ICLR 2026

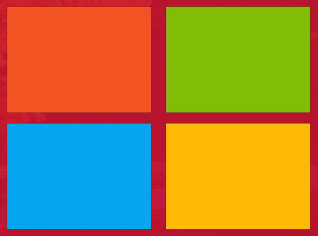
CodeSense: a Real-World Benchmark and Dataset for Code Semantic Reasoning

Monoshi Kumar Roy[1], Simin Chen[2], Benjamin Steenhoek[3], Jinjun Peng[2], Gail Kaiser[2], Baishakhi Ray[2], Wei Le[1]

[1]Iowa State University

[2]Columbia University

[3]Microsoft Data & AI



Acknowledgement: This work was supported in part by NSF CCF-2313054, NSF CCF-2313055, NSF CNS-2247370, DARPA/NIWC Pacific N66001-21-C-4018.

Motivation

```
void foo(int input){
  int n = input * 23;
  if (n <= 3465 && n >= 2287){
     //dangerous code that needs to
be tested
}
```

(A) **Test Input Generation and Program Execution Reasoning**

- Semantic Constraint n in range [2287, 3465]
- Requires solving inequality **2287 ≤ (input ✕ 23) ≤ 3465**
- Model must generate the 'input' (e.g. *input = 100*)

```
void bar (int nbits) {
     FFTContext *s = malloc(sizeof
(* s));
     if (s && nbits==100)
        free(s);
     else ... return s;
 }
```

(B) **Vulnerability Detection, Fault Localization and Program Repair**

- Memory Leak Vulnerability
- The model must track the pointer 's'across control branches
- Requires understanding of "contract" of **malloc()** and **free()** (API semantics)

Why Codesense? (Current Landscape vs. Our Approach)

- **Limitations of Existing Benchmarks**
  - **Synthetic and Narrowly Scoped**
    - HumanEval+ (Liu et al., 2023), LiveCodeBench (Jain et al., 2024)

```
"""Complete the function that takes two integers as inputs
and returns their product as the output.
Assume the inputs are always valid.
Examples:
multiply(8, 2) must return 16
multiply(0, 777) must return 0
multiply(-32,64) must return -2048
"""
```

Why Codesense? (Current Landscape vs. Our Approach)

- **Limitations of Existing Benchmarks**
  - **Synthetic and Narrowly Scoped**
    - HumanEval+ (Liu et al., 2023), LiveCodeBench (Jain et al., 2024)
  - **Task Specific Focus**
    - SWE-Bench (Jimenez et al., 2024), SWE-PolyBench (Rashid et al., 2025)

```
Project: myProject
File: ensemble/_forest.py

342:  def _validate_y_classifier(y):
343:      check_classification_targets(y)
344:      y = np.copy(y)
...
410:  # Issue #2154: 'y' is copied but the
411:  # original reference is still used
412:  # in the sub-estimator calls.
```

The Issue Description:

Title: Memory inefficiency in RandomForest during large-scale fit.
Description: The y array is duplicated unnecessarily on line 344, causing OOM errors on 64GB machines. Needs a reference-pass fix.

Why Codesense? (Current Landscape vs. Our Approach)

- **Limitations of Existing Benchmarks**
  - **Synthetic and Narrowly Scoped**
    - HumanEval+ (Liu et al., 2023), LiveCodeBench (Jain et al., 2024)
  - **Task Specific Focus**
    - SWE-Bench (Jimenez et al., 2024), SWE-PolyBench (Rashid et al., 2025)
  - **Coarse-Grained Focus**
    - CruxEval (Gu et al., 2024), REval (Chen et al., 2024)

```
def f(a, b, s):
    a = a + len(s)
    if a > 10 and "z" in s:
        b = b * 2
    else:
        b = b - a
    if b < 0:
        s = s + "!"
    return [b, s]
assert f(5, 20, "apple") == [10,
'apple']
```

```
def f(a, b, s):
    a = a + len(s)# Executed/not?
    if a > 10 and "z" in s:
        b = b * 2 # Var value+type
    else:
        b = b - a
    if b < 0:
        s = s + "!"
    return [b, s]
assert f(5, 20, "apple") == [10,
'apple']
```

Our Contributions

⚡ The Framework

A scalable, open-source tool for automated execution tracing and ground-truth annotation.

🧩 Multi-Level Tasks

Fine-grained reasoning (Statement, Block, Function) grounded in real-world SE needs.

🌍 Large-Scale Dataset

4,400+ samples from 744 real-world projects in Python, C, and Java.

📊 Empirical Insights

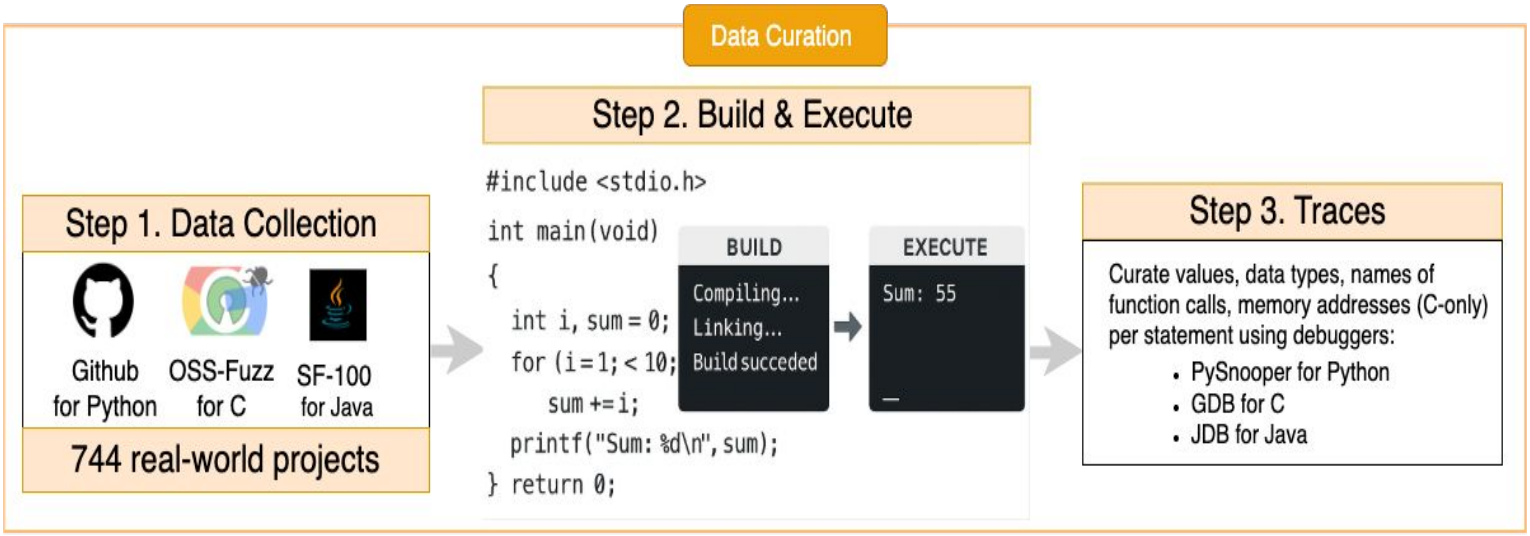
Evaluation of 14 SOTA LLMs and a live leaderboard for reproducibility.

Benchmark Comparison and Novelty

● Full Support | ◐ Partial Support | ○ Not Supported

Benchmark	Real-World Projects	Multilingual	Fine-Grained Reasoning	API Understanding
CruxEval (Gu et al.)	○	○	○	○
CruxEval-X (Xu et al.)	○	●	○	○
REval (Chen et al.)	○	○	◐	○
CodeMind (Liu et al.)	●	●	○	●
CoRe (Xie et al.)	○	●	●	○
CodeSense	●	●	●	●

Benchmark Construction (Data Curation)



🌐**Diverse Real-World Sourcing**

- 744 projects (GitHub, OSS-Fuzz, SF-110).
- **Multi-lingual:** Python, C, and Java.

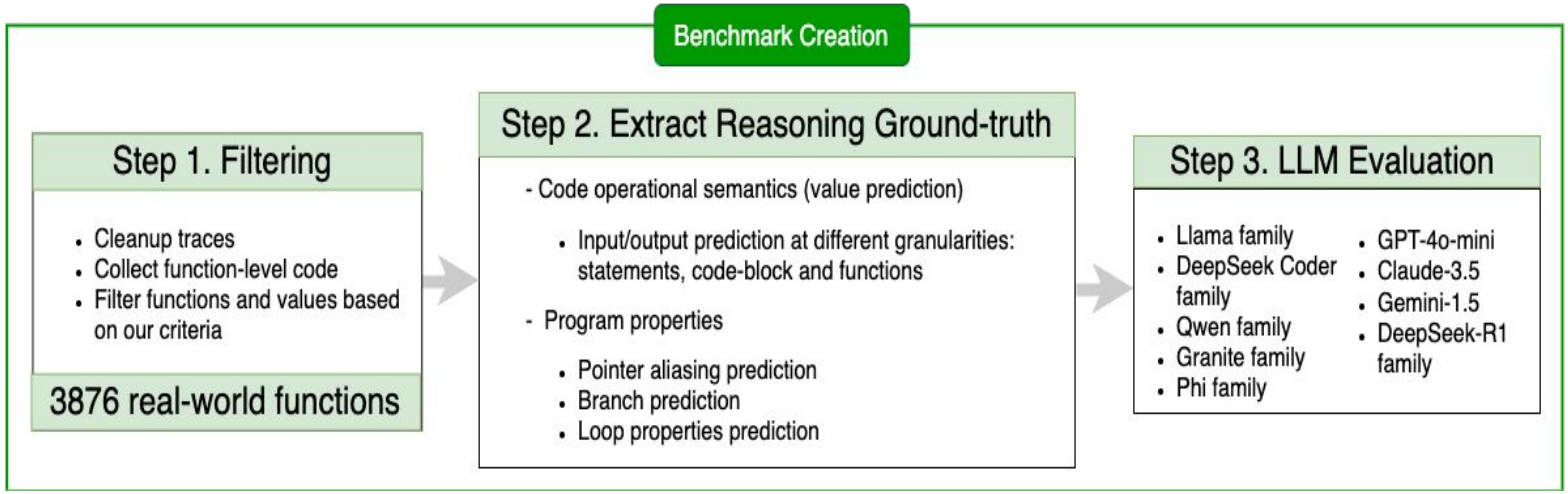
🛠️**Dynamic Tracing Infrastructure**

- Tracing via GDB, JDB, PySnooper

🔍**Granular State Extraction**

- Captures variable values, memory addresses, and full execution paths

Benchmark Construction (Benchmark Creation)



🧹 **Target Data Filtering**
- Filtered and Collected 3876 Real-World Functions.

🧠 **Semantic Ground Truth**
- Captures operational semantics: pointer aliasing, branch outcomes, loop properties.

🤖 **Large-Scale LLM Evaluation**
- Standardized testing across **14 state-of-the-art models**.
- Covers diverse families including **GPT-4, Claude, Llama, and DeepSeek**.

Defining Code Reasoning Tasks & Research Questions

RQ1: **Block-Level Semantics (Task 1)**
- **Goal:** Predict final state for statement chunks

RQ2: **Statement-level Semantics (Task 2)**
- **Goal:** Output Prediction for atomic operations
- Arithmetic, Boolean, API, Assignment

RQ3: **Program Properties (Task 3)**
- **Loops**: Iteration counts and value propagation
- **Pointers:** Alias analysis and memory location tracking
- **Branches:** Conditional outcome prediction

```
def calc(x, y): # x = 5, y = 15
    val = x - y  # Block 1
    val = abs(val) * 2 # Block 2
    val = val + 10  # Block 3
    ...
    return val  # Whole function
(output)
```

Defining Code Reasoning Tasks & Research Questions

RQ1: **Block-Level Semantics (Task 1)**

- **Goal:** Predict final state for statement chunks

RQ2: **Statement-level Semantics (Task 2)**

- **Goal:** Output Prediction for atomic operations
- Arithmetic, Boolean, API, Assignment

RQ3: **Program Properties (Task 3)**

- **Loops**: Iteration counts and value propagation
- **Pointers:** Alias analysis and memory location tracking
- **Branches:** Conditional outcome prediction

```
def process_data(val, factor):

    offset = val * 2 + factor #
val = 2, factor = 10
    is_high = offset > 20
#offset= 14, is_high True/False?


    final_score = round(offset,
-1) # offset = 14, final_score?

    return final_score
```

Defining Code Reasoning Tasks & Research Questions

RQ1: **Block-Level Semantics (Task 1)**

- **Goal:** Predict final state for statement chunks

RQ2: **Statement-level Semantics (Task 2)**

- **Goal:** Output Prediction for atomic operations
- Arithmetic, Boolean, API, Assignment

RQ3: **Program Properties (Task 3)**

- **Loops**: Iteration counts and value propagation
- **Pointers:** Alias analysis and memory location tracking
- **Branches:** Conditional outcome prediction

```python
def data(lst): #lst = [1,2,3]
    val = 0
    for i in range(len(lst)):
# no. of iterations?
        val += lst[i] + 5 # val
after second iteration?

    if val > 10: # Will the
branch be taken?
        return val
    return 0
```

```c
void check_alias(int *p, int *q)
{
    int x = 100;
    p = &x;
    q = p;
    x = 200;
    // RQ3 Question: Are 'p' and
'q' aliases at this point?
    // Result: YES (True)
}
```

Defining Code Reasoning Tasks & Research Questions

RQ5: **Does Prompting Technique Help?**

- **Goal:** 0-shot, In-Context Learning, CoT

RQ5: **Approximation Semantics**

- **Goal:** How models perform on abstract values?

RQ6: Comparing Programming Language

- I/O Predictions
- Python, C, Java

```python
def process_data(val, factor):
    offset = val * 2 + factor
# Guess the range of offset
    is_high = offset > 20

    final_score = round(offset, -1)

    return final_score
```

Experimental Setup: Dataset and Models

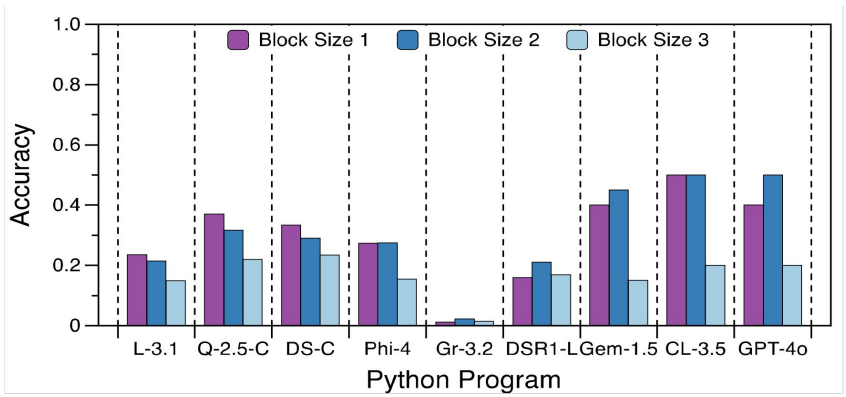
Task	Python	C	Java	Total
Block	1860	731	-	2591
Function	308	94	74	476
Statement	545	485	-	1030
Loop	105	-	-	105
Pointer	-	49	-	49
Branch	232	-	-	232
Total Samples	3050	1359	74	4483

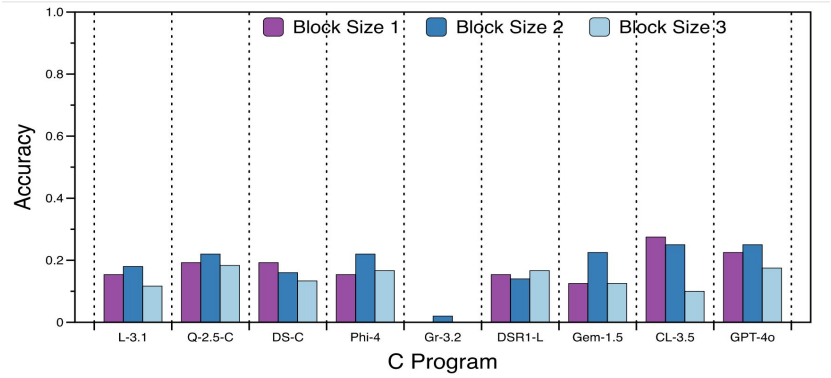
Models Evaluated

We evaluated a diverse range of proprietary and open-source models:

- **Proprietary:** GPT-4o-mini, Claude 3.5 Sonnet, Gemini 1.5 Flash
- **Open-Source:** Llama-3.1, Qwen3, Qwen-2.5-Coder (7B, 32B), Phi-3.5/4, Granite-3.2
- **Specialized:** DeepSeek-R1-Distill-Qwen, DeepSeek-R1-Distill-Llama

RQ1 Results- Block Level Code Semantics



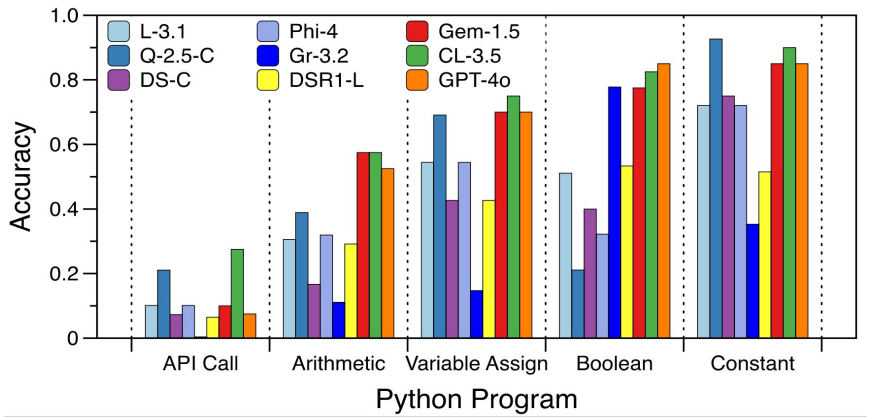


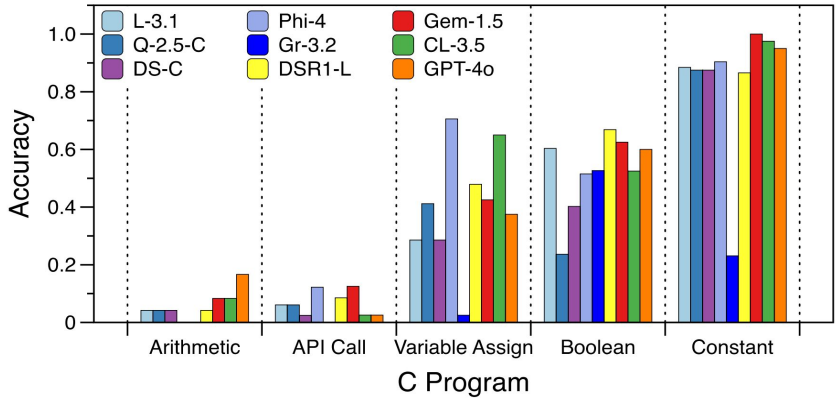
Output Prediction for different Block Sizes

💡 Key Insights

- ➢ Even for 1-statement, Low accuracy (<30% in C and 50% in Python)
- ➢ Significant lower performance in open-source models
- ➢ Challenges
  - Individual statement reasoning
  - Variable tracking

RQ2 Results- Statement Level Semantics



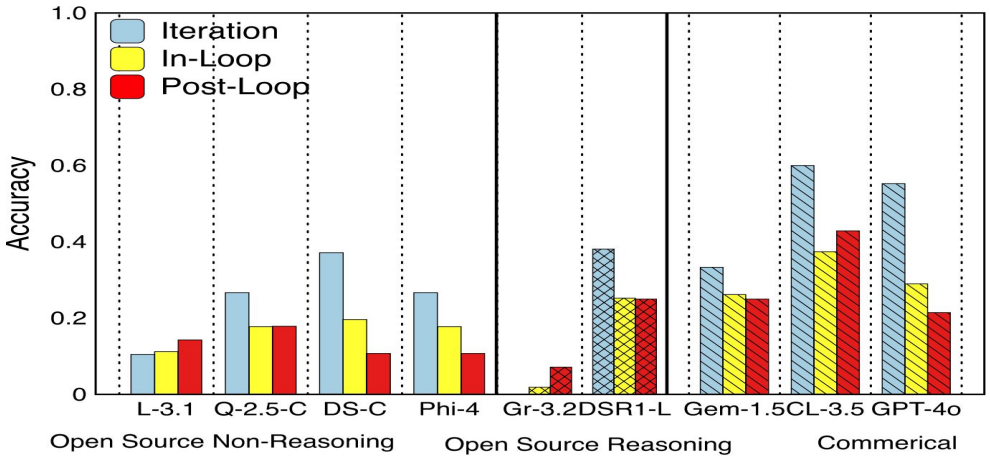


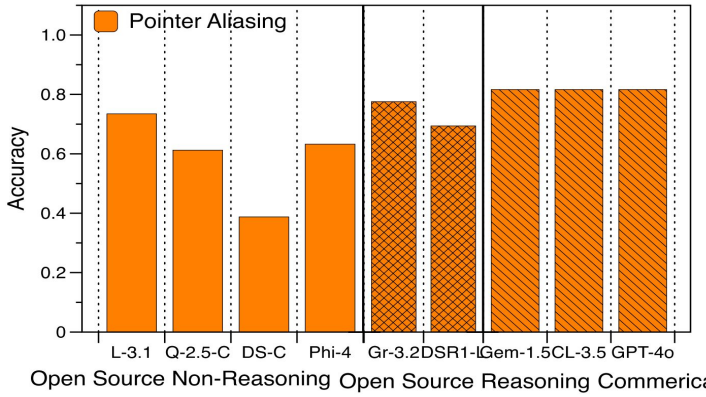
What Types of Statement Models understand well?

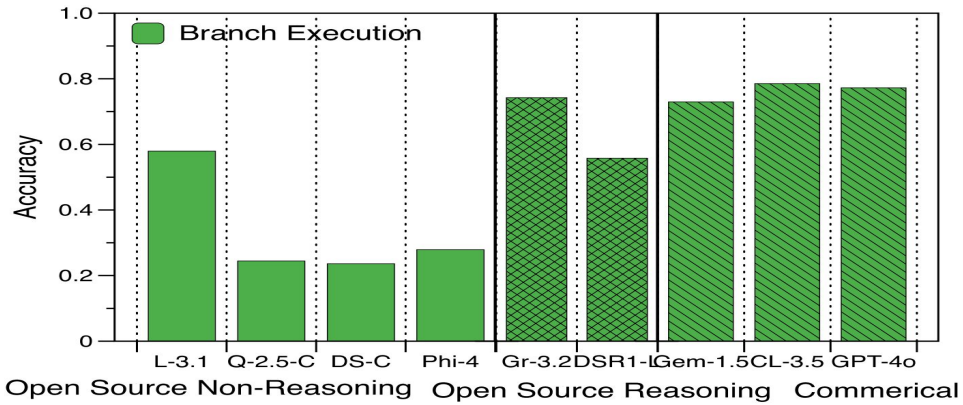
Key Insights

- Logic Dependent Difficulty
  - **Hardest**: API calls
  - **Easiest**: Boolean, Constant
- Good at simple assignments (e.g., a = 3)
- No significant advantage over 'reasoning' models

RQ3 Results- Code Properties





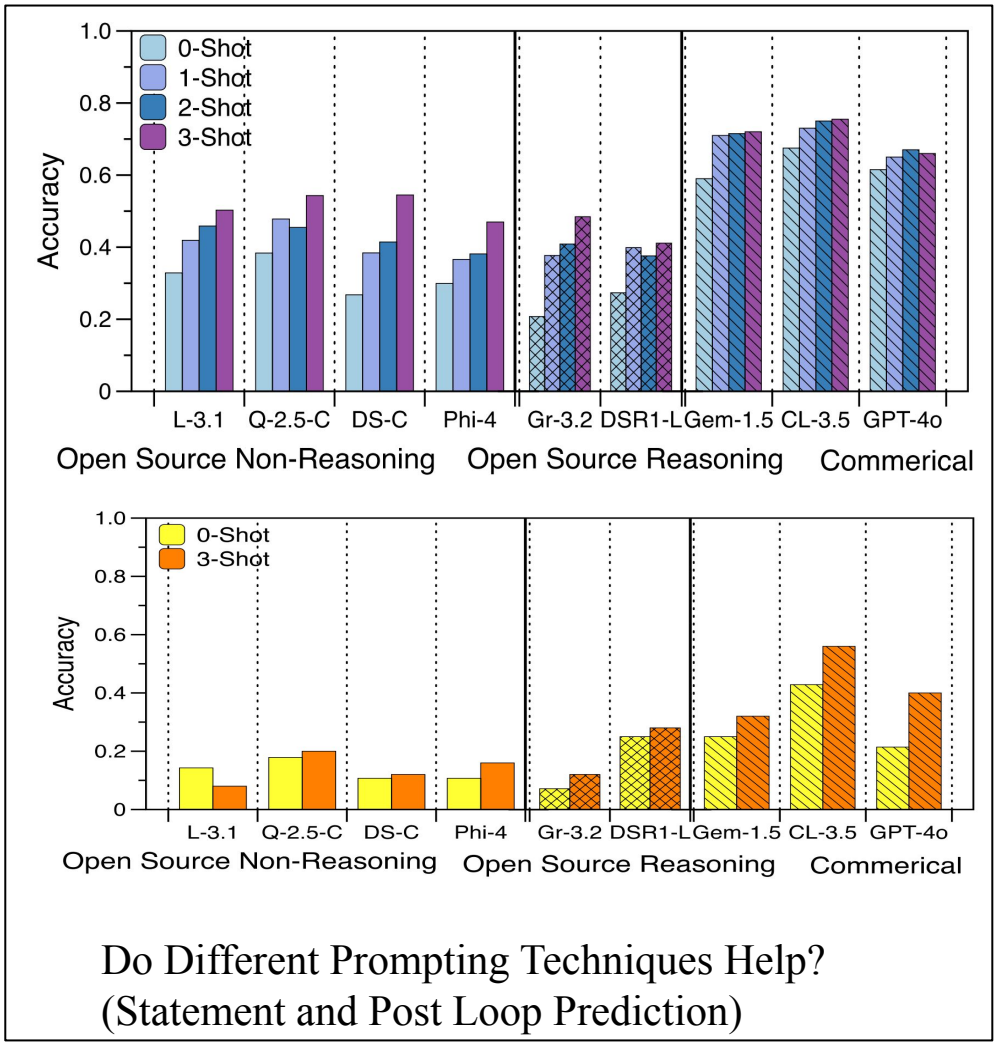


Performance on different program properties

Key Insights

- Models struggle with loop's state tracking
- High performance on Branch and pointer (Binary Prediction)

RQ4 Results- Different Prompting Techniques

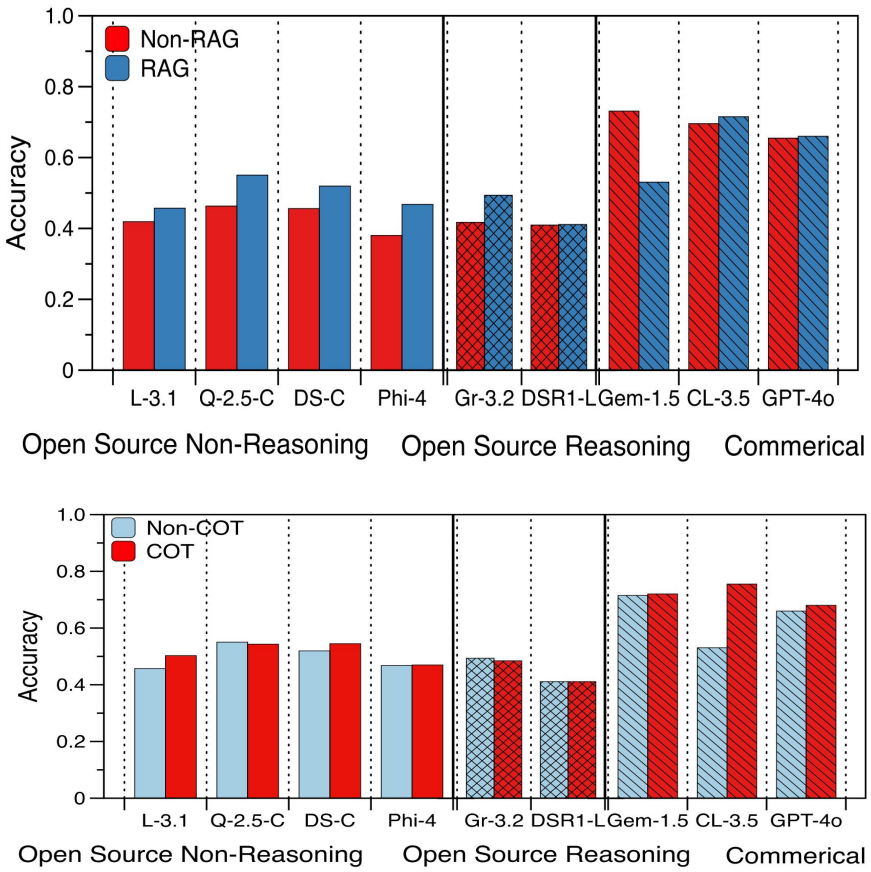


Do Different Prompting Techniques Help?
(Statement and Post Loop Prediction)

Key Insights

- In-Context shot examples help
- Significant performance gap between 0 shot and 3 shots

RQ4 Results- Different Prompting Techniques

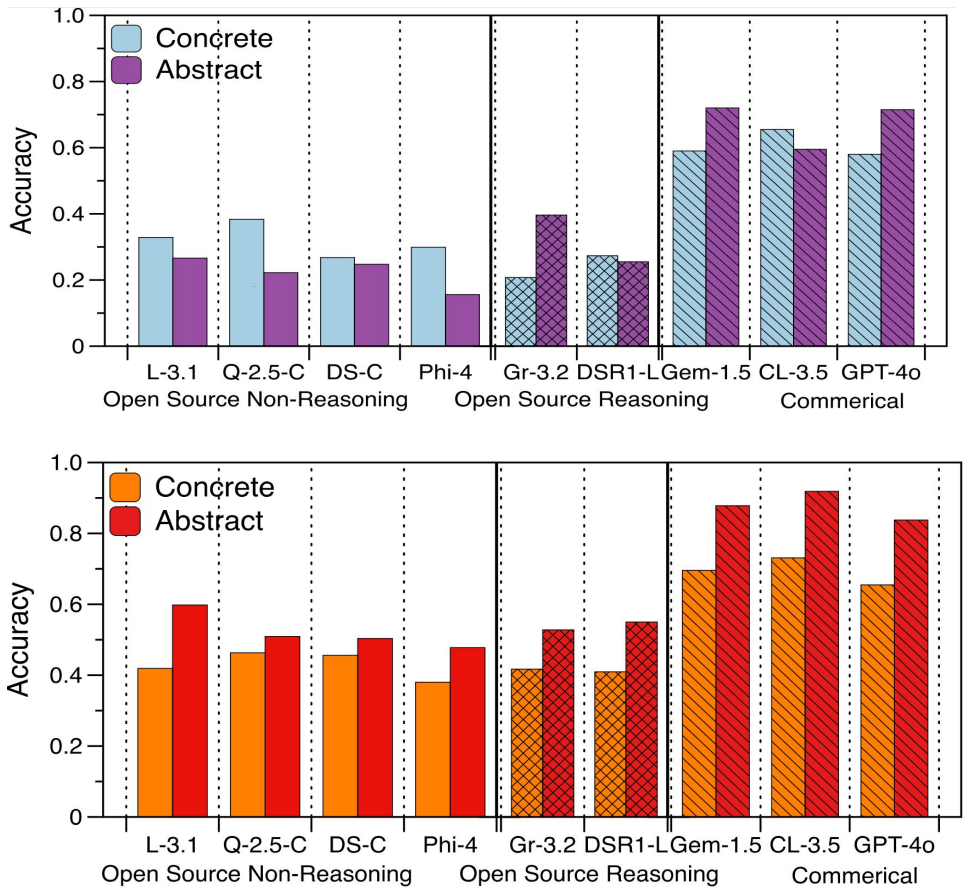


Do Different Prompting Techniques Help? (Statement Prediction)

Key Insights

- Relevant in-context examples (RAG-style) helps
- CoT (Think-step-by-step) provides minimal to no benefit for statement prediction

RQ5 Results- Approximation of Code Semantics

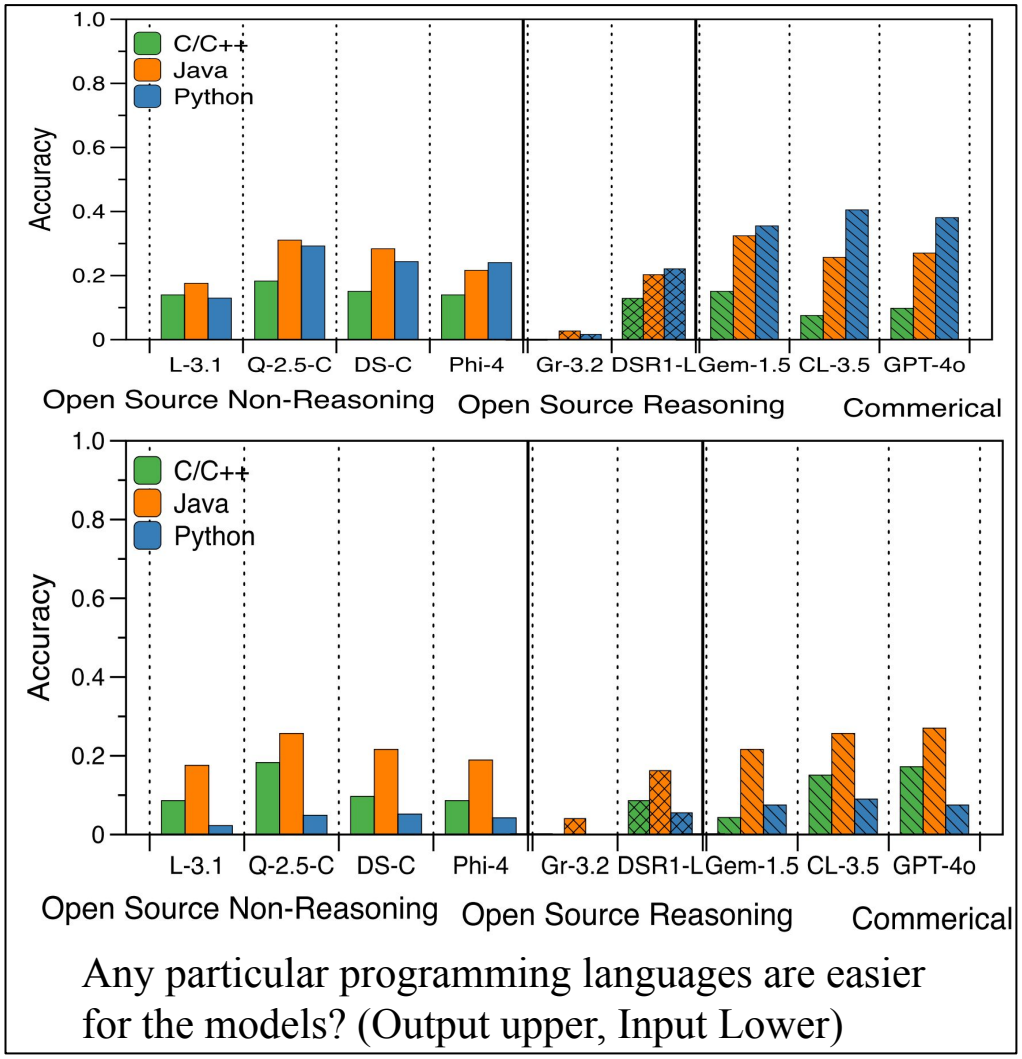


Can model understand "Approximation" of Code Semantics? (Statement Prediction)

💡 Key Insights

- Models are better at predicting abstract ranges
- Models can not apply abstract mapping without in-context example

RQ6 Results- Different Programming Languages



Any particular programming languages are easier for the models? (Output upper, Input Lower)

💡 Key Insights

- Python and Java consistently outperform C in output prediction.
- Python is hard for input prediction
  - Harder to "trace back" high-level abstractions

Comparison With Existing Benchmark

Model	CruxEval	CodeSense	Drop
DeepseekR1-Distill-Qwen-14B	75%	37%	38%
Qwen-2.5-14B	52%	27%	25%
Qwen2.5-Coder-7B	50%	30%	20%

Discussion & Implications

⚠️ **The Semantic Gap:** Current LLMs fail when state becomes complex

🏗️ **Training Paradigm Shift:** Pre-training on raw code text is not sufficient

🛠️ **Tooling for SE**: Benchmarks are critical before deploying LLMs in critical SE tasks

Limitations

🌍 **Language Scope:** Memory-safe languages like Rust or web-focused languages like JavaScript are future work

🔬 **Trace Depth:** Execution traces are limited by test suite coverage in real-world projects

⚡ **Computational Cost**: High-granularity probing of 14+ SOTA models is resource-intensive

Future Work & Appreciation

🚀 **Fine-Tuning on Real-world Traces**: Using the real-world trace data to fine-tune models with execution awareness. (Ding et al., 2023a; 2024)

📈 **Benchmark Expansion:** Incorporating more diverse program properties and multi-agent reasoning tasks.

🙏 **Acknowledgements:** My advisor (Dr. Wei Le), All my co-authors, collaborators and lab mates.

References

Alex Gu, Baptiste Rozière, Hugh Leather, Armando Solar-Lezama, Gabriel Synnaeve, and Sida I. Wang. Cruxeval: A benchmark for code reasoning, understanding and execution, 2024. URL:https://arxiv.org/abs/2401.03065.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code, 2024.
URL:https://arxiv.org/abs/2403.07974.

Junkai Chen, Zhiyuan Pan, Xing Hu, Zhenhao Li, Ge Li, and Xin Xia. Reasoning runtime behavior of a program with llm: How far are we?, 2024.
URL: https://arxiv.org/abs/2403.16437.

Mark Chen et al. Evaluating large language models trained on code, 2021.
URL: https://arxiv.org/abs/2107.03374.

References

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues?, 2024. URL: https://arxiv.org/abs/2310.06770.

Alex Mathai, Chenxi Huang, Petros Maniatis, Aleksandr Nogikh, Franjo Ivančić, Junfeng Yang, and Baishakhi Ray. "KGym: A Platform and Dataset to Benchmark Large Language Models on Linux Kernel Crash Resolution." Advances in Neural Information Processing Systems, vol. 37, 2024, pp. 78053–78078. URL: https://proceedings.neurips.cc/paper_files/paper/2024/hash/8e9ed2a28af7d9085180e3817b2c9a57-Abstract-Datasets_and_Benchmarks_Track.html

Ruiyang Xu, Jialun Cao, Yaojie Lu, Ming Wen, Hongyu Lin, Xianpei Han, Ben He, Shing-Chi Cheung, and Le Sun. Cruxeval-x: A benchmark for multilingual code reasoning, understanding and execution, 2025. URL: https://arxiv.org/abs/2408.13001.

Yangruibo Ding, Ben Steenhoek, Kexin Pei, Gail Kaiser, Wei Le, and Baishakhi Ray. Traced:Execution-aware pre-training for source code, 2023a. URL: https://arxiv.org/abs/2306.07487.

References

Changshu Liu, Shizhuo Dylan Zhang, Ali Reza Ibrahimzada, and Reyhaneh Jabbarvand. Codemind:A framework to challenge large language models for code reasoning, 2024. URL: https://arxiv.org/abs/2402.09664.

Danning Xie, Mingwei Zheng, Xuwei Liu, Jiannan Wang, Chengpeng Wang, Lin Tan, and Xiangyu Zhang. Core: Benchmarking llms code reasoning capabilities through static analysis tasks, 2025. URL: https://arxiv.org/abs/2507.05269

Yangruibo Ding, Jinjun Peng, Marcus J. Min, Gail Kaiser, Junfeng Yang, and Baishakhi Ray. Semcoder: Training code language models with comprehensive semantics reasoning, 2024. URL: https://arxiv.org/abs/2406.01006.