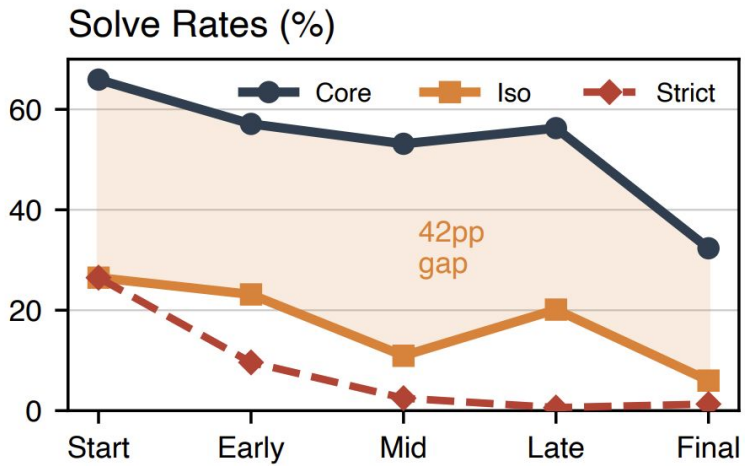
$$\text{mass}(f) = \text{CC}(f) \times \sqrt{\text{SLOC}(f)}$$

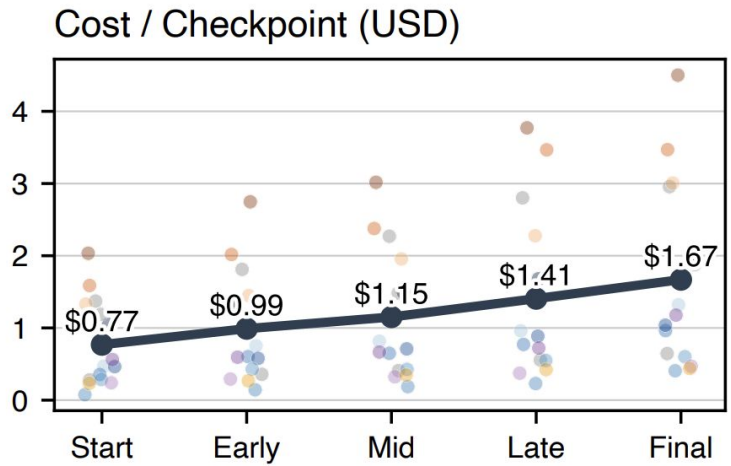
$$\text{Erosion} = \frac{\sum_{f \in \mathcal{F}} \mathbb{I}[\text{CC}(f) > 10] \cdot \text{mass}(f)}{\sum_{f \in \mathcal{F}} \text{mass}(f)}$$

Verbosity: fraction of redundant or duplicated code

$$\text{Verbosity} = \frac{|\{\text{AST-Grep Flagged Lines} \cup \text{Clone Lines}\}|}{\text{LOC}}$$

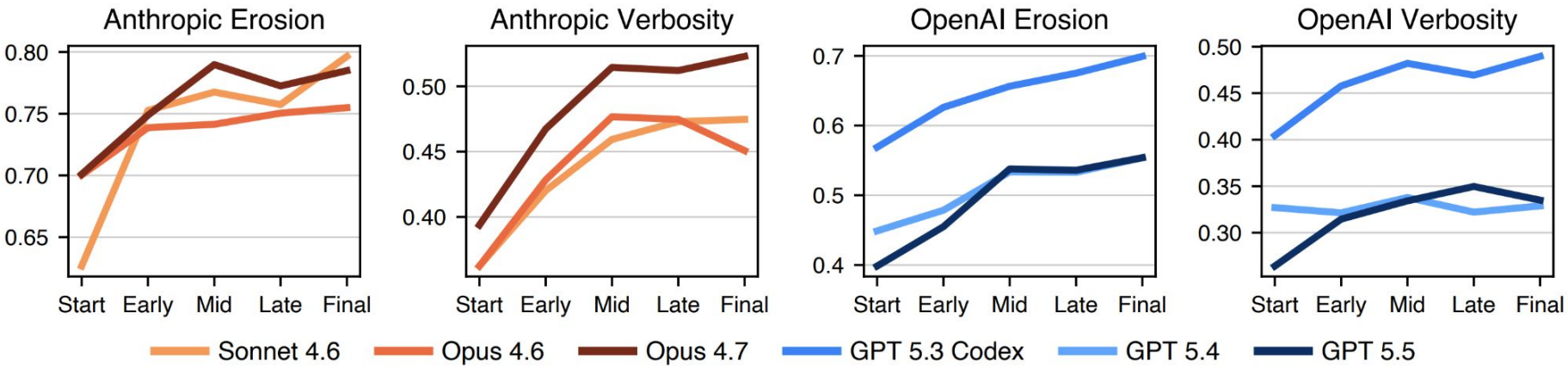
Results





Solve rates decrease despite cost increase as agents progress through SlopCodeBench tasks

Code Quality Over Time



Code quality decreases as agents progress through SlopCodeBench tasks

Agents vs. Humans

Agent trajectories have higher verbosity and erosion than human repository histories!

Focusing on the Right Metrics Without Bias is Crucial

Previous Metric

Single-shot Tasks

Correctness (via tests)

New Metrics

Iterative Evaluation

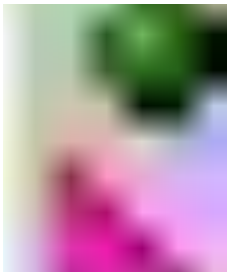
Verbosity and Erosion

IneqMath

Solving Inequality Proofs with Large Language Models

Pan Lu

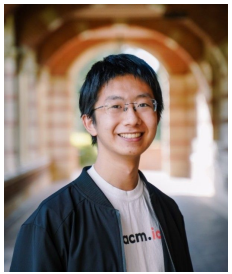
Jiayi Sheng



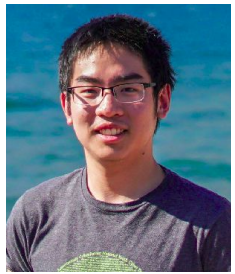
Luna Lyu

Jikai Jin

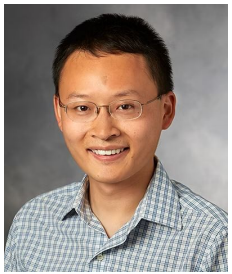
Tony Xia



Alex Gu



James Zou



IneqMath

Inequality proof problems reformulated into two verifiable subtasks

Find the largest C for all a,b>0:

$$a + b \geq C\sqrt{ab}$$

Bound Estimation

Determine the relationship for all a,b>0:

$$a + b \ ? \ 2\sqrt{ab}$$

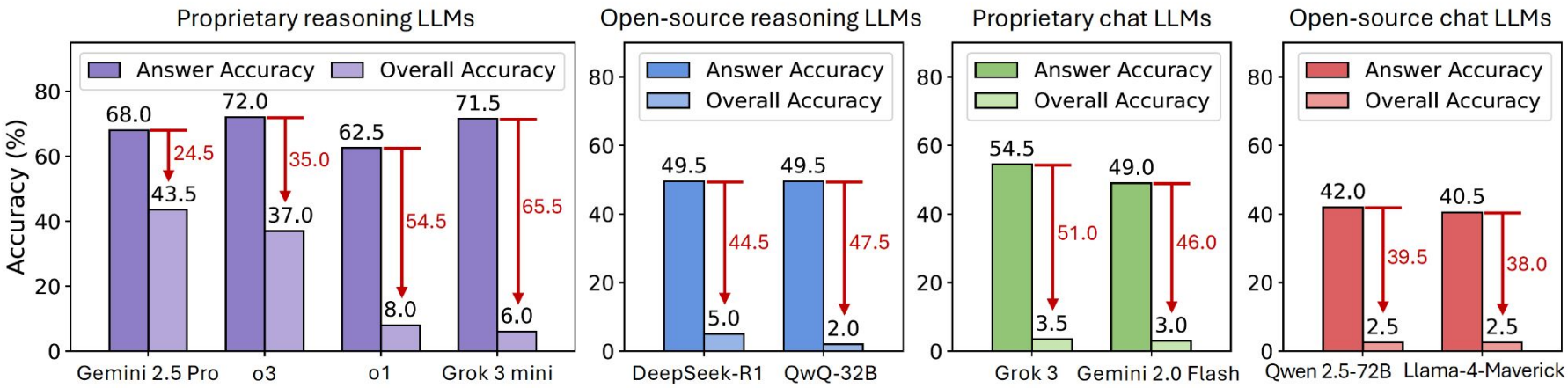
Relation Prediction

IneqMath Testing Example 1: Bound Problem

Question: Let $a_1, a_2, \ldots, a_n > 0$ such that $a_1 + a_2 + \ldots + a_n < 1$. Determine the minimal constant C such that the following inequality holds for all $a_1, a_2, \ldots, a_n$:

$$\frac{a_1 \cdot a_2 \ldots a_n \left(1 - a_1 - a_2 - \ldots - a_n\right)}{\left(a_1 + a_2 + \ldots + a_n\right)\left(1 - a_1\right)\left(1 - a_2\right)\ldots\left(1 - a_n\right)} \leq C\frac{3}{n^{n-1}}.$$

Soundness Gap



LLMs often guess the right answer, but their step-by-step reasoning is unsound!

Focusing on the Right Metrics Without Bias is Crucial

Previous Metric

Final-answer Accuracy

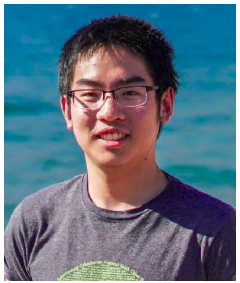
New Metric

Overall Accuracy (including proof soundness)

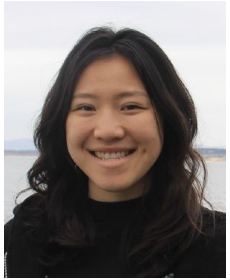
🕵️ The Counterfeit Conundrum

Can Code Language Models Grasp the Nuances of Their Incorrect Generations?

Alex Gu | Wen-Ding Li* | Theo X. Olausson* | Naman Jain* | Celine Lee* | Koushik Sen | Armando Solar-Lezama







TEACHING LARGE LANGUAGE MODELS TO SELF-DEBUG

Xinyun Chen[1] **Maxwell Lin**[2] **Nathanael Schärli**[1] **Denny Zhou**[1]
[1] Google DeepMind [2] UC Berkeley
{xinyunchen,schaerli,dennyzhou}@google.com, mxlin@berkeley.edu

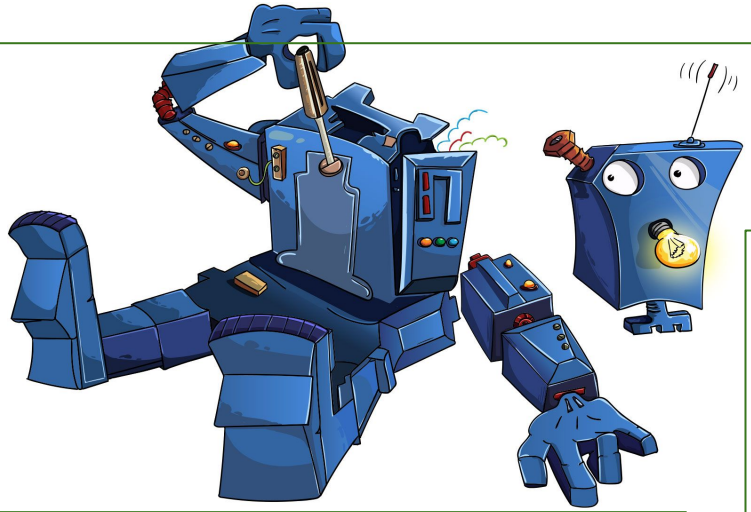
Self-Edit: Fault-Aware Code Editor for Code Generation

Kechi Zhang, Zhuo Li, Jia Li ♂, Ge Li*, Zhi Jin*
Key Lab of High Confidence Software Technology (PKU), Ministry of Education
School of Computer Science, Peking University, China
{zhangkechi,lizhmq}@pku.edu.cn, lijia@stu.pku.edu.cn,
{lige,zhijin}@pku.edu.cn

Given a list of distinct strings, check if any two have the same length.

```
>>> same_length(["aa", "b", "ccc", "dd"])
True
>>> same_length(["a", "bb", "ccc"])
False
```

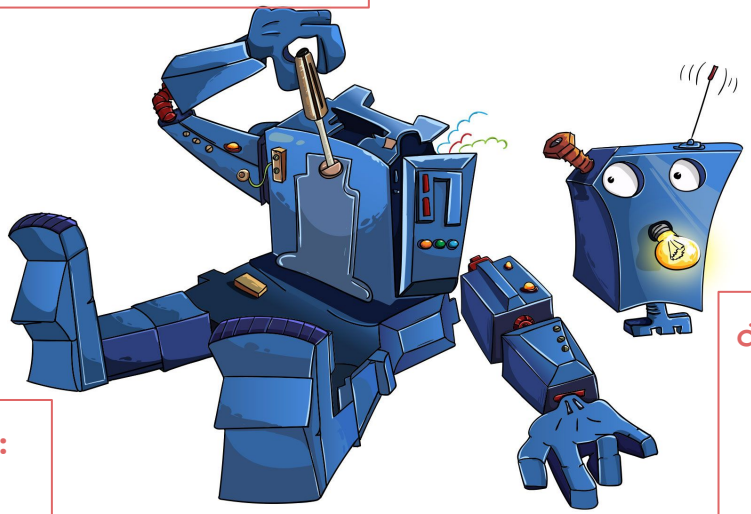
```
def same_length(s):
  return len(list(set([len(i) for i in s]))) != len(s)
```



```
def same_length(s):
  l = [len(i) for i in s]
  l = set(l)
  if len(s) > len(l):
    return True
  return False
```

```
def same_length(s):
  x = []
  for i in range(len(s)):
    for j in range(i+1, len(s)):
      x.append(len(s[i]) == len(s[j]))
  return True in x
```

```
def same_length(s):
    return len(s[0]) == len(s[1])
```



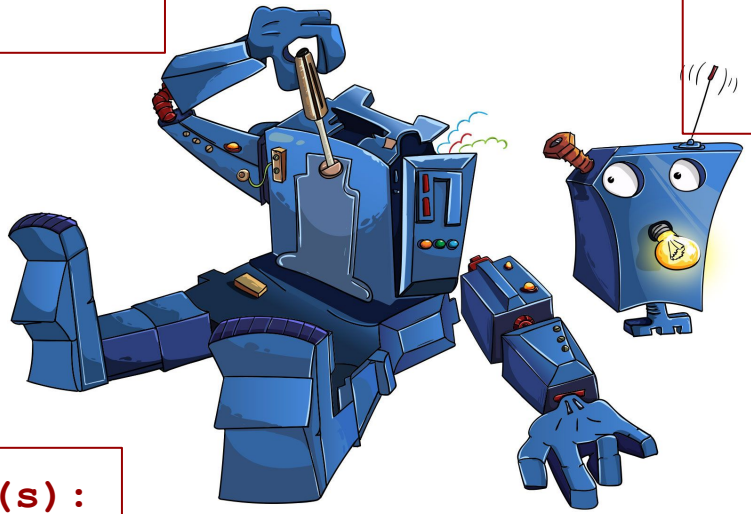
```
def same_length(s):
  if s == []: return False
  for a, b in zip(s, s[1:]):
    if len(a) == len(b):
      return True
  return False
```

```
def same_length(s):
  if len(s) < 2:
    return False
  return True
```

```
def same_length(s):
  for a in s:
    for b in s:
      if len(a) == len(b):
        return True
  return False
```

```
def same_length(s):
  pass
```

```
def same_length(s):
  return len(s) == 1
  if s == []:
    return False
```

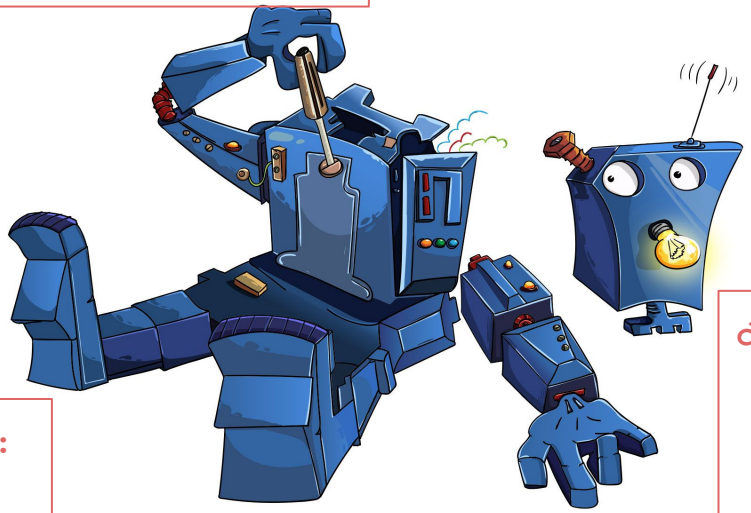


```
def same_length(s):
  return -1
```

```
def same_length(s):
  s = s +
  s = s + +
  s = s + s + s
  s = s + s
  s = s * s
```

```
def same_length(s):
  # write code here
  # CODE YOUR SOLUTION
```

```
def same_length(s):
    return len(s[0]) == len(s[1])
```

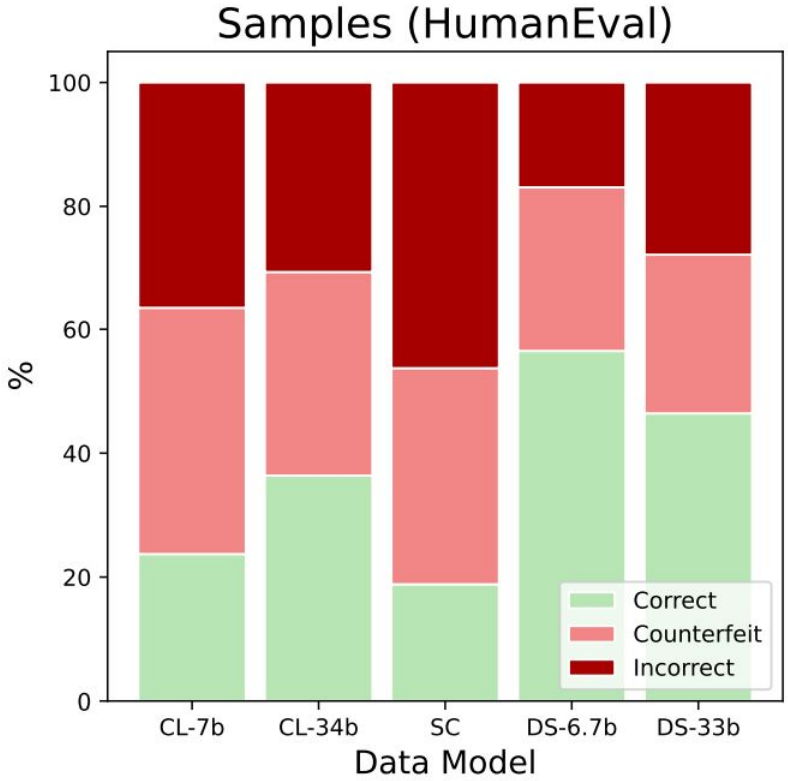


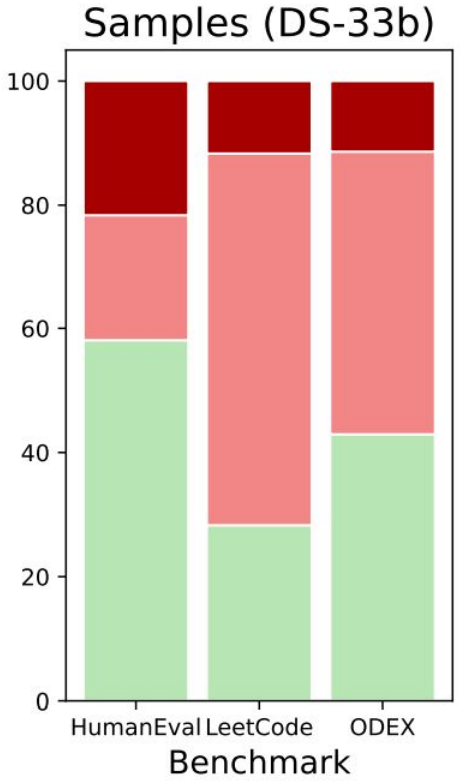
```
def same_length(s):
  if s == []: return False
  for a, b in zip(s, s[1:]):
    if len(a) == len(b):
      return True
  return False
```

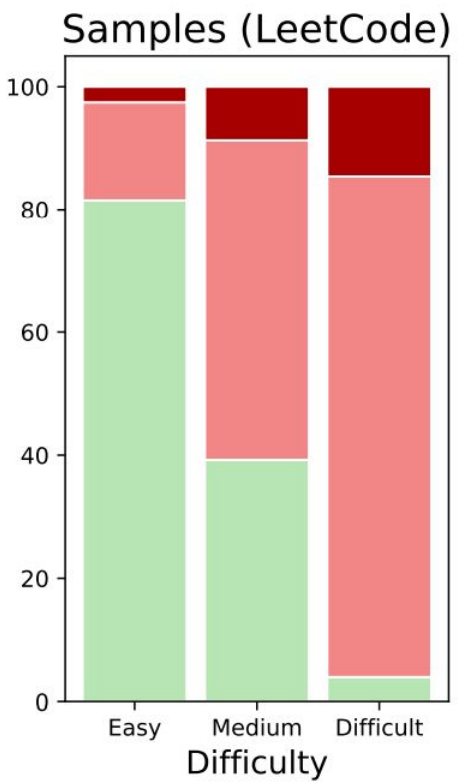
```
def same_length(s):
  if len(s) < 2:
    return False
  return True
```

```
def same_length(s):
  for a in s:
    for b in s:
      if len(a) == len(b):
        return True
  return False
```

counterfeit sample: **incorrect code generated by a model that passes weak correctness checks**







***counterfeits* are prevalent across all models, datasets, and difficulty levels!**

Generating Counterfeits

Datasets

HumanEval, LeetCode, ODEX: natural language to Python program

Models

Sample at T=0.6 from various models (CodeLlama, DeepSeek-I, StarCoder)

Counterfeit Filter

HumanEval: >10% of EvalPlus tests

ODEX: Passes `ast.parse` and under 500 characters

LeetCode: Wrong Answer verdict, filtering out crashed or hanging programs

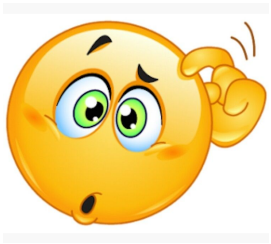
Code Understanding

Correctness Checking

```
# Given a number, adds one
def addOne(x):
    if x <= 3:
        return x + 1
    return x + 1 + x


Answer: Wrong
```

Execution Prediction

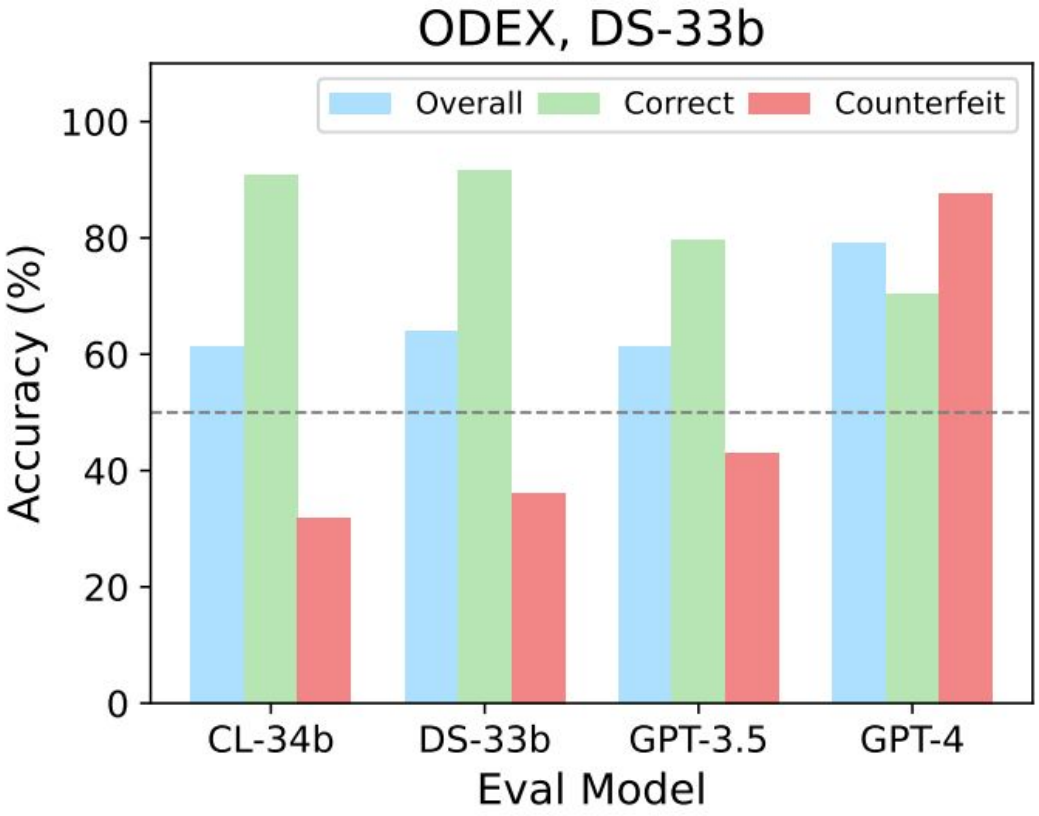
```
def addOne(x):
    if x <= 3:
        return x + 1
    return x + 1 + x

assert addOne(5) == ??

Answer: 11
```



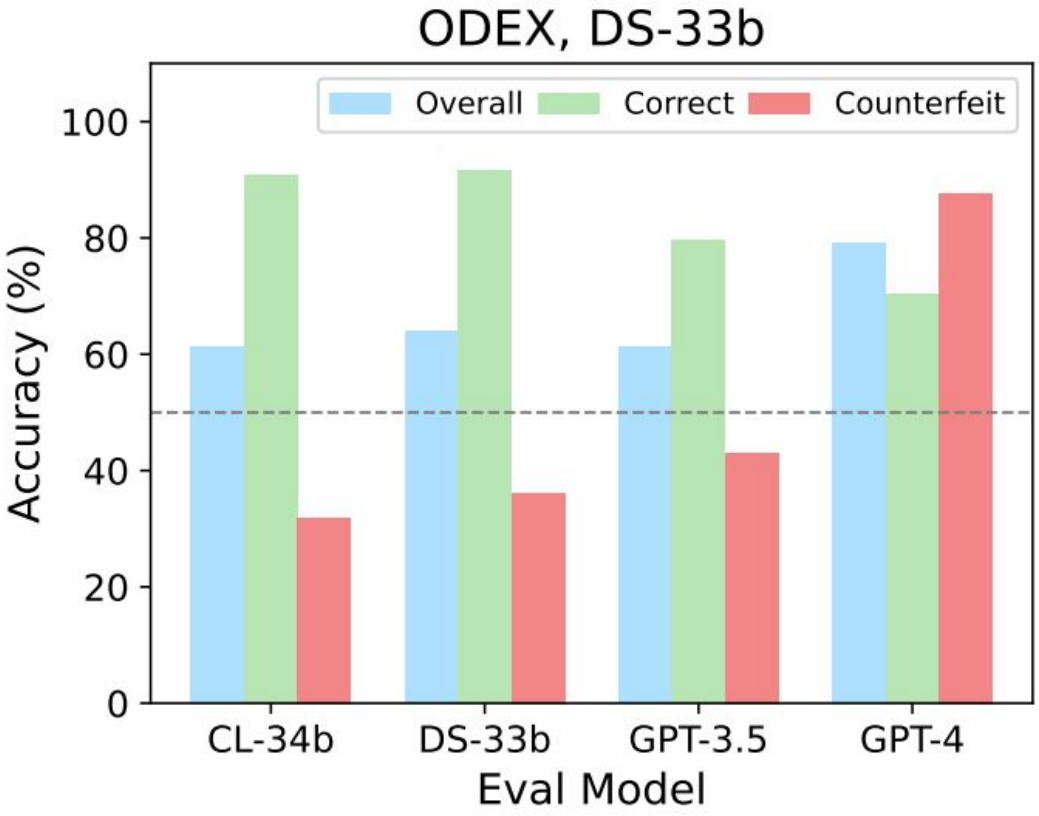
Program Repair

```
# Given a number, adds one
def addOne(x):
    if x <= 3:
        return x + 1
    return x + 1 + x

Possible Answer:
def addOne(x): return x + 1
```

Correctness Checking



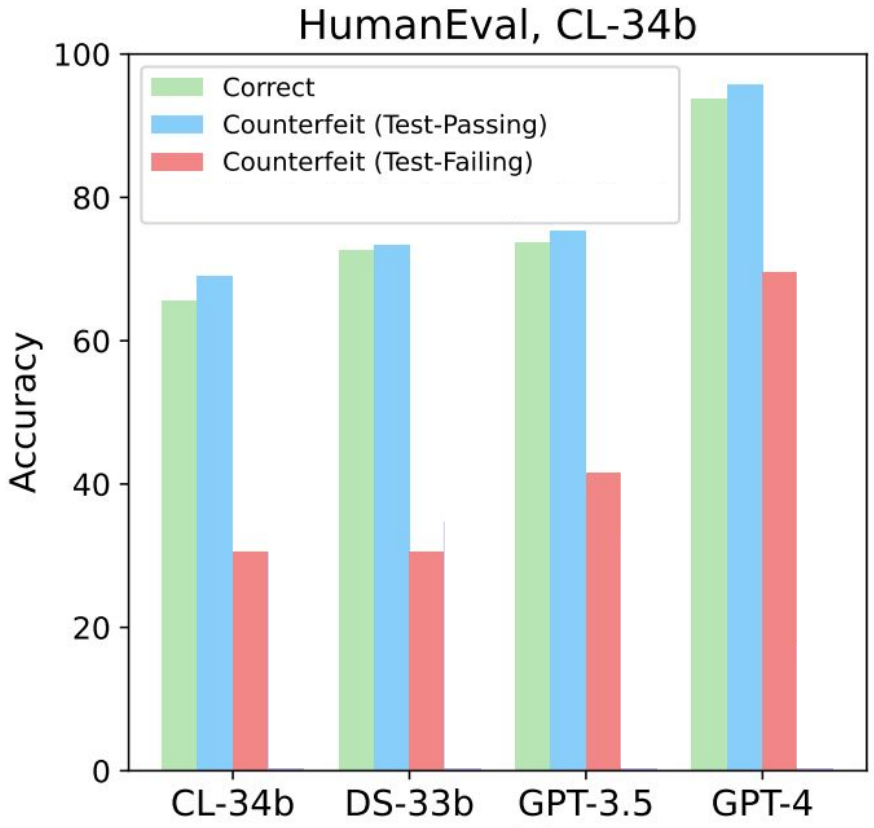
Correctness Checking



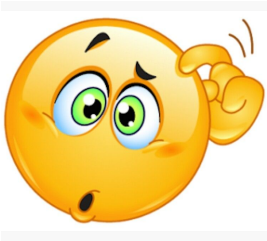
models often classify counterfeit samples as correct!

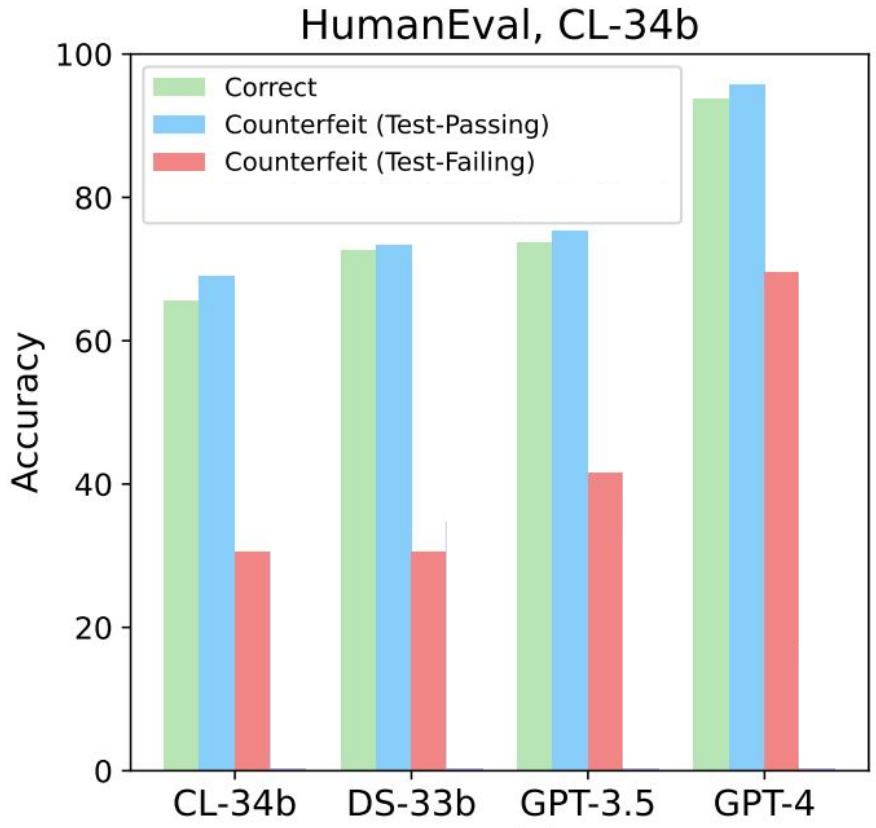
GPT-4 is an exception!

Execution Prediction



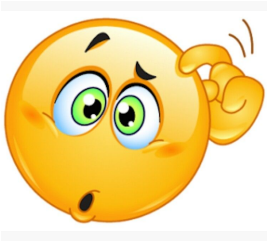
Execution Prediction

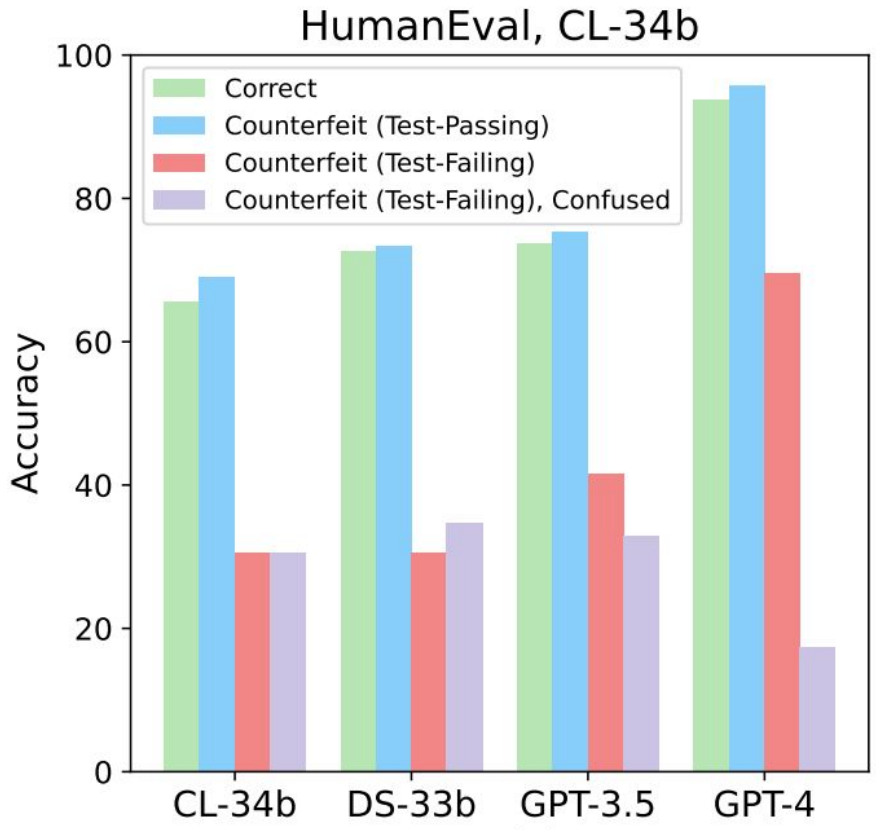




models are much better at predicting execution of test-passing samples

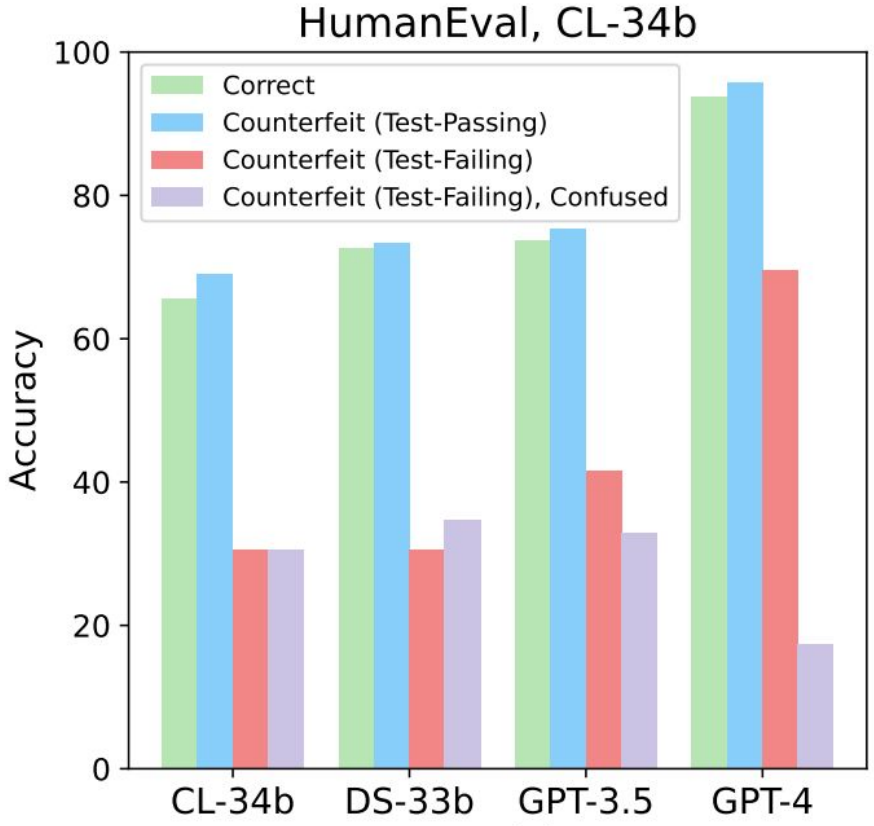
Execution Prediction

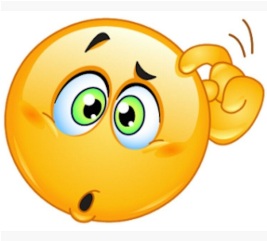




models are much better at predicting execution of test-passing samples

Execution Prediction

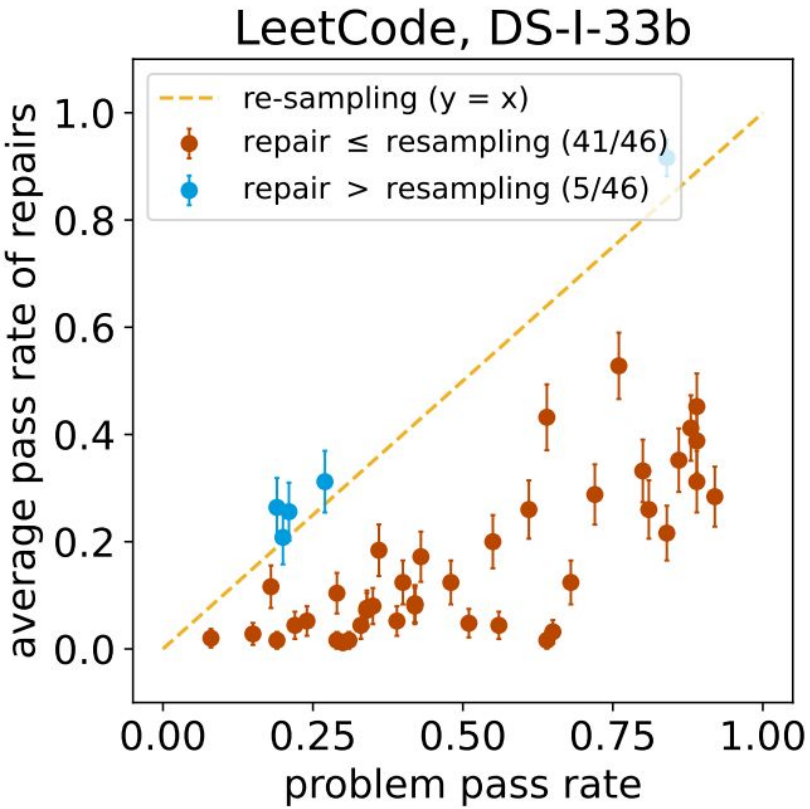




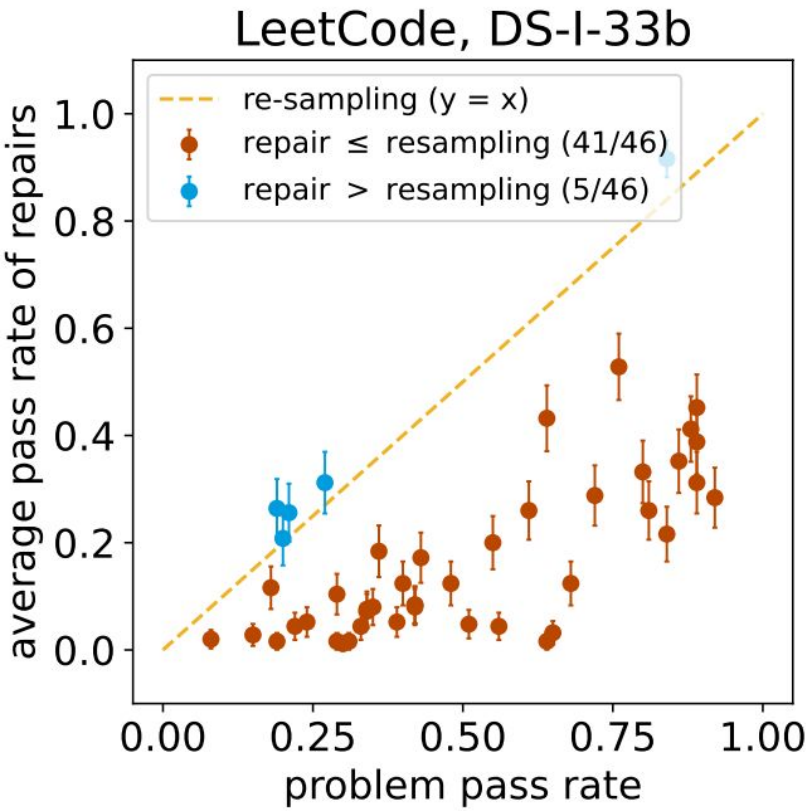
models are much better at predicting execution of test-passing samples

often, models hallucinate the semantics of counterfeit programs as correct ones!

Program Repair



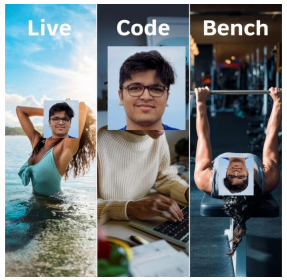
Program Repair



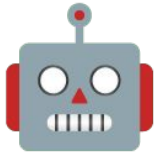
repairing counterfeit programs is generally even harder than sampling from scratch!

Focusing on the Right Metrics Without Bias is Crucial

Metric	***Bias Uncovered***
Code Understanding (correctness checking, execution, repair)	*Obviously incorrect programs vs. counterfeit programs*



LiveCodeBench

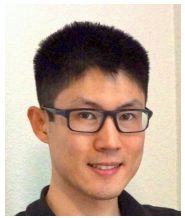

Holistic and Contamination Free Evaluation of Large Language Models for Code

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, Ion Stoica










LiveCodeBench

Holistic, time-indexed evaluation of algorithmic problems

2956. Find Common Elements Between Two Arrays

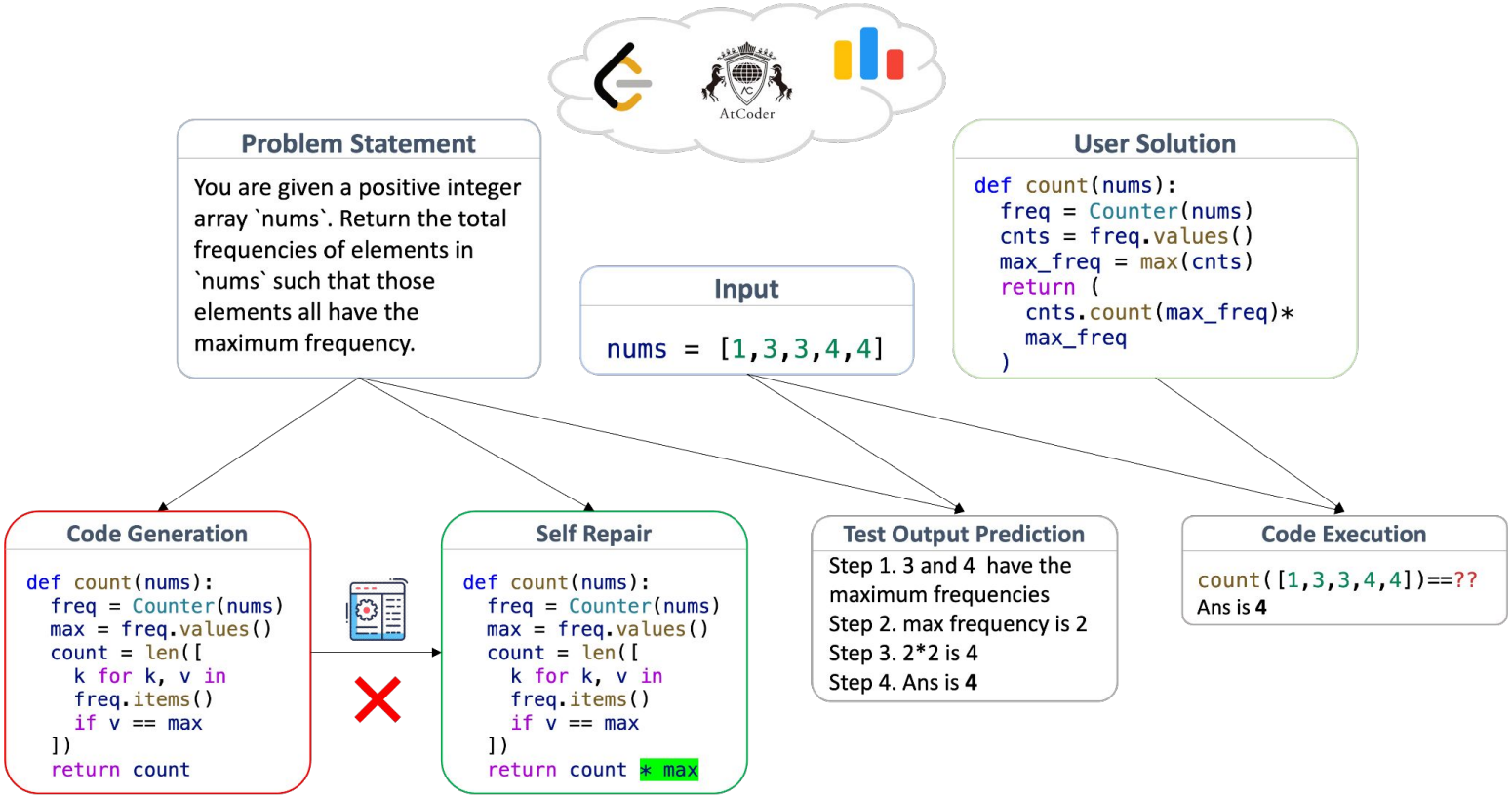
You are given two **0-indexed** integer arrays `nums1` and `nums2` of sizes `n` and `m`, respectively.

Consider calculating the following values:

- The number of indices `i` such that `0 <= i < n` and `nums1[i]` occurs **at least** once in `nums2`.
- The number of indices `i` such that `0 <= i < m` and `nums2[i]` occurs **at least** once in `nums1`.

Return *an integer array* `answer` *of size* `2` *containing the two values* ***in the above order***.

Perform Holistic Evaluations



Three Lessons from Evaluating Language Models on Reasoning Tasks - Alex Gu

Time as a Control Knob

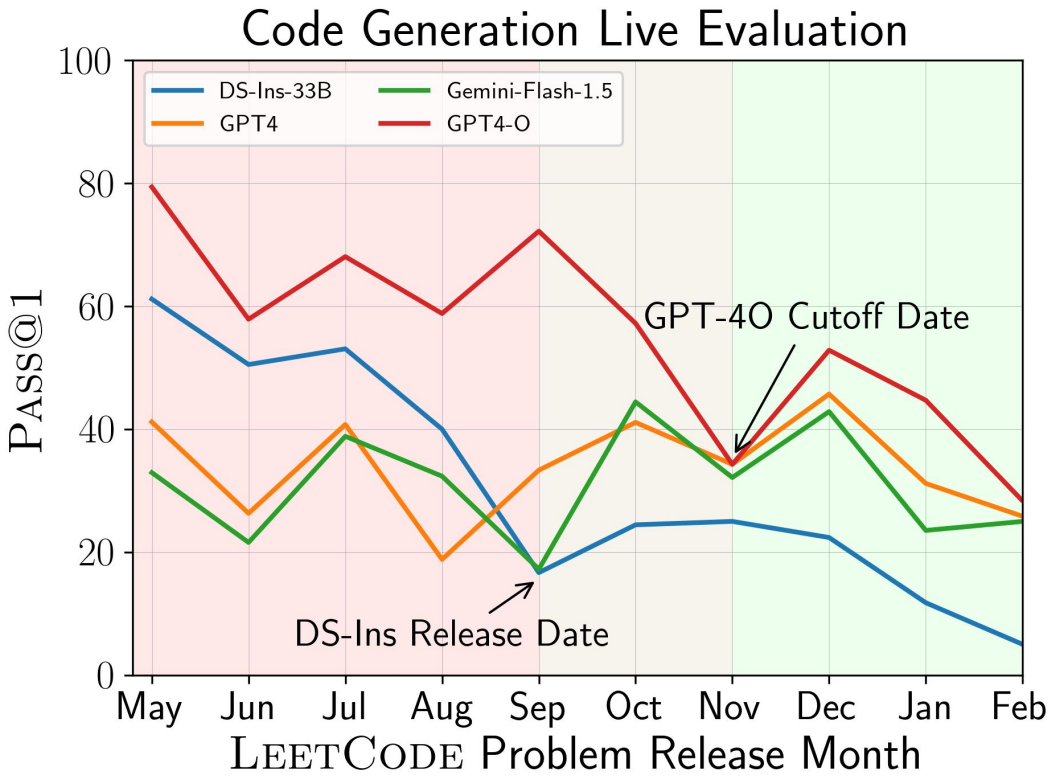
511 problems selected in the current time window.You can change start or end date to change the time window.

We estimate cutoff dates based on release date and performance variation. Feel free to adjust the slider to see the leaderboard at different time windows. Please offer feedback if you find any issues!

5/1/2023 — 6/1/2024

Rank	Model	Pass@1 ↓	Easy-Pass@1	Medium-Pass@1
	Claude-3.5-Sonnet	57.1	88.4	55.9
	GPT-4O-2024-05-13	48.8	89.2	39.2
	DeepSeekCoder-V2	45.9	82.7	37.5
1	GPT-4-Turbo-2024-04-09	43.9	81.5	34.5
2	GPT-4-Turbo-1106	38.8	80.3	24
3	Gemini-Pro-1.5-May	37	76.9	21.8
4	Claude-3-Opus	35.9	78.5	18.3
5	GPT-4-0613	35.6	75	21
	Codestral-Latest	34.1	70.9	20.8
	Qwen2-Ins-72B	33.7	66.4	23.3

Evidence of Contamination



Focusing on the Right Metrics Without Bias is Crucial

Metrics

code generation, code repair, execution, test output prediction

Bias Uncovered

Contamination

Focusing on the right metrics without bias is crucial.

Right Metrics

CRUXEval

SlopCodeBench

IneqMath

Without Bias

Counterfeit Conundrum

LiveCodeBench

1. Focusing on the Right Metrics Without Bias is Crucial

2. Looking Beyond the Score Reveals Hidden Nuances

3. What We Measures Shapes What We Build

CRUXEval

Input Prediction (I)

Find an input producing a given output

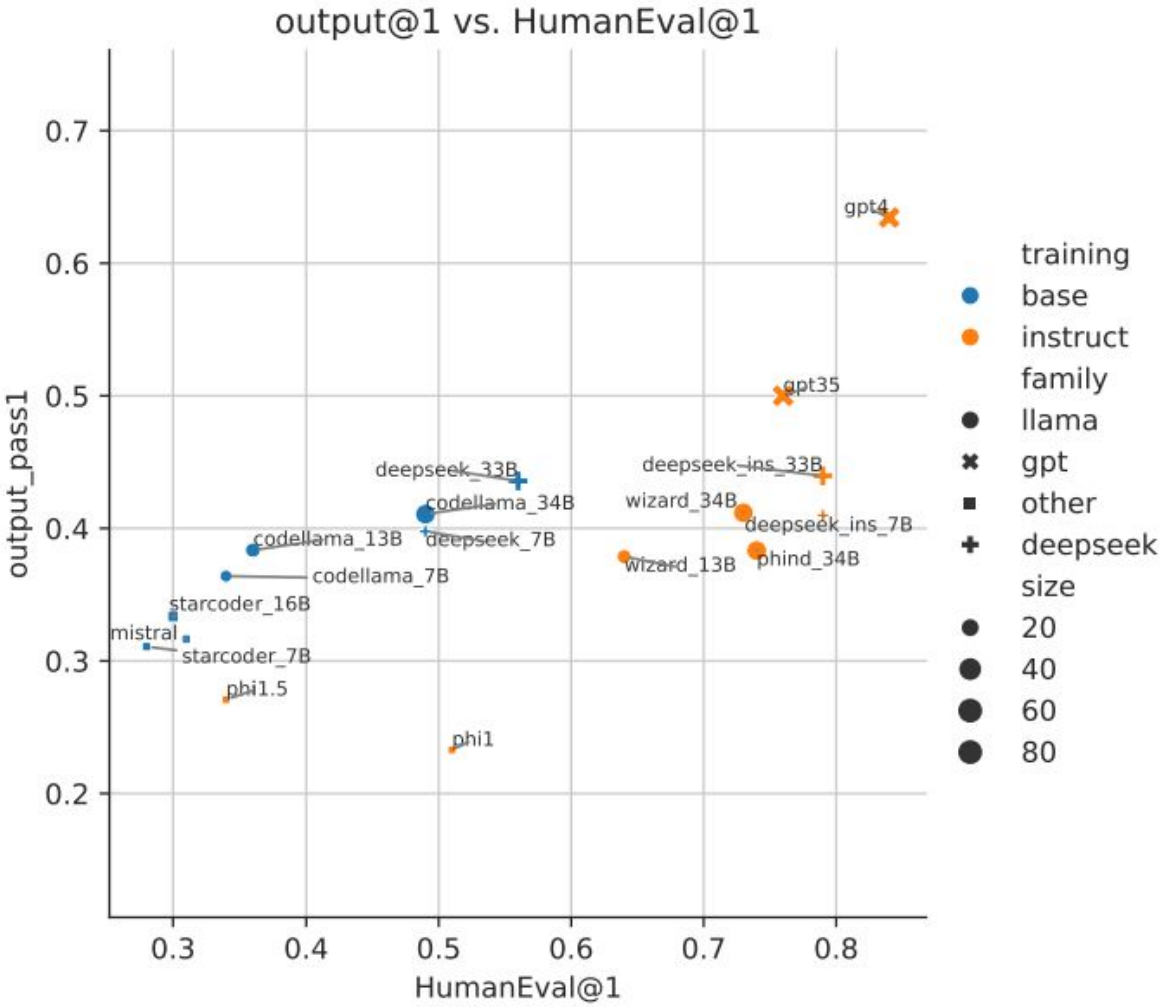
(code understanding + reasoning)

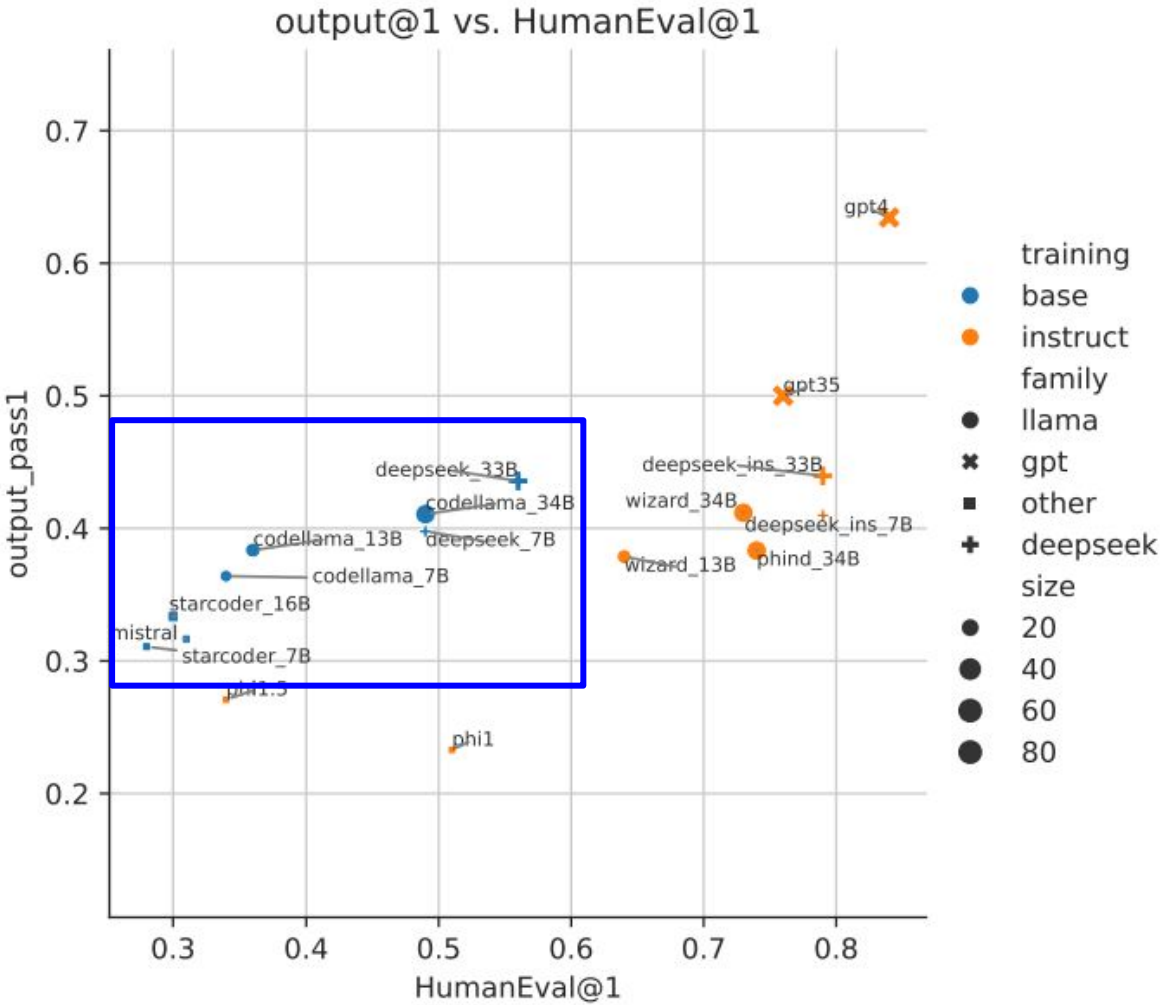
Output Prediction (O)

Execute the function on a given input

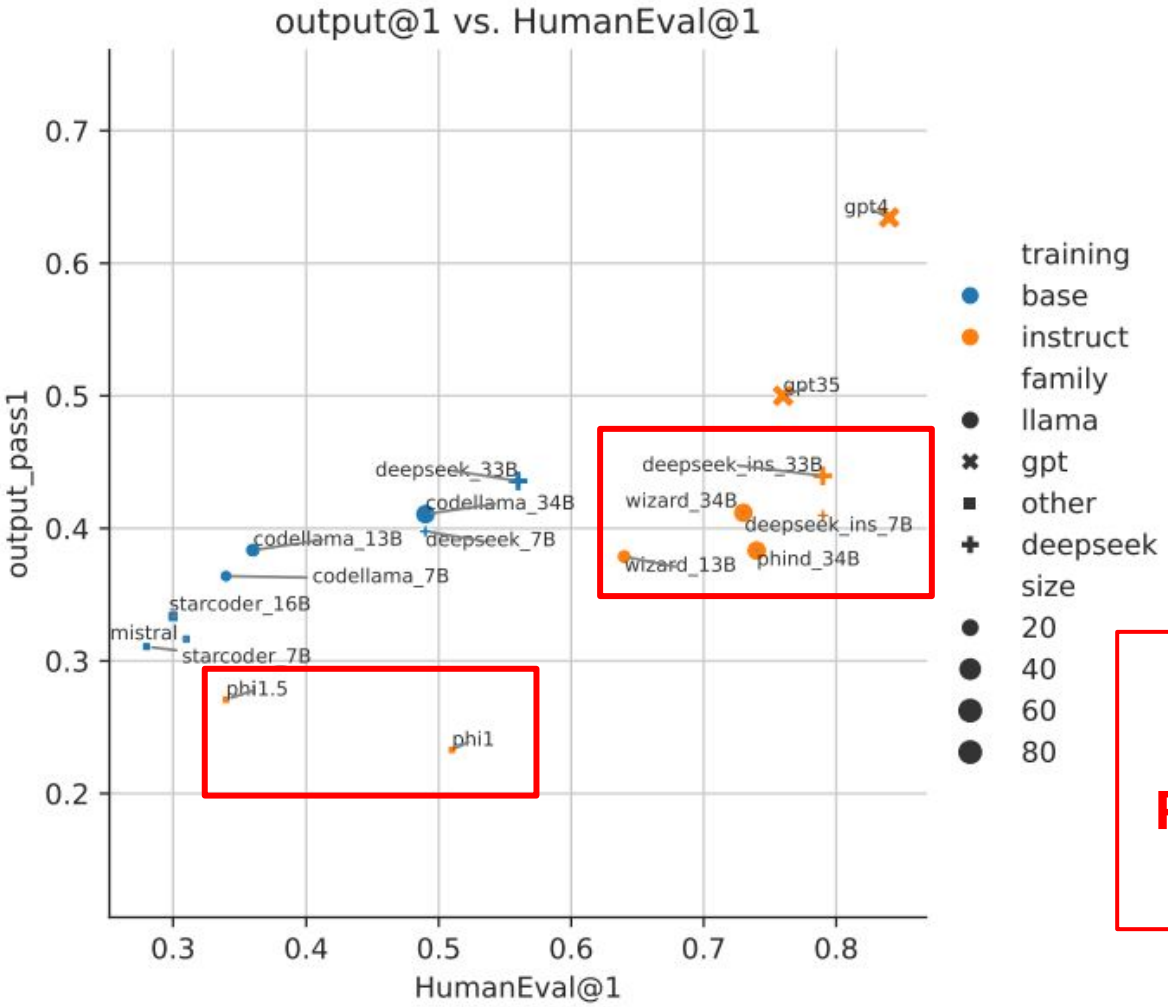
(code execution)

```python
def f(string):
    string_x = string.rstrip("a")
    string = string_x.rstrip("e")
    return string

# output prediction, CRUXEval-O
assert f("xxxxaaee") == ??
## GPT4: "xxxx", incorrect

# input prediction, CRUXEval-I
assert f(??) == "xxxxaa"
## GPT4: "xxxxaae", correct
```

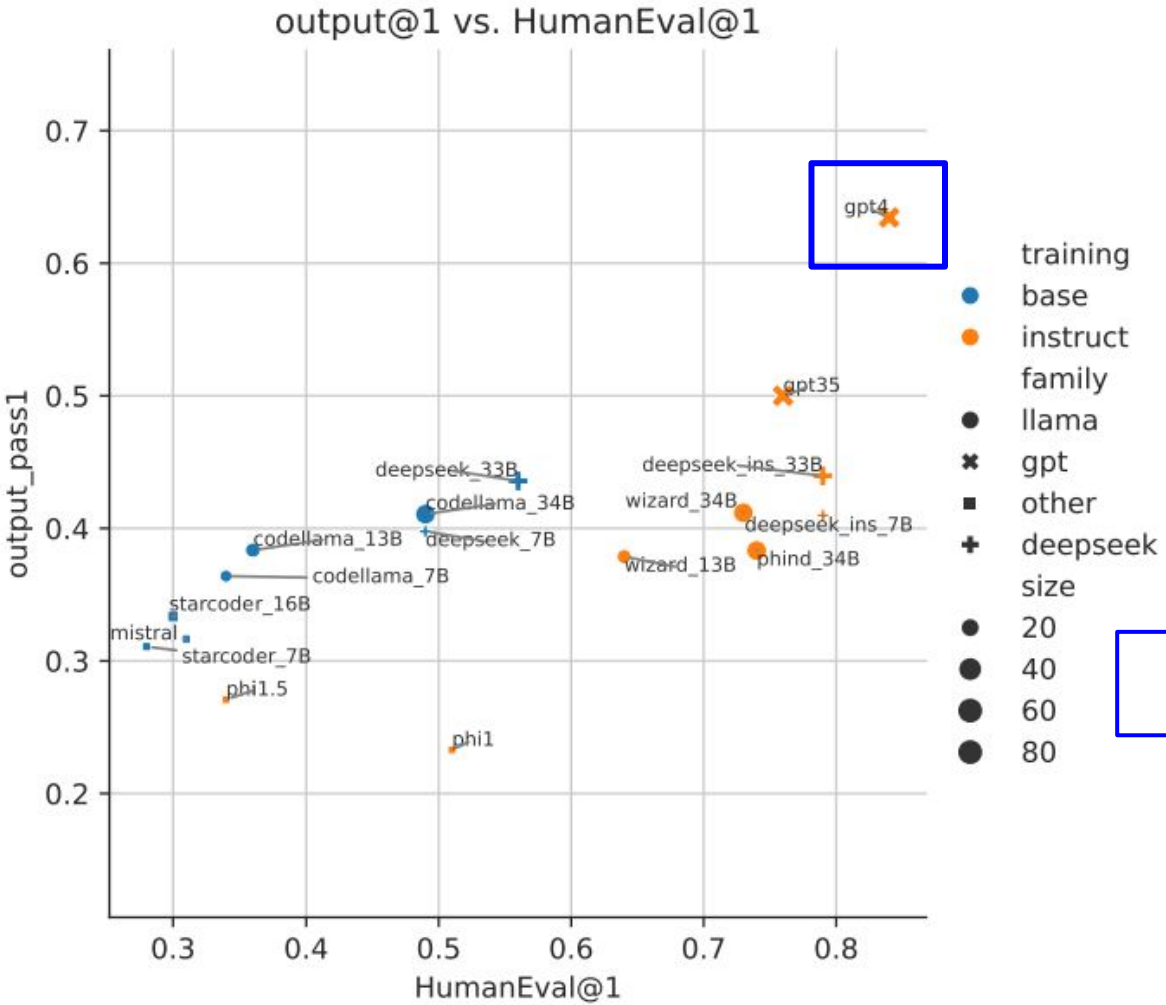

output@1 vs. HumanEval@1
output_pass1
HumanEval@1
0.2
0.3
0.4
0.5
0.6
0.7
0.3
0.4
0.5
0.6
0.7
0.8
gpt4
gpt35
deepseek_33B
codellama_34B
deepseek_ins_33B
wizard_34B
deepseek_ins_7B
codellama_13B
deepseek_7B
wizard_13B
phind_34B
codellama_7B
starcoder_16B
mistral
starcoder_7B
phi1.5
phi1
training
base
instruct
family
llama
gpt
other
deepseek
size
20
40
60
80



Base models show a correlation.

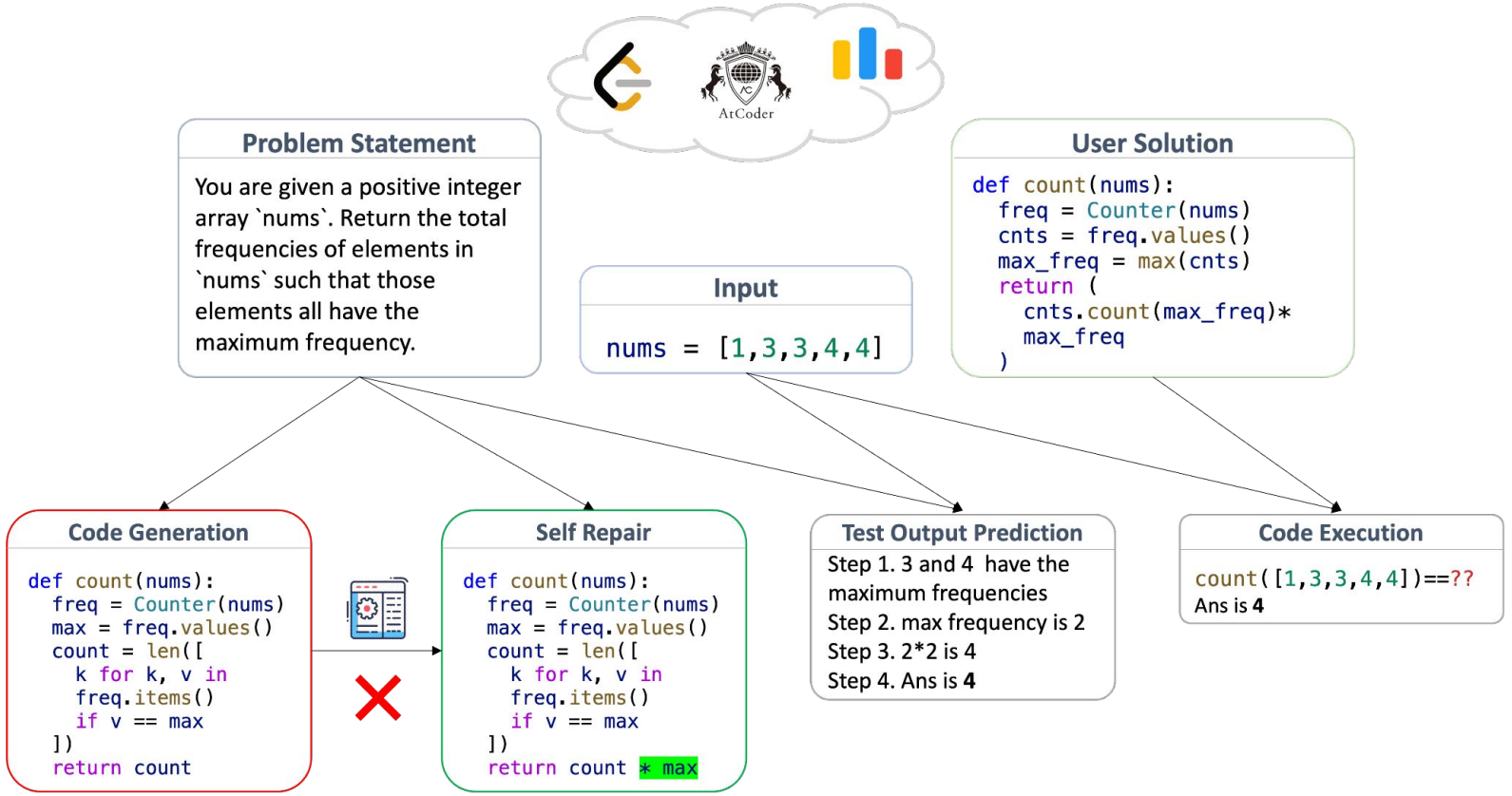


Models distilled on GPT data (WizardCoder, Phind, Phi) do NOT perform better than their base models!



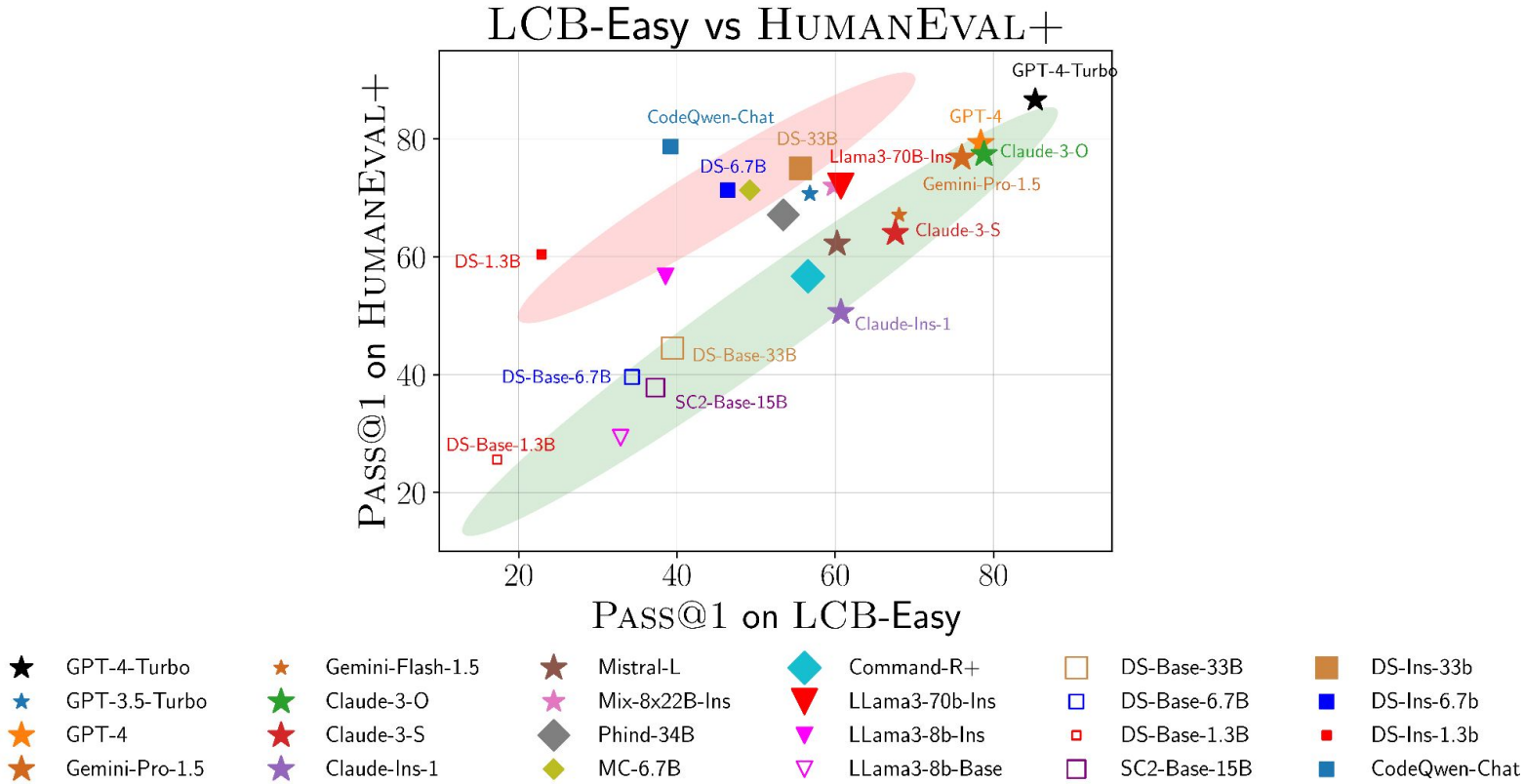
GPT-4 is a clear outlier!

LiveCodeBench

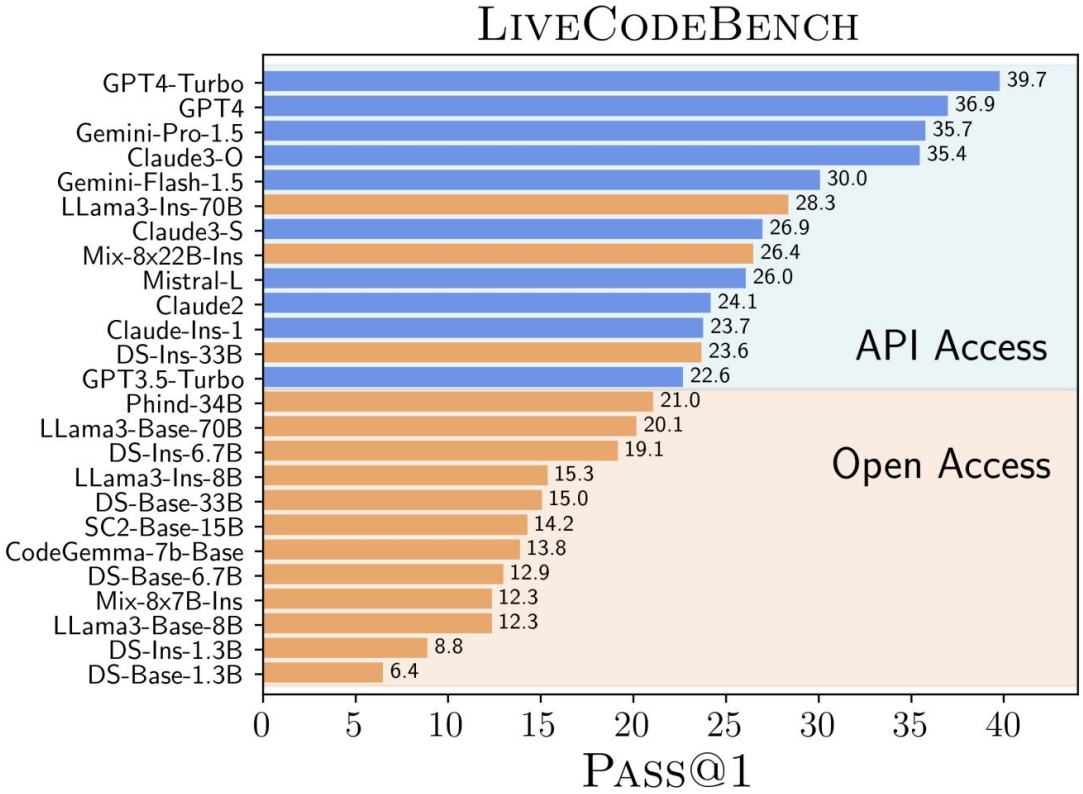


Three Lessons from Evaluating Language Models on Reasoning Tasks - Alex Gu

Detecting Overfitting



Open Models vs. Closed Models



Looking Beyond the Score Reveals Hidden Nuances

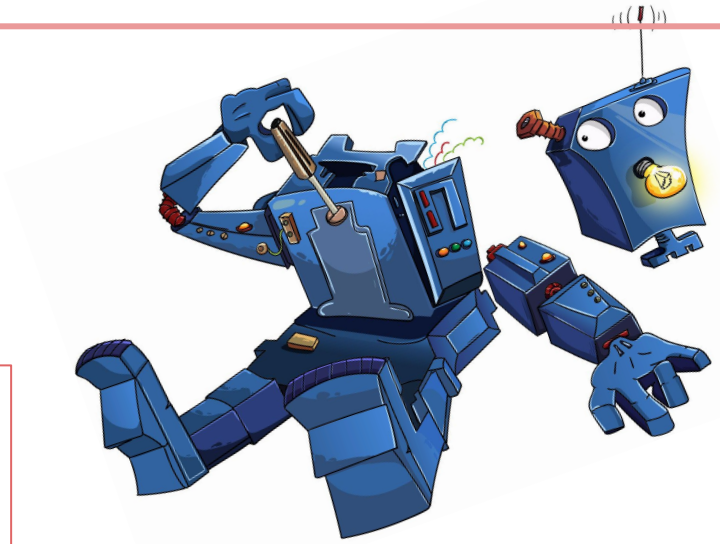
Hidden Nuances

Overfitting to HumanEval

Open vs. Closed Models

The Counterfeit Conundrum

counterfeit sample: **incorrect code generated by a model that passes weak correctness checks**



```
def same_length(s):
  if len(s) < 2:
    return False
  return True
```

```
def same_length(s):
  for a in s:
    for b in s:
      if len(a) == len(b):
        return True
  return False
```

Are different counterfeits different?

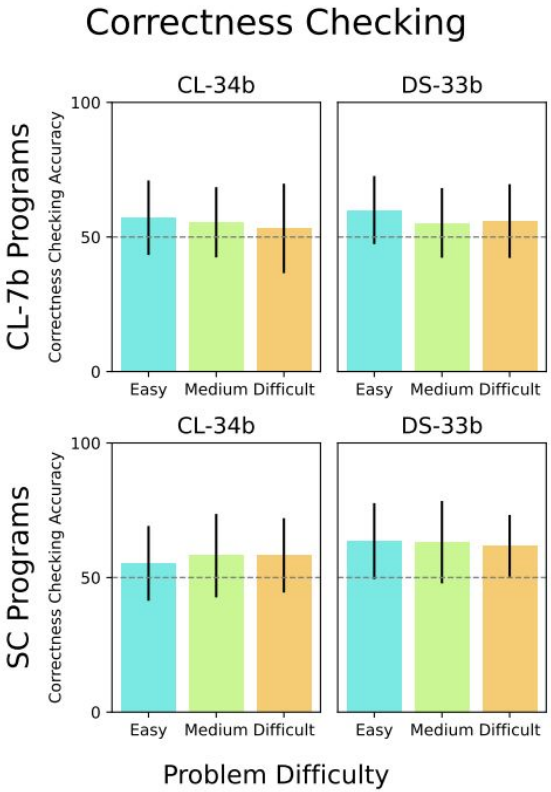
(Q1): Is it easier for models to understand counterfeit samples from problems it finds easier?

Are different counterfeits different?

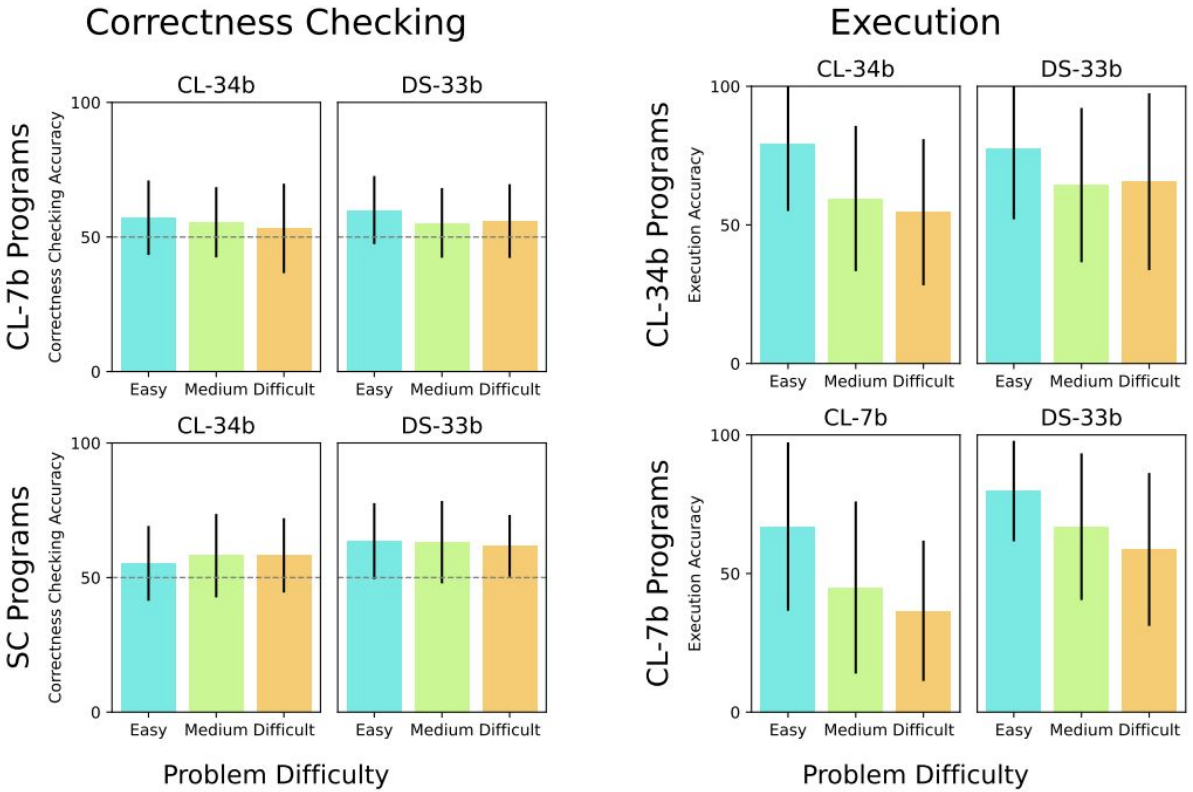
(Q1): Is it easier for models to understand counterfeit samples from problems it finds easier?

(Q2): Do models perceive their own counterfeit samples differently?

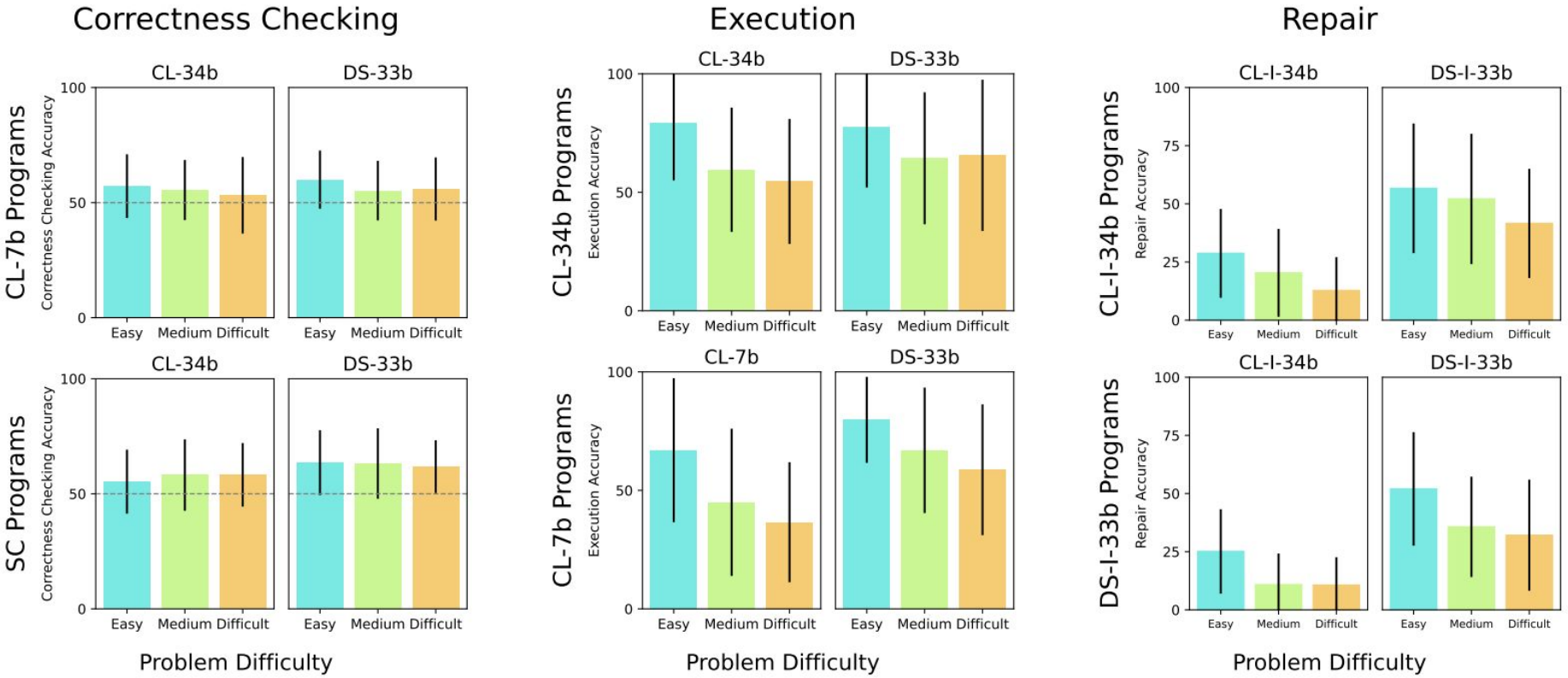
(Q1): Is it easier for models to understand counterfeit samples from problems it finds easier?



(Q1): Is it easier for models to understand counterfeit samples from problems it finds easier?



(Q1): Is it easier for models to understand counterfeit samples from problems it finds easier?



(Q2): Do stronger models generate harder counterfeit samples?

No!

Models of all strengths can generate difficult counterfeit samples!

Looking Beyond the Score Reveals Hidden Nuances

Hidden Nuances

Counterfeits are only weakly correlated with problem difficulty

All models generate similarly difficult counterfeits

🔗 LINC: A Neurosymbolic Approach for Logical Reasoning by Combining Language Models with First-Order Logic Provers

Theo X. Olausson*

Alex Gu*

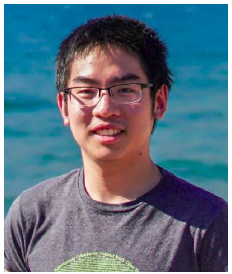
Benjamin Lipkin*

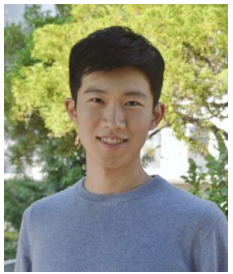
Cedegao E. Zhang*

Armando Solar-Lezama

Joshua B. Tenenbaum

Roger Levy

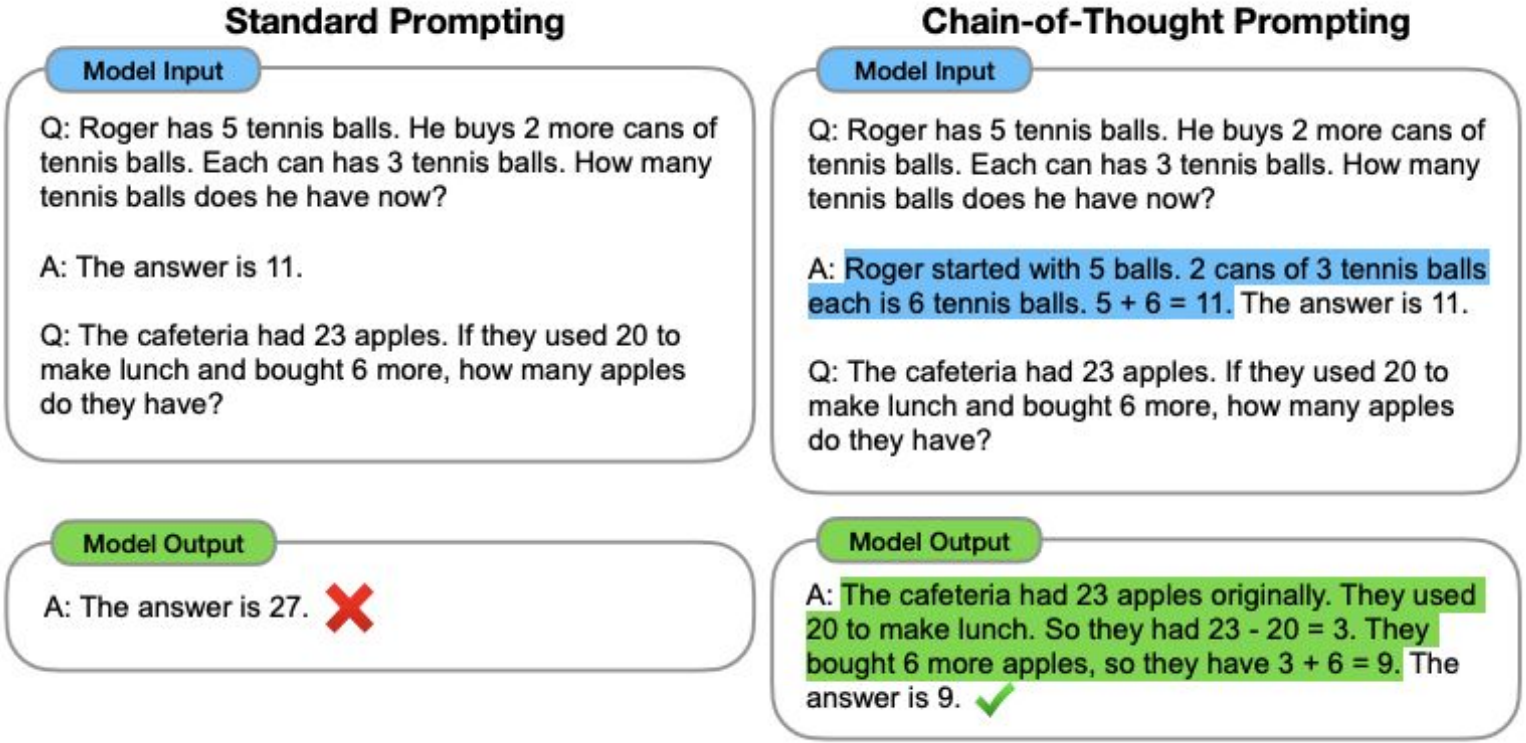



Logical Reasoning in Natural Language

Input

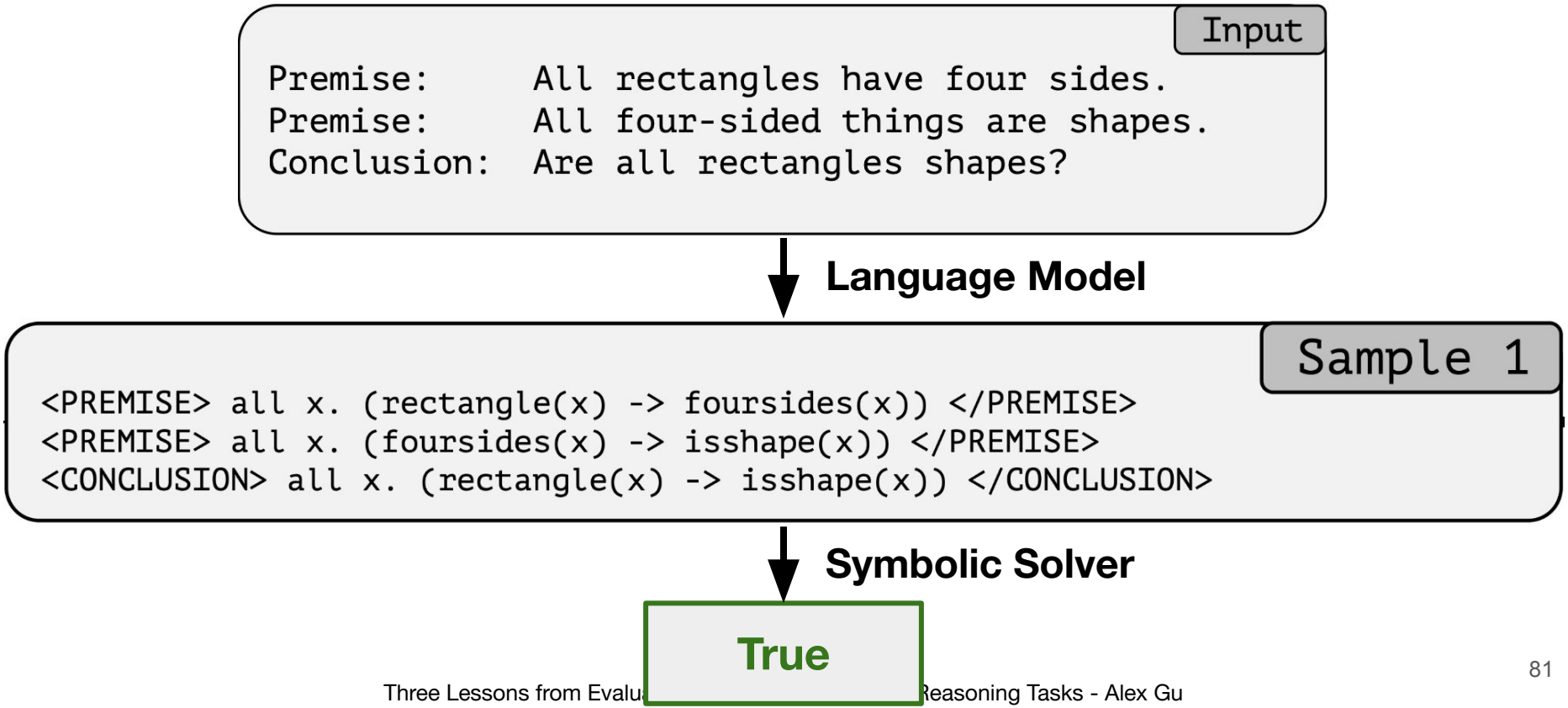
```
Premise:     All rectangles have four sides.
Premise:     All four-sided things are shapes.
Conclusion:  Are all rectangles shapes?
```

Chain of Thought (CoT)



Chain-of-Thought Prompting Elicits Reasoning in Large Language Models, Wei et al., NeurIPS 2022

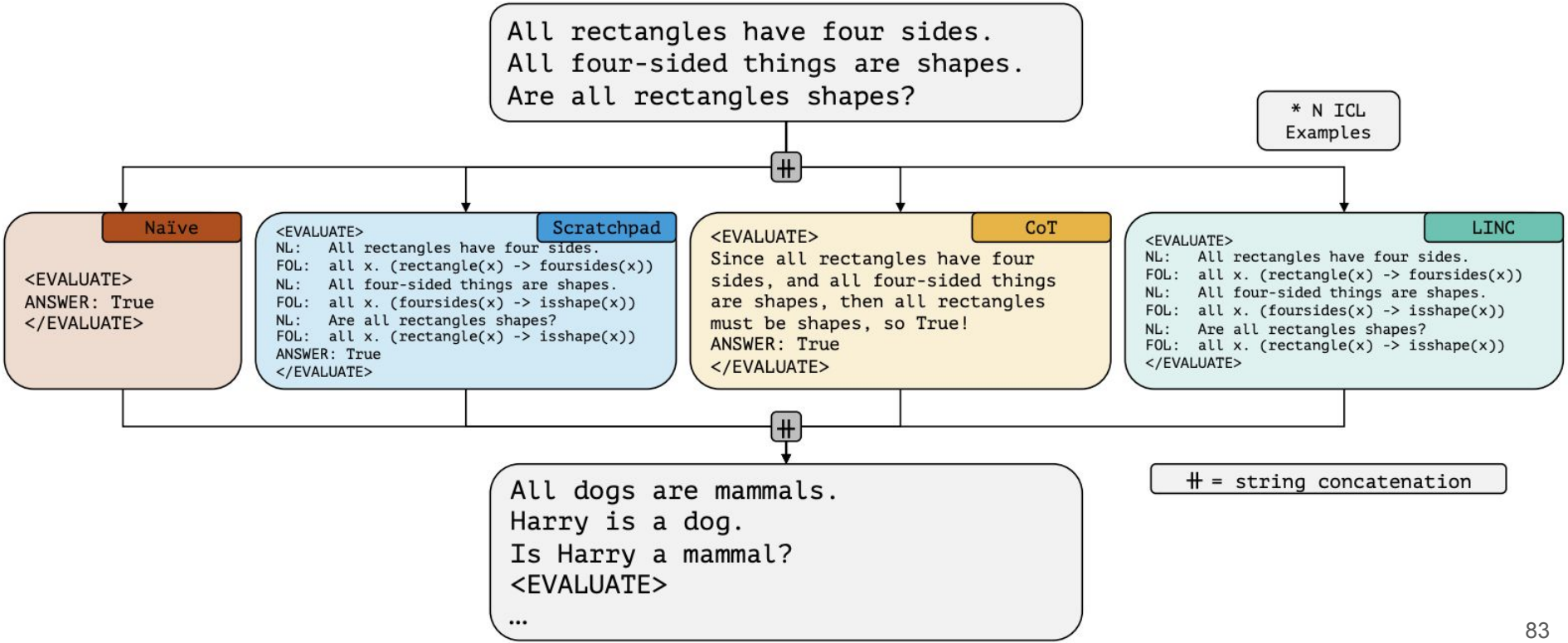
LINC: Logical Inference via Neurosymbolic Computation



LINC: Logical Inference via Neurosymbolic Computation

Using LINC, the difficulty is shifted from faithful reasoning to formalization!

Quantitative Evaluation



Datasets

ProofWriter

Premises:
The cow is big.
The cow needs the dog.
The dog sees the rabbit.
The rabbit chases the cow.
The rabbit chases the dog.
The rabbit is big.
The rabbit sees the dog.
If the cow is blue and the cow needs the rabbit then the cow needs the dog.
If the cow chases the dog then the cow sees the rabbit.
If something is big then it chases the dog.

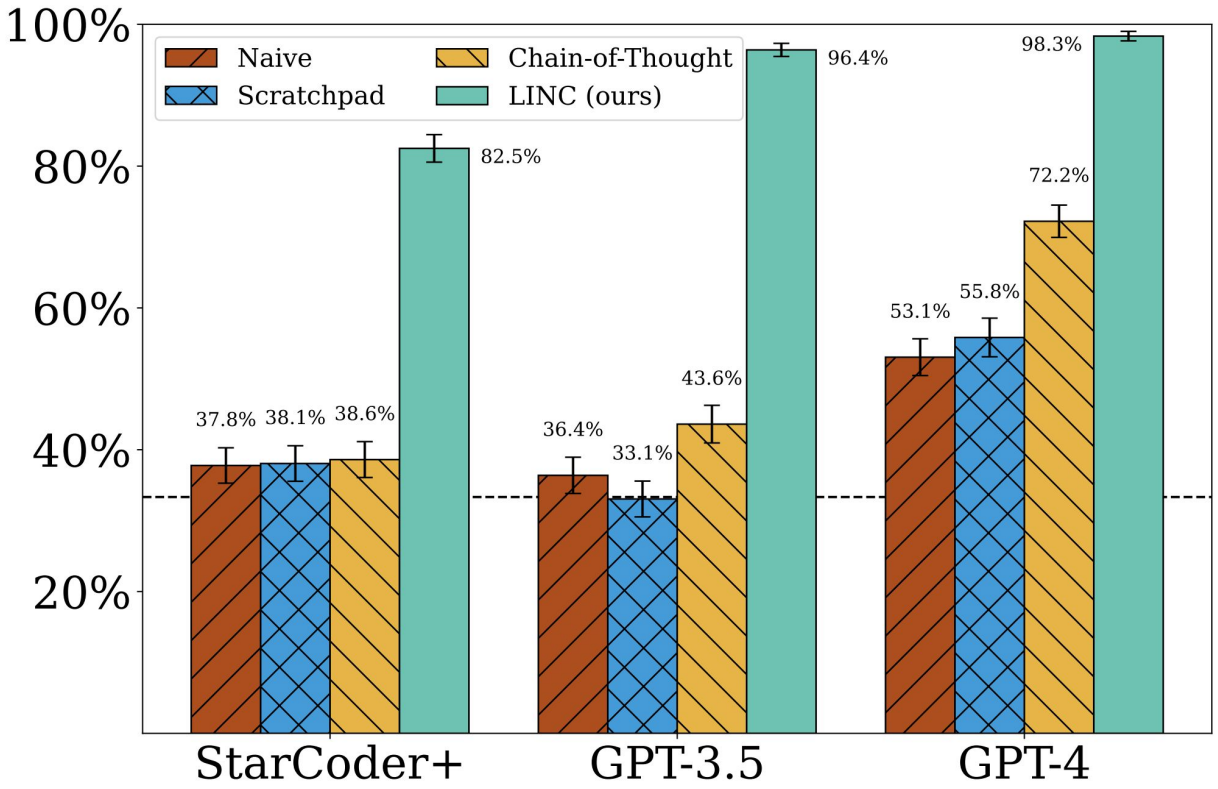
Conclusion:
The cow sees the rabbit?

FOLIO

Premises:
Some fish may sting.
Stonefish is a fish.
It stings to step on a stonefish.
Stonefish stings cause death if not treated.
To treat stonefish stings, apply heat to the affected area or use an antivenom.

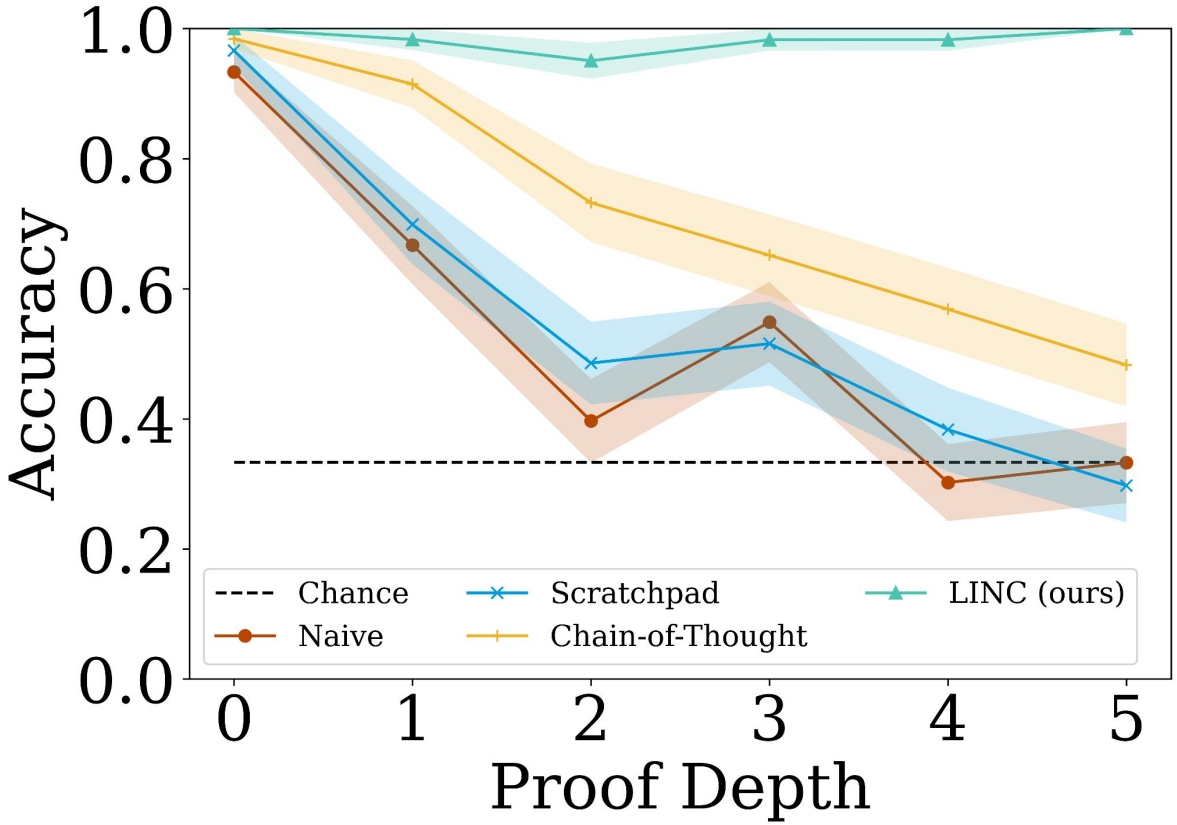
Conclusion:
Stings of some fish can cause death if not treated.

ProofWriter Results



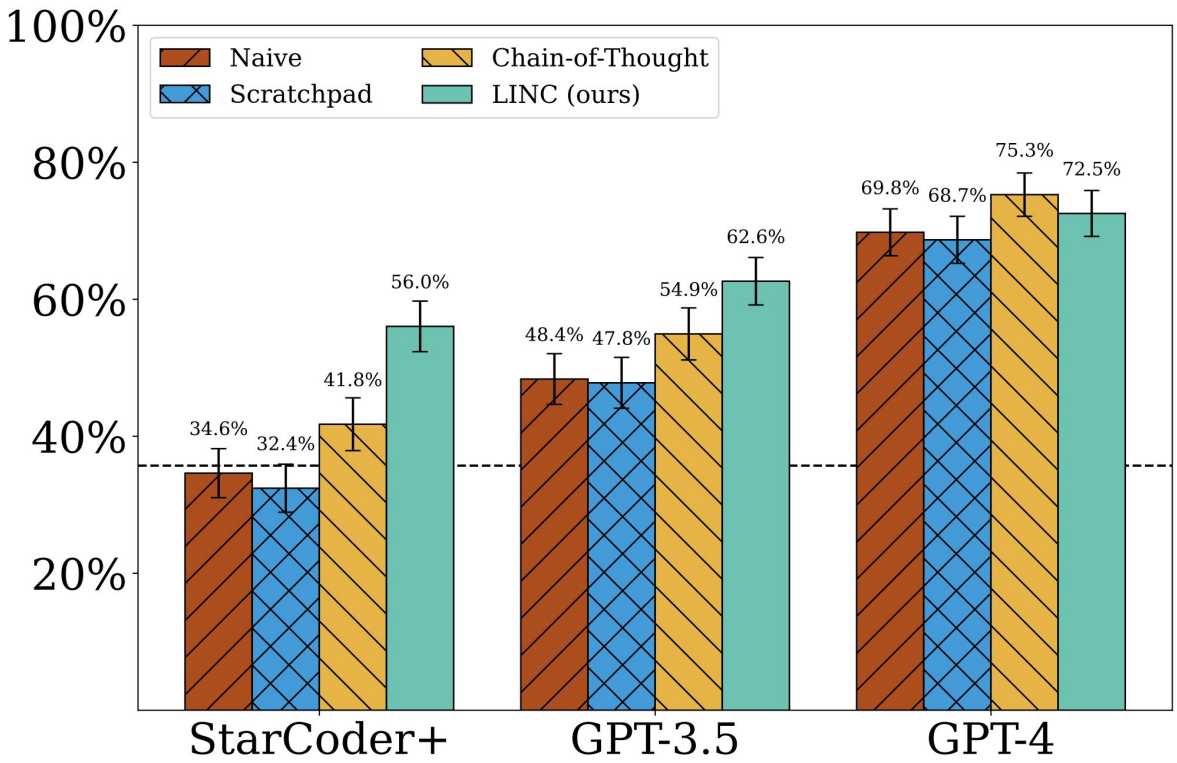
LINC shows significant improvements across the board!

ProofWriter Results



Other methods perform worse as proof depth increases, but LINC remains strong!

FOLIO Results



LINC leads to gains for StarCoder+ and GPT-3.5, but CoT is strongest for GPT-4!

Failure Modes of LINC

L1: FOL fails to capture implicit information not mentioned in the premises

Premise 1: All people reading books are happy
all x. **Person(x)** & ReadBook(x) -> Happy(x)

Premise 2: Ben and Ced each read a book.
ReadBook(Ben) & ReadBook(Ced)

Conclusion: Ben is happy
Happy(Ben)

Correct: True
Predicted: Unknown
Person(Ben)?

Failure Modes of LINC

L2: FOL fails to capture information explicitly mentioned in the premises due to the choice of representation.

Premise 1: Theo is happy and healthy
HappyAndHealthy(Theo)

Conclusion: Theo is happy
Happy(Theo)

Correct: True
Predicted: Unknown
HappyAndHealthy -> Happy?

Failure Modes of CoT

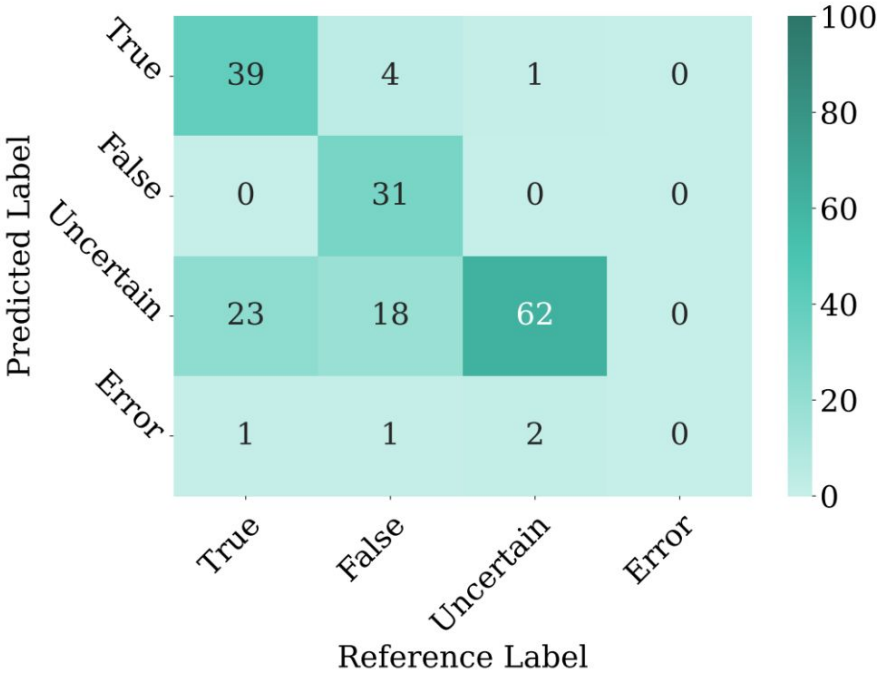
CoT concludes something different than it suggests.

CoT makes incorrect logical deductions.

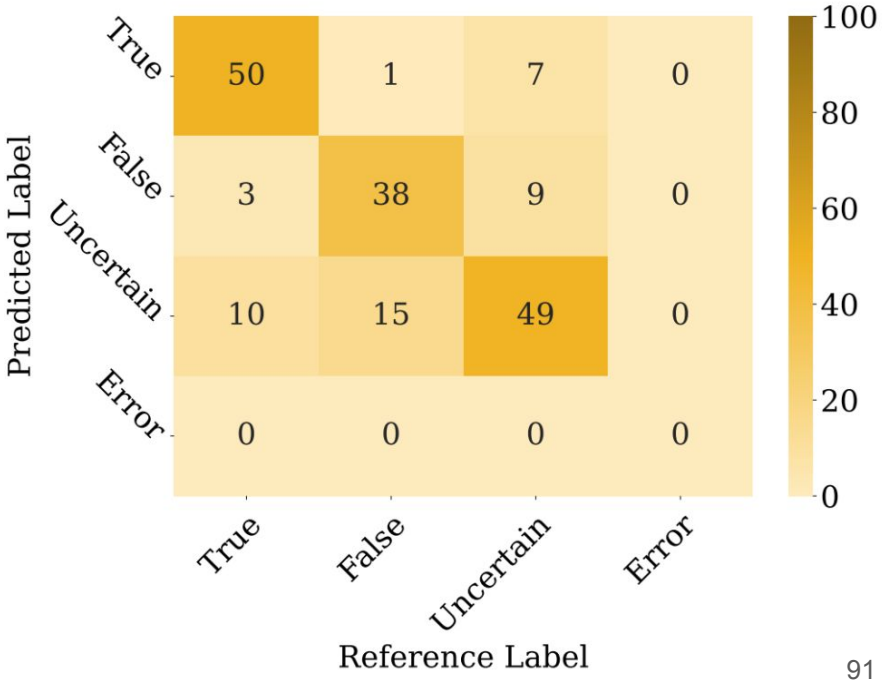
CoT fails to find complex paths of reasoning.

1. Compared to CoT, LINC has worse recall but better precision on True/False predictions.

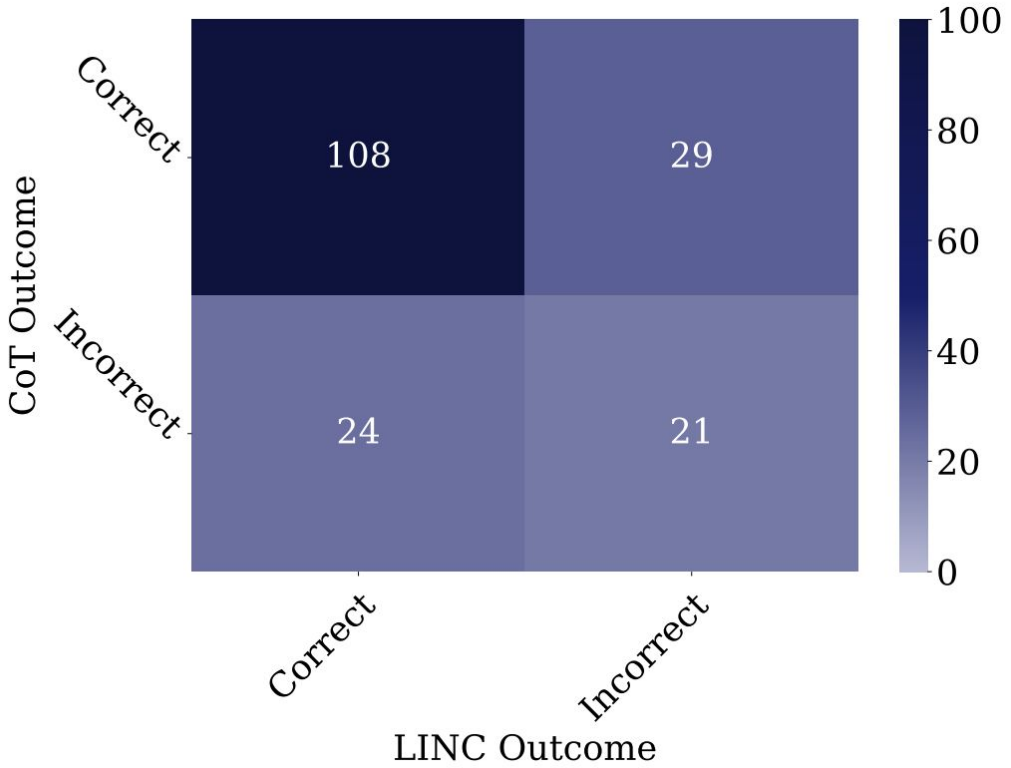
Confusion Matrix, LINC



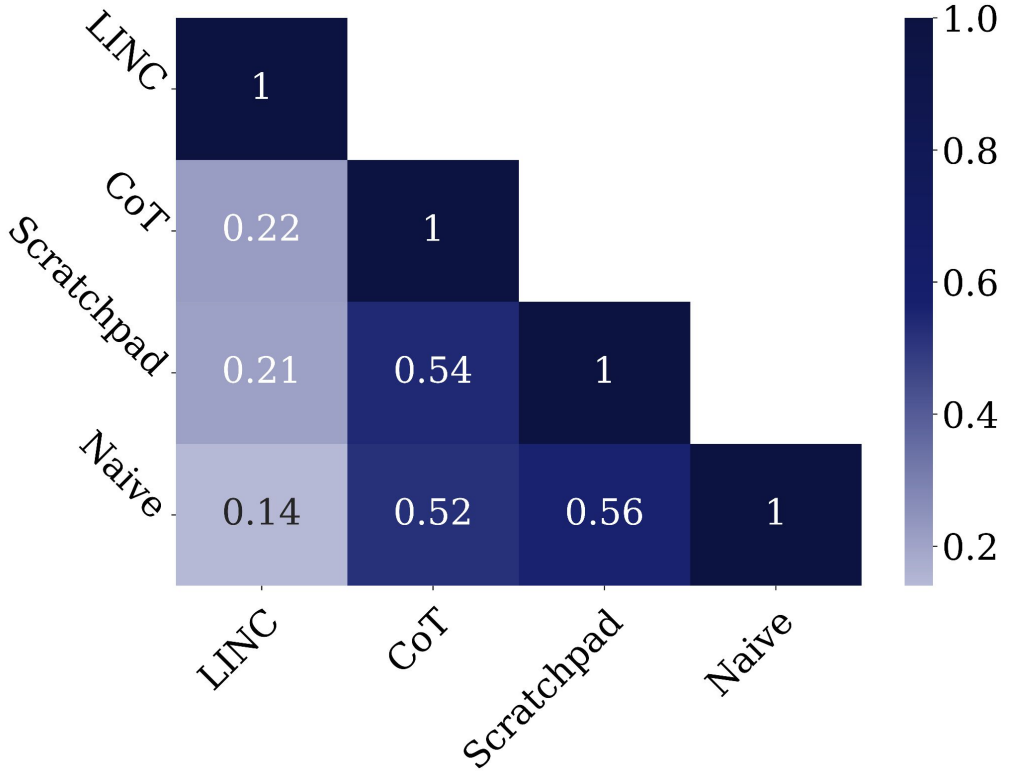
Confusion Matrix, CoT



2. LINC and CoT mispredict on different examples.



3. Mispredictions of in-context reasoning baselines are more similar to each other than they are with mispredictions of LINC.



Looking Beyond the Score Reveals Hidden Nuances

Hidden Nuance

LINC and CoT have different failure modes and mispredict differently

Looking Beyond the Score Reveals Hidden Nuances

CRUXEval

LiveCodeBench

Counterfeit Conundrum

LINC

1. Focusing on the Right Metrics Without Bias is Crucial

2. Looking Beyond the Score Reveals Hidden Nuances

3. What We Measures Shapes What We Build

CRUXEval

What We Measure: Code Execution and Reasoning

Inspiring a Family of Code Reasoning Benchmarks

Code Execution as an Explicit Target

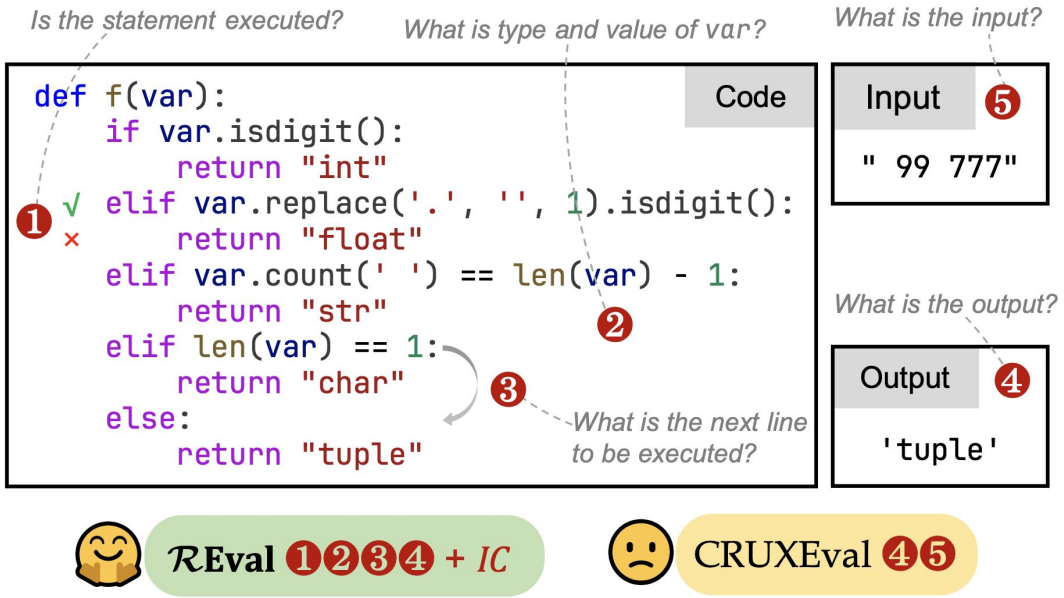
Adoption by Frontier Model Developers

Inspiring a Family of Code Reasoning Benchmarks

CRUXEval-X: translation into 19 programming languages

Figure from Xu et al., CRUXEVAL-X: A Benchmark for Multilingual Code Reasoning, Understanding and Execution

Inspiring a Family of Code Reasoning Benchmarks



REval: extending programs to runtime state prediction

Figure from [Chen et al., Reasoning Runtime Behavior of a Program with LLM: How Far Are We?]
Three Lessons from Evaluating Language Models on Reasoning Tasks - Alex Gu

Inspiring a Family of Code Reasoning Benchmarks

```
__global__ void GEMV(const float* A,
                     const float* x,
                     float* y,
                     int R,
                     int C) {

    // Calculate the row index
    // assigned to the thread
    int r = blockIdx.x * blockDim.x
            + threadIdx.x;

    // Return if out of bounds
    if (r >= R) return;
    float s = 0.0f;

    for (int c = 0; c < C; c++) {
        s += A[r * C + c] * x[c];
    }

    y[r] = s;
}
```

```
__global__ void GEMV(const float* A, const float* x,
                     float* y, int R, int C) {
    __shared__ float tile[32]; // tiling with shared memory
    int r = blockIdx.x * blockDim.x + threadIdx.x;
    bool valid = (r < R);
    float s = 0.0f;
    for (int start = 0; start < C; start += 32) {
        for (int i = threadIdx.x; i < 32; i += blockDim.x) {
            int c = start + i;
            if (c < C) tile[i] = x[c]; // load x into tile
        }
        __syncthreads();
        if (valid) {
            for (int j = 0; j < min(32, C - start); j++) {
                s += A[r * C + (start + j)] * tile[j];
            }
        }
        __syncthreads();
    }
    if (valid) y[r] = s;
}
```

EquiBench: check if two programs are equivalent

Figure from [Wei et al., EquiBench: Benchmarking Large Language Models' Reasoning about Program Semantics via Equivalence Checking]

CRUXEval

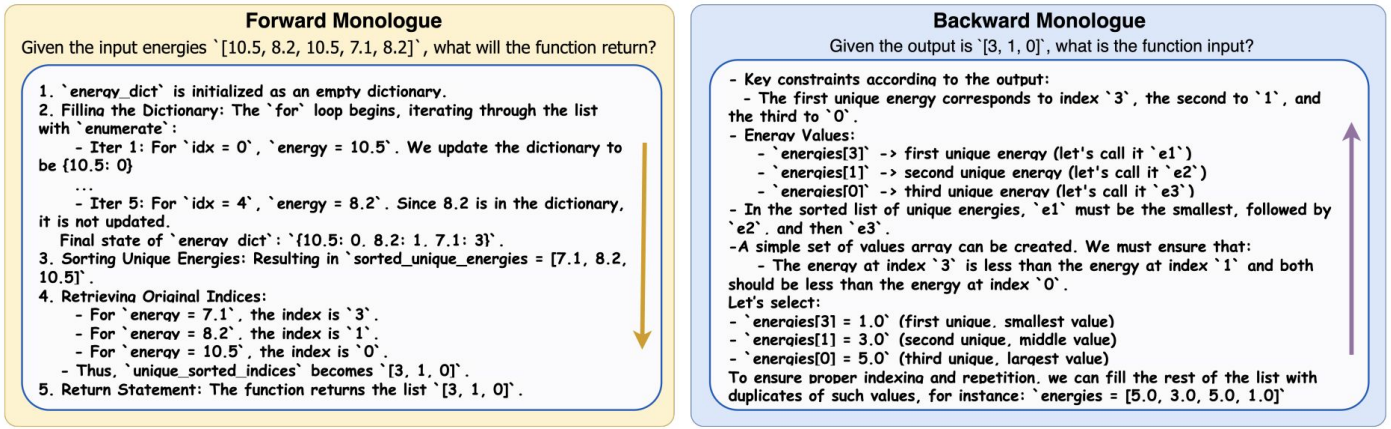
What We Measure: Code Execution and Reasoning

Inspiring a Family of Code Reasoning Benchmarks

Code Execution as an Explicit Target

Adoption by Frontier Model Developers

Code Execution as an Explicit Target



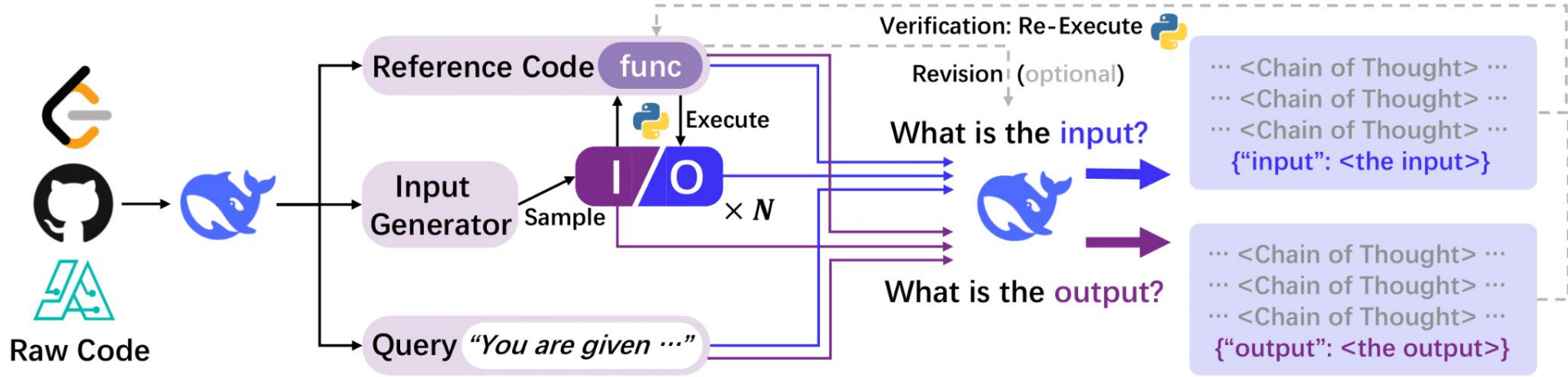
SemCoder

Code Execution as an Explicit Target

Model	Size	Code Generation			Execution Reasoning		
		HEval (+)	MBPP (+)	LCB-Lite	CXEval-I	CXEval-O	LCB-Exec
GPT-3.5-Turbo	-	76.8 (70.7)	**82.5 (69.7)**	23.9	50.3	59.0	43.6
CodeLlama-Python	13B	42.7 (38.4)	63.5 (52.6)	10.6	40.5	36.0	23.2
CodeLlama-Inst	13B	49.4 (41.5)	63.5 (53.4)	12.5	45.6	41.2	25.7
StarCoder2	15B	46.3 (37.8)	55.1 (46.1)	16.0	46.9	46.2	33.6
StarCoder2-Inst	15B	67.7 (60.4)	78.0 (65.1)	15.5	47.1	50.9	29.6
CodeLlama-Python	7B	37.8 (35.4)	59.5 (46.8)	7.1	40.4	34.0	23.0
CodeLlama-Inst	7B	36.0 (31.1)	56.1 (46.6)	10.6	36.0	36.8	30.7
StarCoder2	7B	35.4 (29.9)	54.4 (45.6)	11.6	38.2	34.5	26.3
Magicoder-CL	7B	60.4 (55.5)	64.2 (52.6)	11.4	34.0	35.5	28.6
Magicoder-S-CL	7B	70.7 (67.7)	68.4 (56.6)	12.1	42.0	35.8	30.0
DeepSeekCoder	6.7B	47.6 (39.6)	72.0 (58.7)	20.3	39.5	41.2	36.1
DeepSeekCoder-Inst	6.7B	73.8 (70.7)	74.9 (65.6)	21.1	41.9	43.2	34.0
Magicoder-DS	6.7B	66.5 (60.4)	75.4 (61.9)	25.5	45.5	41.9	38.8
Magicoder-S-DS	6.7B	76.8 (71.3)	75.7 (64.4)	23.3	44.6	43.5	38.4
SemCoder (Ours)	6.7B	73.2 (68.9)	**79.9** (65.3)	22.4	62.5	**65.1**	59.7
SemCoder-S (Ours)	6.7B	**79.3 (74.4)**	79.6 (**68.5**)	**27.5**	**63.6**	63.9	**61.2**

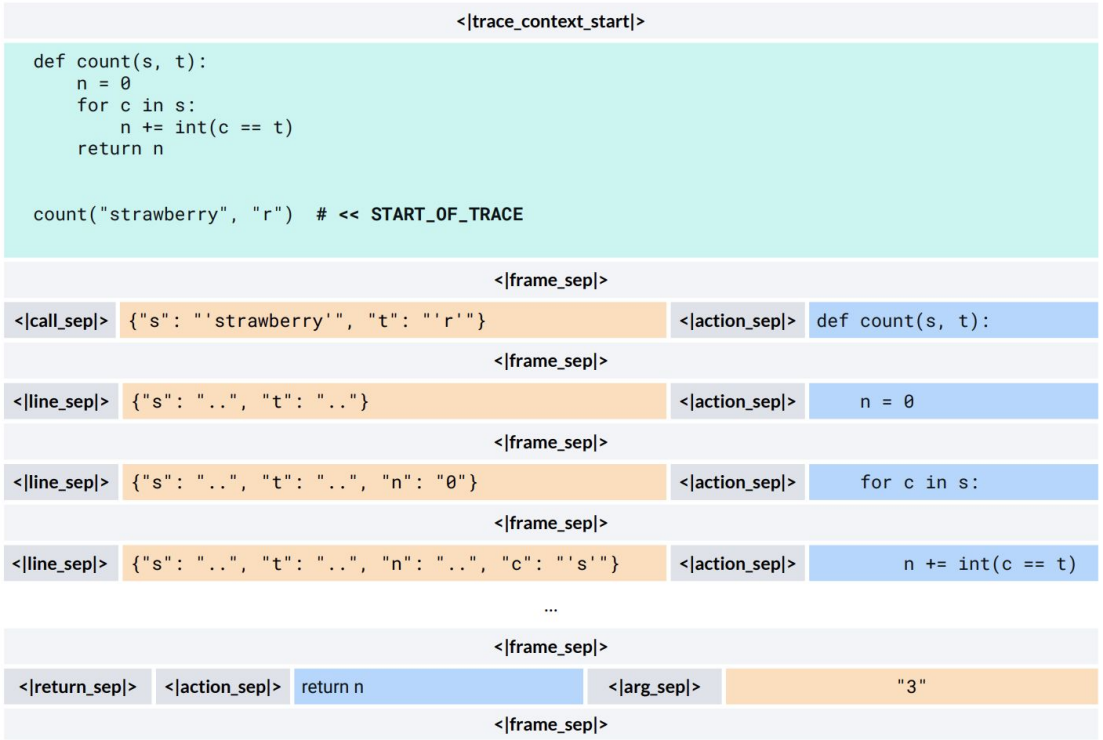
SemCoder

Code Execution as an Explicit Target



Code-IO

Code Execution as an Explicit Target



Code World Model

CRUXEval

What We Measure: Code Execution and Reasoning

Inspiring a Family of Code Reasoning Benchmarks

Code Execution as an Explicit Target

Adoption by Frontier Model Developers

Adoption by Frontier Model Developers

	Context length	HumanEval	MBPP	CruxEval-O	RepoBench	Spider	HumanEvalFIM average	HumanEval average
Codestral 22B	**32k**	**81.1%**	78.2%	**51.3%**	**34.0%**	63.5%	**91.6%**	**61.5%**
CodeLlama 70B	4k	67.1%	70.8%	47.3%	11.4%	37.0%	-	51.9%
DeepSeek Coder 33B	16k	77.4%	**80.2%**	49.5%	28.4%	60.0%	78.2%	57.6%
Llama 3 70B	8k	76.2%	76.7%	26.0%	18.4%	**67.1%**	-	61.2%
		Python				SQL	Average on several languages	

Codestral, Mistral AI

Adoption by Frontier Model Developers

Table 4: Performance comparison of Kimi-K2-Base against leading open-source models across diverse tasks.

	Benchmark (Metric)	#Shots	Kimi-K2-Base	DeepSeek-V3-Base	Llama4-Maverick-Base	Qwen2.5-72B-Base
	Architecture	-	MoE	MoE	MoE	Dense
	# Activated Params	-	32B	37B	17B	72B
	# Total Params	-	1043B	671B	400B	72B
English	MMLU	5-shots	**87.79**	87.10	84.87	86.08
	MMLU-pro	5-shots	**69.17**	60.59	63.47	62.80
	MMLU-redux	5-shots	**90.17**	89.53	88.18	87.77
	SuperGPQA	5-shots	**44.67**	39.20	38.84	34.23
	GPQA-Diamond(avg@8)	5-shots	48.11	**50.51**	49.43	40.78
	SimpleQA	5-shots	**35.25**	26.49	23.74	10.31
	TriviaQA	5-shots	**85.09**	84.11	79.25	76.03
	BBH	3-shots	**88.71**	88.37	87.10	84.09
	HellaSwag	5-shots	94.60	89.44	86.02	**95.27**
	AGIEval	-	**84.23**	81.57	67.55	76.87
	ARC-Challenge	0-shot	**95.73**	93.77	94.03	95.56
	WinoGrande	5-shots	**85.32**	84.21	77.58	84.14
Code	CRUXEval-I-cot	0-shots	**74.00**	62.75	67.13	61.12
	CRUXEval-O-cot	0-shots	**83.50**	75.25	75.88	66.13
	LiveCodeBench(v6)	1-shots	**26.29**	24.57	25.14	22.29
	EvalPlus	-	**80.33**	65.61	65.48	66.04

Kimi-K2

Adoption by Frontier Model Developers

DeepSeek-Coder-V2

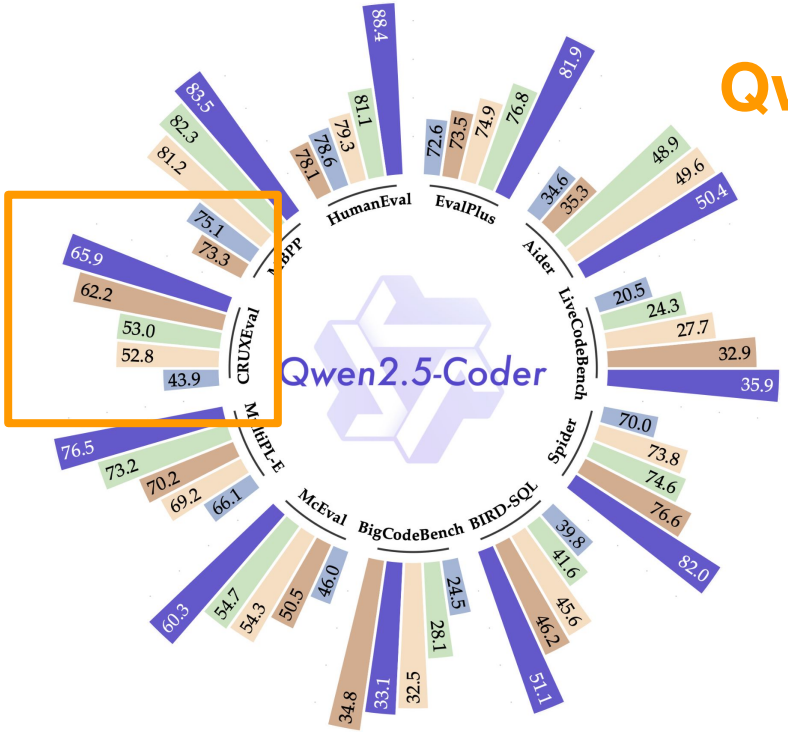
4.4. Code Understanding and Reasoning

To assess the code reasoning capabilities of our models, we utilize the CRUXEval benchmark. This benchmark comprises 800 Python functions paired with corresponding input-output examples. It is divided into two distinct tasks: CRUXEval-I, which requires the large language model (LLM) to predict the output based on the given input, and CRUXEval-O, where the model must predict the input from the known output. This structure challenges the model's ability to understand and reason through Python code in both forward and reverse directions. Table 8 presents the performance of various language models on the CruxEval benchmark, which assesses models on two metrics: CruxEval-I-COT and CruxEval-O-COT. Among the open-source models, DeepSeek-Coder-V2-Instruct stands out significantly. It scores 70.0% on CruxEval-I-COT and 75.1% on CruxEval-O-COT, showcasing its superior capability within the open-source domain. However, when compared to larger closed-source models, there is a performance gap. This performance gap may largely be attributed to the fact that DeepSeek-Coder-V2-Instruct operates with only 21 billion activation parameters, which is considerably fewer than those in larger, more advanced closed-source models like GPT-4o. This limitation in model complexity could restrict its learning and problem-solving capacities.

Adoption by Frontier Model Developers

Table 15: *Accuracy on the CRU*

Model	CRUXEval-	
	Pass@1	Pass
StarCoderBase-3B	27.1	43.
DeepSeekCoder-1.3B	27.8	44.
StableCode-3B	**33.5**	**53.**
StarCoder2-3B	32.7	50.
StarCoderBase-7B	29.7	47.
CodeLlama-7B	35.9	52.
DeepSeekCoder-6.7B	**41.9**	**62.**
StarCoder2-7B	34.6	53.
StarCoderBase-15B	31.3	49.
CodeLlama-13B	42.5	62.
StarCoder2-15B	**48.1**	**66.**
CodeLlama-34B	**47.2**	**66.**
DeepSeekCoder-33B	46.5	64.



Qwen2.5-Coder

Three Lessc

LiveCodeBench

What We Measure: Holistic, Contamination-Free Coding

Popularizing contamination-free benchmarking

Adoption by Frontier Model Developers

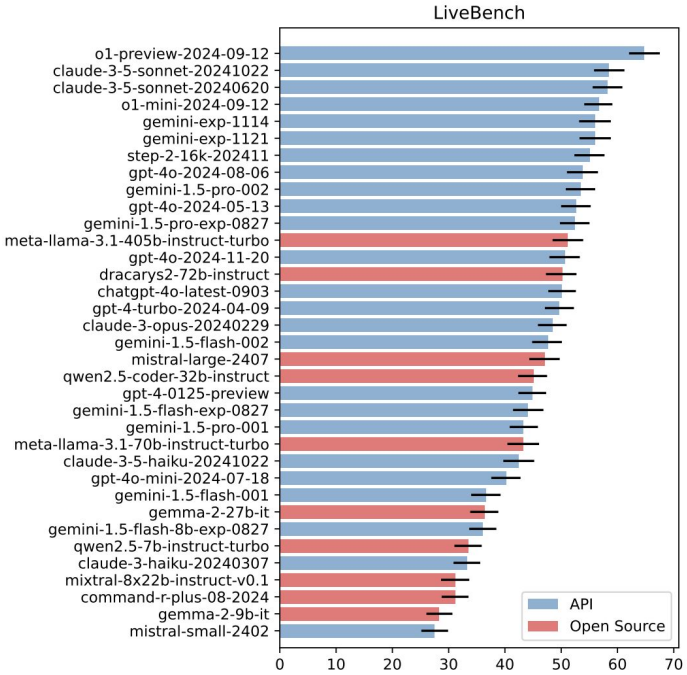
LiveCodeBench

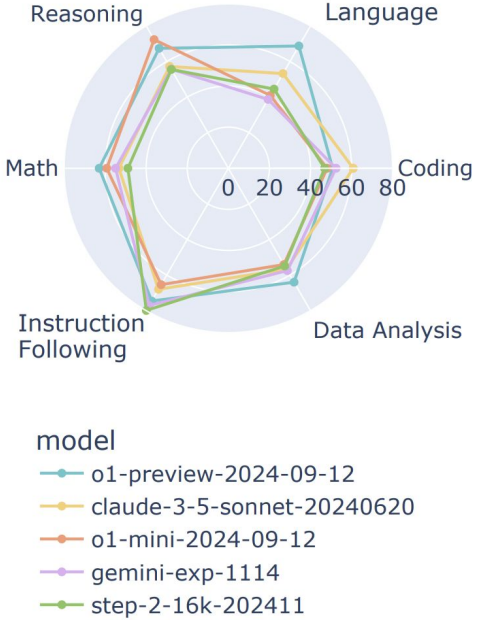
What We Measure: Holistic, Contamination-Free Coding

Popularizing contamination-free benchmarking

Adoption by Frontier Model Developers

Popularizing Contamination-Free Benchmarking





LiveBench

Figure from [White, et al., LiveBench: A Challenging, Contamination-Limited LLM Benchmark]

Popularizing Contamination-Free Benchmarking

Leveraging Online Olympiad-Level Math Problems for LLMS Training and Contamination-Resistant Evaluation

Paper | Code | Home | LiveAoPSBench Data

LiveAoPSBench

Performance over time

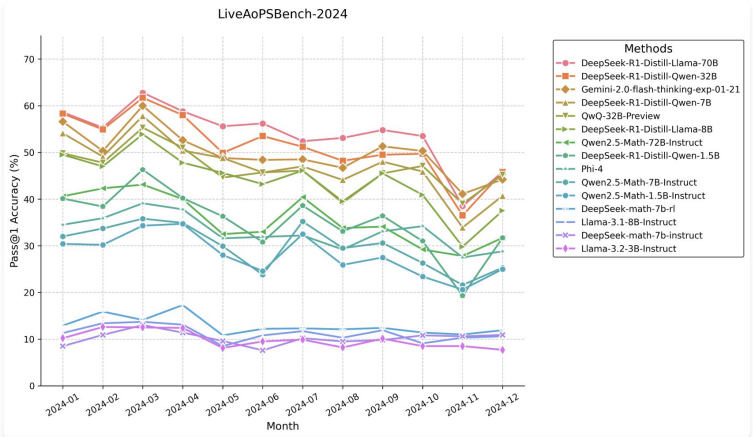


Figure from https://livemathbench.github.io/leaderboard.html

Popularizing Contamination-Free Benchmarking

LiveCodeBench Pro: How Do Olympiad Medalists Judge LLMs in Competitive Programming?

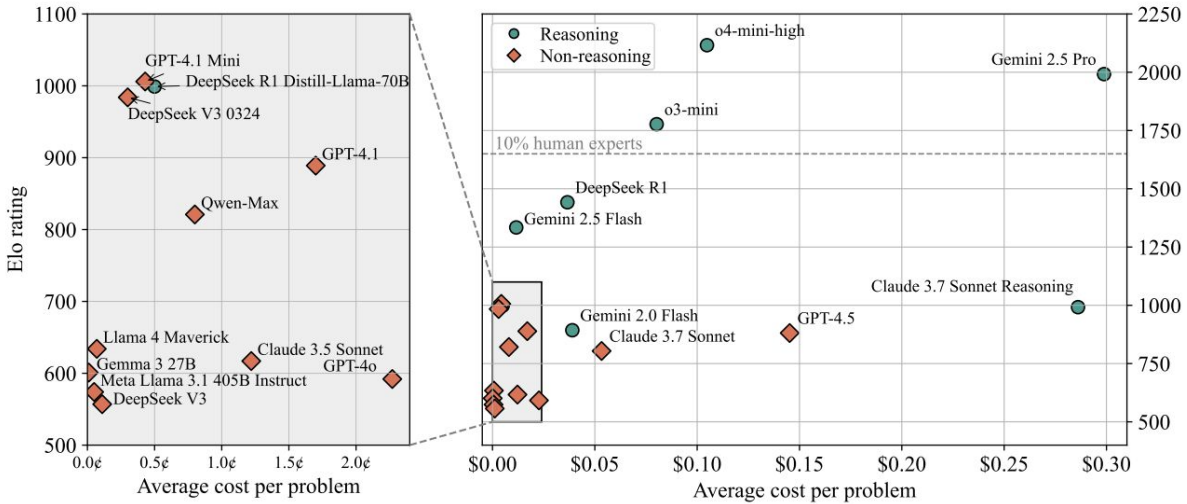


Figure from [Zheng, et al., LiveCodeBench Pro: How Do Olympiad Medalists Judge LLMs in Competitive Programming?]

Popularizing Contamination-Free Benchmarking

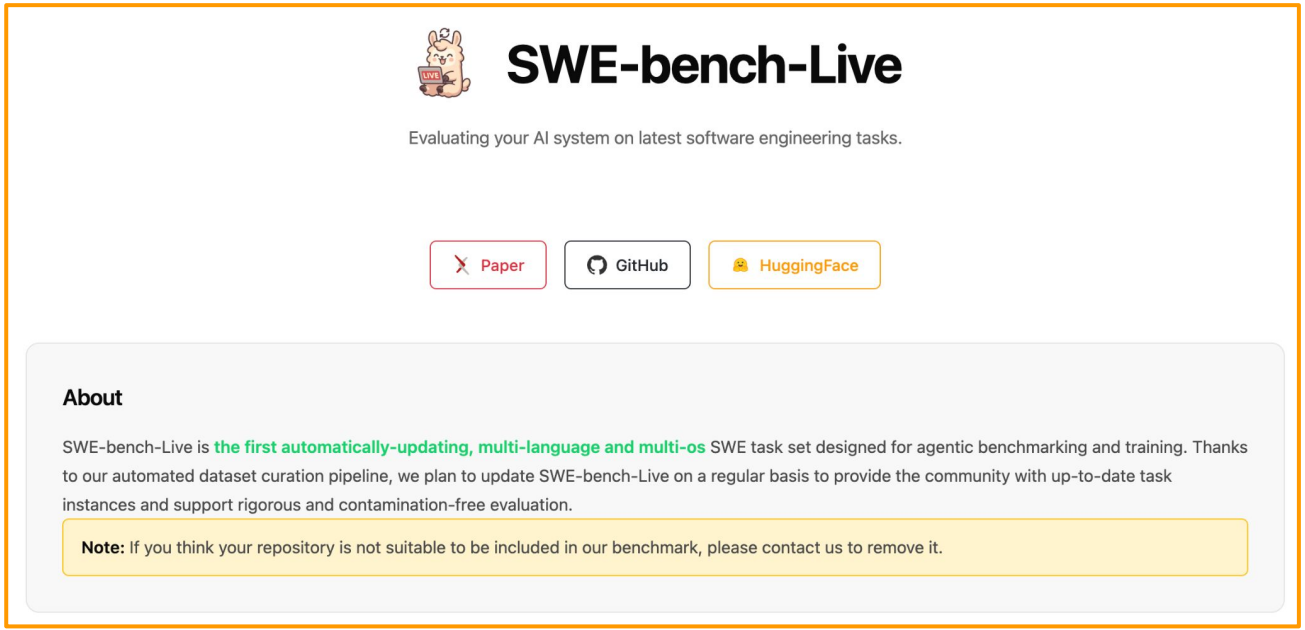


Figure from https://swe-bench-live.github.io/

LiveCodeBench

What We Measure: Holistic, Contamination-Free Coding

Popularizing contamination-free benchmarking

Adoption by Frontier Model Developers

Adoption by Frontier Model Developers

DeepSeek-V3

Model	LiveCodeBench-CoT		MATH-500	
	Pass@1	Length	Pass@1	Length
DeepSeek-V2.5 Baseline	31.1	718	74.6	769
DeepSeek-V2.5 +R1 Distill	37.4	783	83.2	1510

Table 9 | The contribution of distillation from DeepSeek-R1. The evaluation settings of LiveCodeBench and MATH-500 are the same as in Table 6.

effectiveness and robustness of the alignment process.

5.4. Discussion

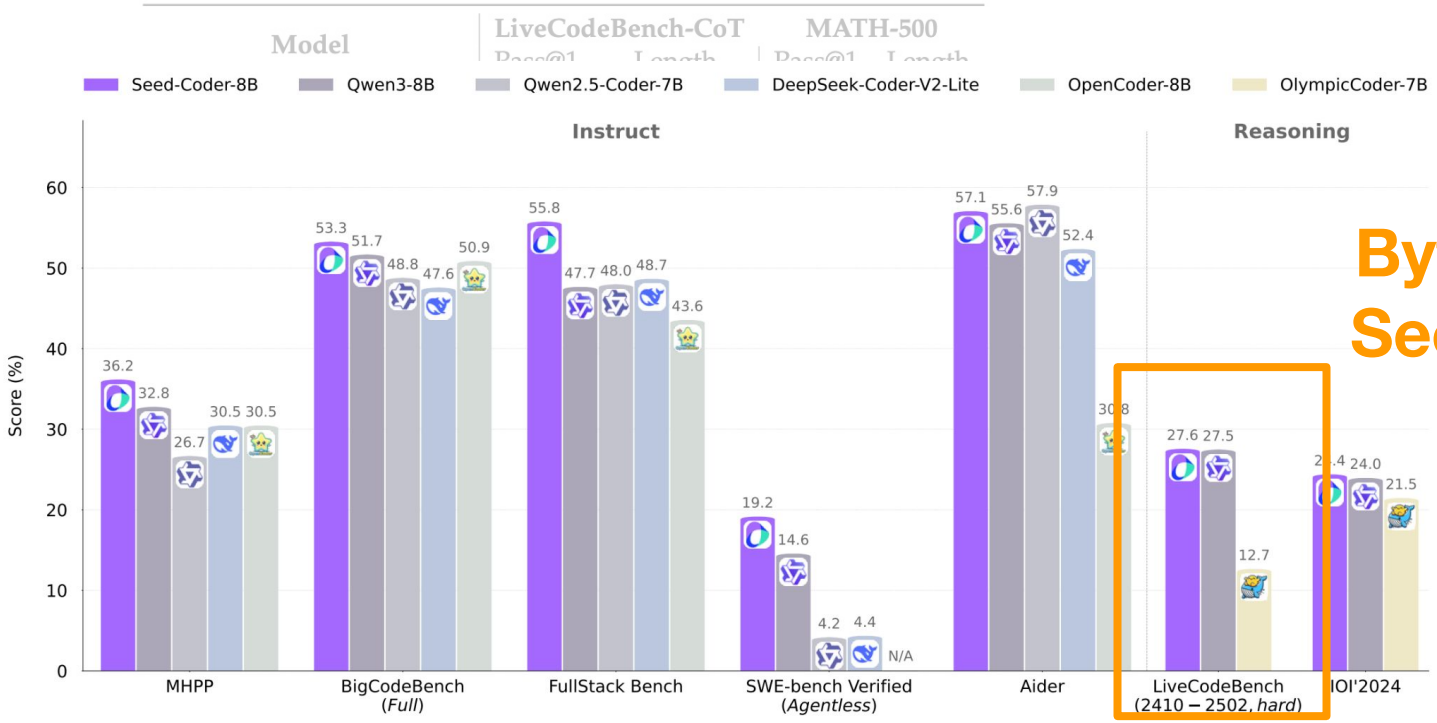
5.4.1. Distillation from DeepSeek-R1

We ablate the contribution of distillation from DeepSeek-R1 based on DeepSeek-V2.5. The baseline is trained on short CoT data, whereas its competitor uses data generated by the expert checkpoints described above.

Table 9 demonstrates the effectiveness of the distillation data, showing significant improvements in both LiveCodeBench and MATH-500 benchmarks. Our experiments reveal an interesting trade-off: the distillation leads to better performance but also substantially increases the average response length. To maintain a balance between model accuracy and computational efficiency, we carefully selected optimal settings for DeepSeek-V3 in distillation.

Figure from DeepSeek-V3 Technical Report

Adoption by Frontier Model Developers

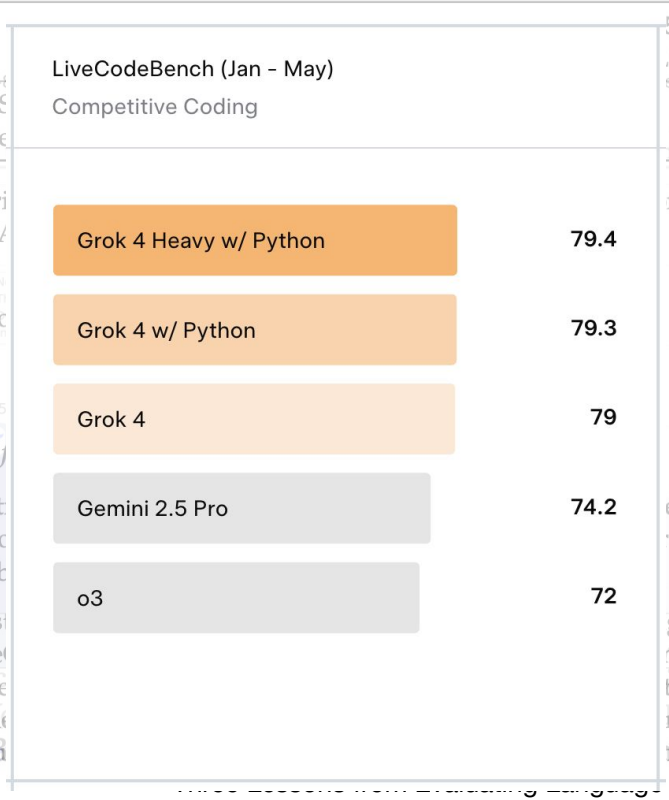


ByteDance SeedCoder

Figure from ByteDance SeedCoder Technical Report

Figure 1. Benchmark performance of instruct and reasoning variants of Seed-Coder-8B.

Adoption by Frontier Model Developers



Grok 4

Figure from https://x.ai/news/grok-4

Adoption by Frontier Model Developers

Model	LiveCodeBench-CoT		MATH-500	
	Pass@1	Length	Pass@1	Length
DeepSeek-V2.5 Baseline	31.1	718	74.6	769

Gemini 2.5: Pushing the Frontier with Advanced Reasoning, Multimodality, Long Context, and Next Generation Agentic Capabilities.

Gemini 2.5

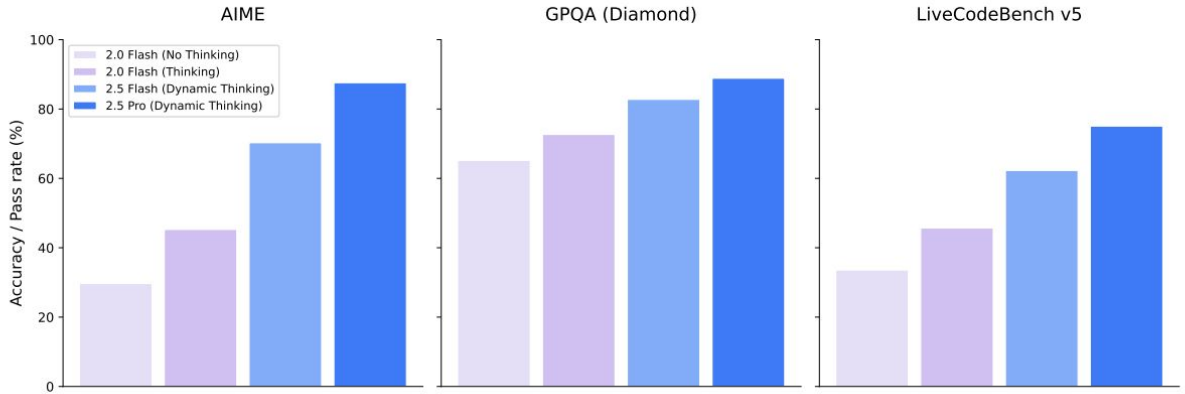


Figure from Gemini 2.5 Technical Report

Figure 3 | Impact of "Thinking" on Gemini's performance on AIME 2025 (Balunović et al., 2025), LiveCodeBench (corresponding to 10/05/2024 - 01/04/2025 in the UI) (Jain et al., 2024) and GPQA diamond (Rein et al., 2024) benchmarks.

LeanDojo: Theorem Proving with Retrieval-Augmented Language Models


Kaiyu Yang


Aidan Swope


Alex Gu


Rahul Chalamala


Peiyang Song


Shixing Yu


Saad Godil

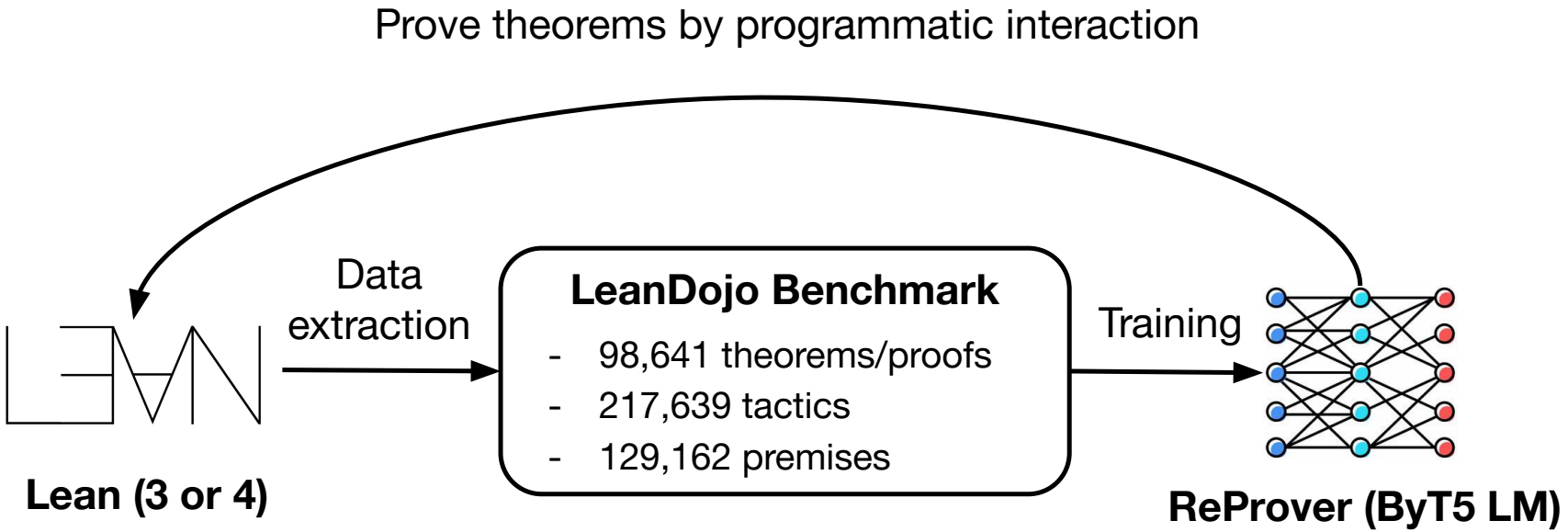

Ryan Prenger


Anima Anandkumar

Democratizing Theorem Proving Research

	Dataset available	Model available	Code available	Interaction tool available	Model size (# params)	Compute (hours)
Jiang et al., LISA, 2021	✓	✗	✗	✓	163M	-
Jiang et al., Thor, 2022	✓	✗	✗	✓	700M	1K on TPU
First et al., Baldur, 2023	✓	✗	✗	✓	62,000M	-
Polu and Sutskever, GPT-f, 2020	✗	✗	✗	✗	774M	40K on GPU
Han et al., PACT, 2022	✓	✗	✗	✓	837M	1.5K on GPU
Polu et al., 2023	✓	✗	✗	✓	774M	48K on GPU
Lample et al., HTPS 2022	✓	✗	✗	✗	600M	34K on GPU
Wang et al., DT-Solver, 2023	✓	✗	✗	✗	774M	1K on GPU
LeanDojo (ours)	✓	✓	✓	✓	**517M**	**120 on GPU**

LeanDojo


Prove theorems by programmatic interaction
Data extraction
LeanDojo Benchmark
- 98,641 theorems/proofs
- 217,639 tactics
- 129,162 premises
Training
Lean (3 or 4)
ReProver (ByT5 LM)

Data Extraction in LeanDojo

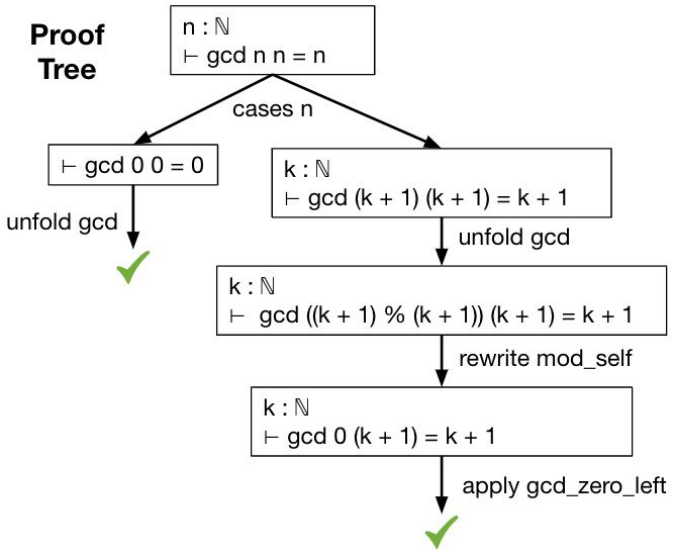
Lean Repo

extract information about Lean repo from GitHub URL

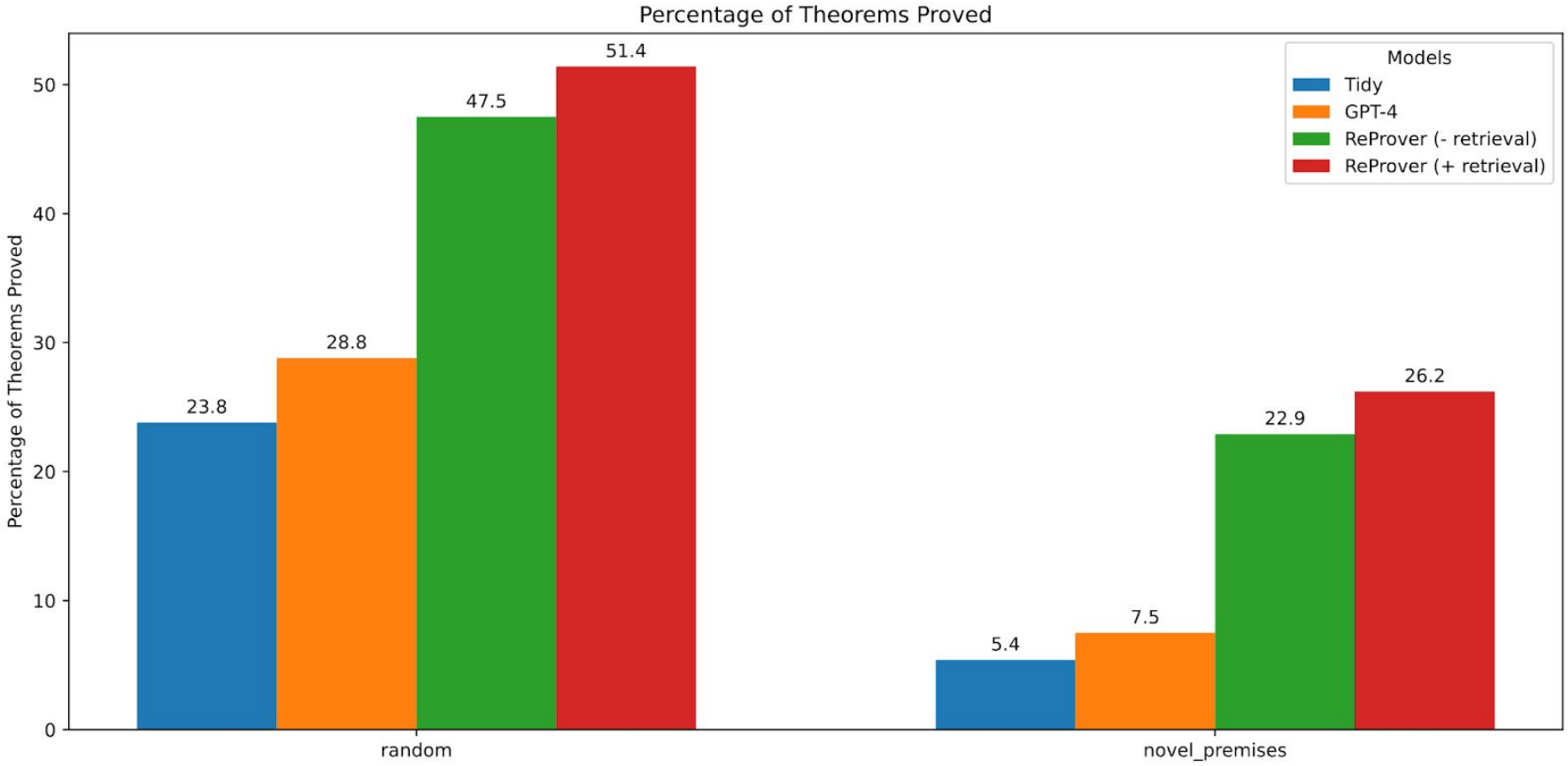
data/nat/gcd.lean

```
def gcd : nat → nat → nat                    -- gcd x y
| 0       y := y                             -- Case 1: x == 0
| (x + 1) y := gcd (y % (x + 1)) (x + 1)     -- Case 2: x > 0

theorem gcd_zero_left (x : nat) : gcd 0 x = x := begin simp [gcd] end

theorem gcd_self (n : nat) : gcd n n = n :=
begin
  cases n,
  { unfold gcd },
  unfold gcd,
  rewrite mod_self,
  apply gcd_zero_left
end
```

Traced Structures



Small, Competitive Model

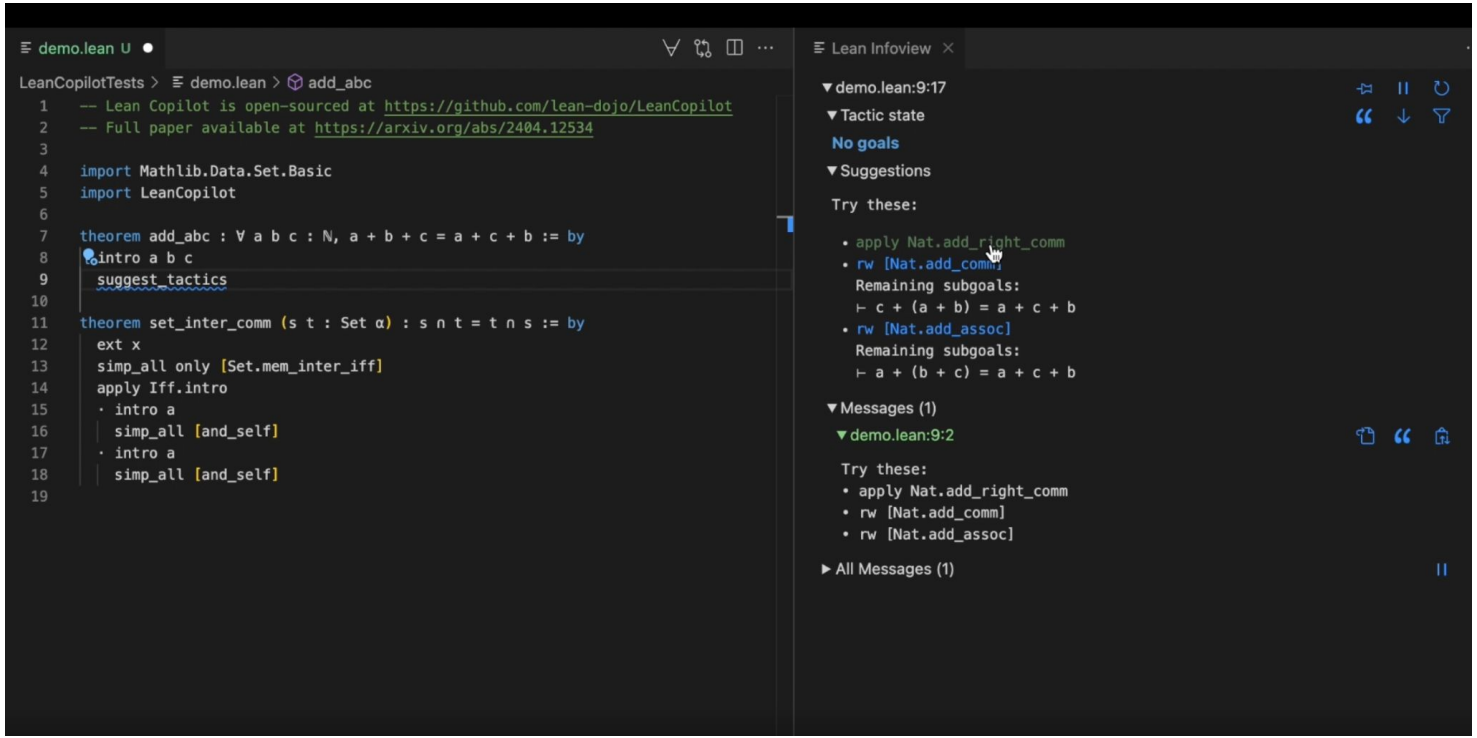


LeanDojo

What We Measure: Lean Theorem Proving

Democratizing theorem proving research through data extraction, benchmark, and modeling

Democratizing Theorem Proving Research



Lean Copilot

Democratizing Theorem Proving Research

LLEMMA: AN OPEN LANGUAGE MODEL FOR MATHEMATICS

Zhangir Azerbayev [1,2] **Hailey Schoelkopf** [2] **Keiran Paster** [3,4]
Marco Dos Santos [5] **Stephen McAleer** [6] **Albert Q. Jiang** [5] **Jia Deng** [1]
Stella Biderman [2] **Sean Welleck** [6,7]

[1] Princeton University [2] EleutherAI [3] University of Toronto [4] Vector Institute
[5] University of Cambridge [6] Carnegie Mellon University [7] University of Washington

Democratizing Theorem Proving Research

Lean-STaR

Table 2: **Pass rates about InternLM2-Plus-7B on the minif2f-test dataset with Lean.** This table shows the pass rates of previous works and our work. The evaluation setting is the same as Table 1.

Approach	Decoding	N	K	S	Pass rate
InternLM2-plus-7b (Ying et al., 2024) (from paper)	Search	1000	1	32	43.4%
InternLM2-plus-7b (Ying et al., 2024) (reproduced)	Search	1000	1	32	42.6%
InternLM2-plus-7b (Ying et al., 2024)	Sampling	50	32	1	40.9%
SFT (InternLM2-plus-7b) (Ying et al., 2024)	Sampling	50	32	1	41.3%
Lean-CoT (InternLM2-plus-7b)	Sampling	50	32	1	43.4%
Lean-STaR (iter-1) (InternLM2-plus-7b)	Sampling	50	32	1	45.4%
InternLM2-plus-7b (Ying et al., 2024)	Sampling	50	64	1	42.2%
SFT (InternLM2-plus-7b) (Ying et al., 2024)	Sampling	50	64	1	43.4%
Lean-CoT (InternLM2-plus-7b)	Sampling	50	64	1	45.5%
Lean-STaR (iter-1) (InternLM2-plus-7b)	Sampling	50	64	1	**46.3%**

🏆 PutnamBench Leaderboard 🏆

Benchmarking formal mathematical reasoning on the Putnam Mathematical Competition.

GITHUB PAPER

With answer | No answer

Lean (out of 672)

#	Model	num-solved	compute
1	Aleph Prover (Logical Intelligence)	668	Avg $68, Limit $1400 per problem
2	Aleph Prover (Logical Intelligence)	637	Avg $54, Limit $400 per problem
3	Goedel-Architect💙	597	pass@4, Avg. $1.47 total cost / problem
4	Seed-Prover 1.5 (ByteDance)	581	10 H20 days per problem
5	Aleph Prover (Logical Intelligence)	500	Avg $23, Limit $100 per problem
6	Hilbert💙	462	avg pass@1840
7	AxProverBase (Axiomatic AI)💙	365	Avg $12.60, Opus 4.5 w/ 50 iterations
8	Seed-Prover (ByteDance)	329	MEDIUM
9	Ax-Prover (Axiomatic AI)💙	91	pass@1, avg. 100 tool calls
10	Goedel-Prover-V2💚	86	pass@184
11	DeepSeek-Prover-V2💚	47	pass@1024
12	GPT-5 (ReAct, 10 turns)	28	pass@1, 10 tool calls
13	DSP+💚	23	pass@128
14	Bourbaki💚	14	pass@512
15	Kimina-Prover-7B-Distill💚	10	pass@192

ks - Alex Gu

```
lemma k_le_3_for_n_le_4_round1_main (n k : ℕ) (verts : Finset ℝ) (lines : Finset (ℝ × ℝ)) (points : Finset (ℕ × ℕ)) (hn : 3 ≤ n) (hcard :
lines.card + verts.card = n) (hallpoints : ∀ p, p ∈ points ↔ p.1 ≥ 1 ∧ p.2 ≥ 1 ∧ p.1 + p.2 ≤ n + 1) (hmain : ∀ p ∈ points, (∃ l ∈ lines,
l.1 * p.1 + l.2 = p.2) ∨ (∃ x ∈ verts, p.1 = x)) (hk : (lines.filter (fun l ↦ l.1 ≠ 0 ∧ l.1 ≠ -1)).card = k) (hn2 : n = 4):
  k ≤ 3 := by
  by_cases h16 : k > 3
  · -- Assume k > 3, we will derive a contradiction
    have h161 : k ≥ 4 := by linarith
    have h162 : (lines.filter (fun l ↦ l.1 ≠ 0 ∧ l.1 ≠ -1)).card ≥ 4 := by linarith
    have h163 : (lines.filter (fun l ↦ l.1 ≠ 0 ∧ l.1 ≠ -1)).card ≤ lines.card := by apply Finset.card_filter_le
    have h164 : lines.card + verts.card = 4 := by linarith
    have h165 : lines.card ≥ 4 := by omega
    have h166 : lines.card = 4 := by omega
    have h167 : verts.card = 0 := by omega
    have h168 : verts = ∅ := by
      exact Finset.card_eq_zero.mp h167
    have h170 : ∀ (p : ℕ × ℕ), p ∈ points → ∃ l ∈ lines, l.1 * (p.1 : ℝ) + l.2 = (p.2 : ℝ) := by
      intro p hp
      have h1701 := hmain p hp
      rcases h1701 with (h1701 | h1702)
      · exact h1701
      · rcases h1702 with ⟨x, hx, _⟩
        have h1703 : x ∈ verts := hx
        rw [h168] at h1703
        contradiction
    have h176 : (lines.filter (fun l ↦ l.1 ≠ 0 ∧ l.1 ≠ -1)).card = 4 := by
      have h1761 : (lines.filter (fun l ↦ l.1 ≠ 0 ∧ l.1 ≠ -1)).card ≥ 4 := h162
      have h1762 : (lines.filter (fun l ↦ l.1 ≠ 0 ∧ l.1 ≠ -1)).card ≤ lines.card := h163
      have h1763 : lines.card = 4 := h166
      linarith
    have h1761 : (lines.filter (fun l ↦ l.1 ≠ 0 ∧ l.1 ≠ -1)).card = lines.card := by linarith
    have h177 : ∀ l ∈ lines, l.1 ≠ 0 ∧ l.1 ≠ -1 := by
      intro l hl
      by_contra h1771
      ...
```

Seed-Prover's Lean Proof of IMO 2025 P1

(total 4000+ lines)

ProofOptimizer

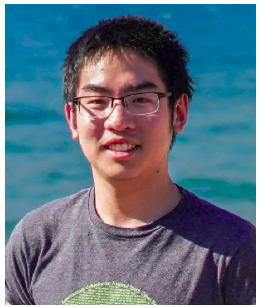
What We Measure: Proof Simplicity

ProofOptimizer: Training Language Models to Simplify Proofs without Human Demonstrations

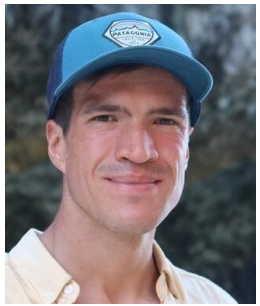
✂️ ProofOptimizer

Training Language Models to Simplify Proofs without Human Demonstrations

Alex Gu



Bartosz Piotrowski



Fabian Gloeckle

Kaiyu Yang

Aram H. Markosyan

Task and Evaluation Metrics

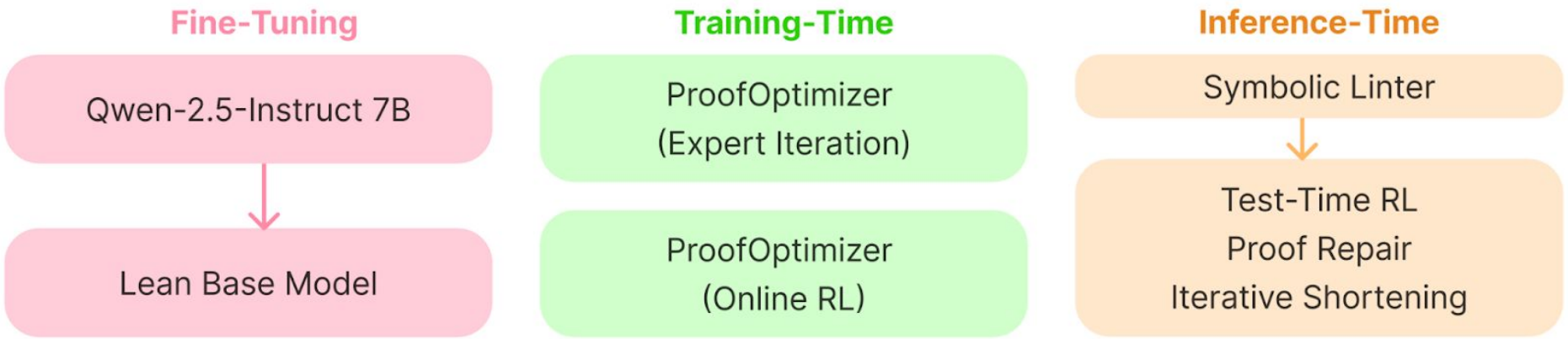
Task: Given a formal statement *s* with proof *p*, minimize a simplicity measure ***L*** (today, we use proof length)

$$p^* = \arg\min \boldsymbol{L}(x), \text{ where } x \text{ proves } s$$

Metrics:

- min@k: minimum shortened proof length
- red@k: maximum relative proof length reduction

ProofOptimizer: Overview



Average Reduction: 87% (miniF2F), 57% (PutnamBench), 49% (Seed-Prover IMO 2025)

Training: Expert Iteration and RL

Dataset: 145K Lean proofs generated by Goedel-Prover-V2

Expert Iteration

1. Sample candidate proofs
2. Filter correct shortenings
3. Fine-tune on (original, shortened) pairs

Reinforcement Learning

Algorithm: GRPO

Reward: *relative shortening*

Results: Expert Iteration vs. RL

Category	Model	Min@1 ↓	Min@32 ↓	Red@1 ↑	Red@32 ↑
Linter	-	*1359*		*0.0%*	
Base	Gemini-2.5-Pro	1348	1303	5.5%	18.0%
	Gemini-2.5-Pro + States	1371	1319	6.1%	19.2%
	Base (7B)	1341	1222	3.9%	20.5%
ExpIt	Base + It 1	1341	1215	5.2%	22.5%
	Base + It 2	1335	1186	6.9%	24.7%
	ProofOptimizer-ExpIt	1328	**1161**	8.2%	**26.3%**
RL	*ProofOptimizer-RL*	**1303**	1258	**14.9%**	21.1%

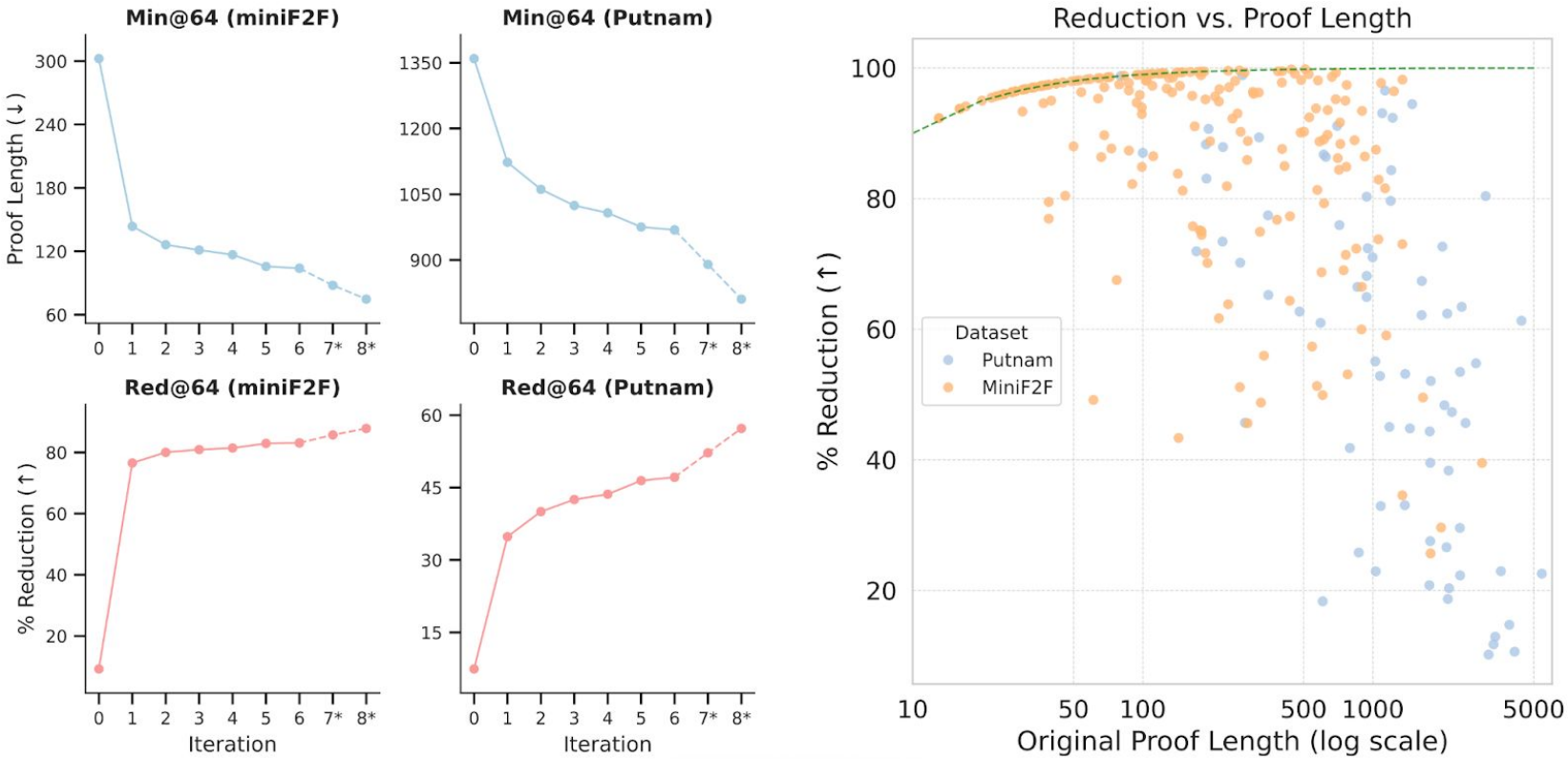
Takeaway: Expert iteration has the best pass@32, while RL has the best pass@1

Inference-Time: Techniques

Symbolic Linter: use Lean compiler messages such as `tactic does nothing` to remove irrelevant lines in the proof

Iterative Shortening: sample *k* candidate shortenings, take shortest, and repeat

Results: Iterative Shortening



```
theorem mathd_algebra_338 -- Original Proof
  (a b c : ℝ)
  (h₀ : 3 * a + b + c = -3)
  (h₁ : a + 3 * b + c = 9)
  (h₂ : a + b + 3 * c = 19) :
  a * b * c = -56 := by
  have h₃ : b = a + 6 := by
    have h₃₁ : -a + b = 6 := by
      have h₃₂ : (a + 3 * b + c) - (3 * a + b
    ↪ + c) = 9 - (-3) := by
        linarith
      linarith
    linarith

  have h₄ : c = a + 11 := by
    have h₄₁ : -a + c = 11 := by
      have h₄₂ : (a + b + 3 * c) - (3 * a + b
    ↪ + c) = 19 - (-3) := by
        linarith
      linarith
    linarith

  have h₅ : a = -4 := by
    have h₅₁ : 3 * a + b + c = -3 := h₀
    rw [h₃, h₄] at h₅₁
    ring_nf at h₅₁ ⊢
    linarith

  have h₆ : b = 2 := by
    rw [h₃]
    rw [h₅]
    <;> norm_num

  have h₇ : c = 7 := by
    rw [h₄]
    rw [h₅]
    <;> norm_num

  have h₈ : a * b * c = -56 := by
    rw [h₅, h₆, h₇]
    <;> norm_num

  exact h₈
```

Listing 10: Original Proof (Length 214)

```
theorem mathd_algebra_338
  (a b c : ℝ)
  (h₀ : 3 * a + b + c = -3)
  (h₁ : a + 3 * b + c = 9)
  (h₂ : a + b + 3 * c = 19) :
  a * b * c = -56 := by
  have : a = -4 := by linarith
  subst_vars
  nlinarith
```

Listing 11: Simplified Proof (Length 11)

```
theorem imo_1960_p2
  (x : ℝ)
  (h₀ : 0 ≤ 1 + 2 * x)
  (h₁ : (1 - Real.sqrt (1 + 2 * x))^2 ≠ 0)
  (h₂ : (4 * x^2) / (1 - Real.sqrt (1 + 2*x))^2 < 2*x + 9)
  (h₃ : x ≠ 0) :
  -(1 / 2) ≤ x ∧ x < 45 / 8 := by
  constructor
  · nlinarith [sq_nonneg (x + 1 / 2)]
  · set s := Real.sqrt (1 + 2 * x) with hs
    have h₅₁ : 0 ≤ 1 + 2 * x := h₀
    have h₅₂ : s ≥ 0 := Real.sqrt_nonneg _
    have h₅₃ : s ^ 2 = 1 + 2 * x := by
      rw [hs]
      rw [Real.sq_sqrt] <;> linarith
    have h₅₄ : (1 - s) ^ 2 ≠ 0 := by simpa [hs] using h₁
    have h₅₅ : s ≠ 1 := by
      intro h
      have h₅₅₁ : (1 - s) ^ 2 = 0 := by
        rw [h]
        norm_num
      contradiction
    have h₅₆ : (s + 1) ^ 2 * (s - 1) ^ 2 = (s ^ 2 - 1) ^ 2 := by
      ring
    have h₅₇ : (s ^ 2 - 1 : ℝ) ^ 2 = 4 * x ^ 2 := by
      rw [h₅₃]
      ring
    have h₅₈ : (4 : ℝ) * x ^ 2 / (s - 1) ^ 2 = (s + 1) ^ 2 := by
      have h₅₈₁ : (s - 1 : ℝ) ^ 2 ≠ 0 := by
        intro h
        have h₅₈₂ : (1 - s : ℝ) ^ 2 = 0 := by
          calc
            (1 - s : ℝ) ^ 2 = (s - 1 : ℝ) ^ 2 := by ring
            _ = 0 := by rw [h]
        contradiction
      field_simp [h₅₈₁] at h₅₇ ⊢
      nlinarith
    have h₅₉ : (4 : ℝ) * x ^ 2 / (1 - s) ^ 2 = (s + 1) ^ 2 := by
      rw [← h₅₈]
      ring
    nlinarith [sq_nonneg (s - 1)]
```

Listing 14: Original Proof (Length 330)

```
theorem imo_1960_p2
  (x : ℝ)
  (h₀ : 0 ≤ 1 + 2 * x)
  (h₁ : (1 - Real.sqrt (1 + 2 * x))^2 ≠ 0)
  (h₂ : (4 * x^2) / (1 - Real.sqrt (1 + 2*x))^2 < 2*x + 9)
  (h₃ : x ≠ 0) :
  -(1 / 2) ≤ x ∧ x < 45 / 8 := by
  constructor
  · nlinarith [sq_nonneg (x + 1 / 2)]
  · have h₅₇ : (4 : ℝ) * x ^ 2 / (1 - Real.sqrt (1 + 2 * x)) ^ 2 = (1 + Real.sqrt (1 + 2 *
    ↪ x)) ^ 2 := by
      have h₅₈ : (1 - Real.sqrt (1 + 2 * x)) ^ 2 ≠ 0 := by assumption
      field_simp [h₅₈]
      nlinarith [sq_sqrt (show 0 ≤ 1 + 2 * x by assumption)]
    nlinarith [sq_sqrt (show 0 ≤ 1 + 2 * x by assumption),
      Real.sqrt_nonneg (1 + 2 * x)]
```

Listing 15: Simplified Proof (Length 125)

1. Focusing on the Right Metrics Without Bias is Crucial

2. Looking Beyond the Score Reveals Hidden Nuances

3. What We Measures Shapes What We Build

1. Focusing on the Right Metrics Without Bias is Crucial

2. Looking Beyond the Score Reveals Hidden Nuances

3. What We Measures Shapes What We Build