Spotlight

🥑 CoRe: Benchmarking LLMs' Code Reasoning Capabilities through Static Analysis Tasks

Danning Xie, Mingwei Zheng, Xuwei Liu, Jiannan Wang, Chengpeng Wang, Lin Tan, Xiangyu Zhang

PURDUE UNIVERSITY

LLMs are Widely Adopted in Coding Tasks

- **LLMs are widely used in coding tasks**: Code generation, taint analysis, fuzzing, etc.
- LLMs are prompted with high-level objectives (e.g., identify buggy lines) or used directly as static analyzers.
- They require a deep understanding of the program semantics

Example: Applying LLMs to Fuzzing

```
 1  int type, len, i = 0;
 2  while (i < n) {
 3    type = a[i];
 4    if (type == 0) {
 5      i++; continue;
 6    }
 7    if (i + 1 >= n) return;
 8    len = a[i + 1];
 9    if (i + len > n) return;
10    if (type == 1)
11      // vulnerable sink
12      memcpy(out, a + i + 2, len);
13    i += len + 2;
14  }
```

Example: Applying LLMs to Fuzzing

```
 1  int type, len, i = 0;
 2  while (i < n) {
 3    type = a[i];
 4    if (type == 0) {
 5      i++; continue;
 6    }
 7    if (i + 1 >= n) return;
 8    len = a[i + 1];
 9    if (i + len > n) return;
10    if (type == 1)
11      // vulnerable sink
12      memcpy(out, a + i + 2, len);
13    i += len + 2;
14  }
```

To trigger line 12, the input needs to satisfy conditions at line 2, 4, 7, 9, 10

Model's Deep Understanding of the Program Semantics is not Well Tested

It requires model to go beyond surface-level pattern matching and understand:

- **how values propagate through statements;**
- **how control structures govern program execution;**
- **how different parts of the program influence one another.**

Such abilities are **under-evaluated**

- Existing benchmarks evaluate LLMs upon code-centric tasks in an end-to-end fashion (e.g., whether the bug is fixed).
- They do not offer direct fine-grained assessment of LLMs' core program analysis skills.

🥑 CoRe: Benchmarking LLMs' Code Reasoning Capabilities through Foundamental Static Analysis Tasks

- High-quality, human-verified
- **Multi-lingual**: C/C++, Java, Python
- Evaluate LLMs on **fundamental static analysis tasks**: *data dependency*, *control dependency*, *information flow*.
- Leverage **Semantics-Aware Diverse Sampling** strategy
- Consists of **12,553** diverse task instances from **180** programs

Background & Motivation

Three core reasoning tasks:

- Data Dependency
- Control Dependency
- Information Flow

Background & Motivation

Three core reasoning tasks:

- **Data Dependency**
- Control Dependency
- Information Flow

A data dependency occurs when the value of one variable depends on the value of another, typically arising when variables are assigned and then subsequently used [1].

[1] V Aho Alfred, S Lam Monica, and D Ullman Jeffrey. Compilers principles, techniques & tools. pearson Education, 2007.

Background & Motivation

Three core reasoning tasks:

- Data Dependency
- **Control Dependency**
- Information Flow

Control dependency captures whether the execution of one statement is governed by another [2].

[2] Andy Podgurski and Lori A. Clarke. A formal model of program dependences and its implications for software testing, debugging, and maintenance. IEEE Transactions on software Engineering, 16(9):965–979, 1990.

Background & Motivation

Three core reasoning tasks:

- Data Dependency
- Control Dependency
- **Information Flow**

Information flow [3,4] captures how the value of one variable can influence another through data or control dependencies. It may be **explicit**, as in direct assignments, or **implicit**, when control conditions determine which value a variable receives.

[3] Dorothy E Denning and Peter J Denning. Certification of programs for secure information flow. Communications of the ACM, 20(7):504–513, 1977.
[4] Samir Genaim and Fausto Spoto. Information flow analysis for java bytecode. In International Workshop on Verification, Model Checking, and Abstract Interpretation, pages 346–362. Springer, 2005.

Task Formulation

- **Pairwise Query**: given two program elements (variables or lines), determine whether a specific dependency exists. If so, provide a valid trace.

> <Detailed definition, instruction, and examples>
> Below is your target snippet.
>
> `<target code with line number>`
>
> **Question**: Does `(src_var, src_line)` have data dependence over `(dst_var, dst_line)` in function `target_function_name`? If so, provide a trace.
> **Output:**

- **Target-centric Query** (Dependency Source Enumeration): given a single program element, list all other elements in the same function that have the specified dependency relation over it.

> **Question**: Which lines have control dependence over `dst_line` in function `target_function_name`? List all such lines.

Experimental Setup - Models

Model	Size	Reasoning?
Claude 3.7	-	✓
Claude 3.5	-	✗
DeepSeek R1	671B	✓
DeepSeek V3	671B	✗
Gemini 2.5 Pro	-	✓
GPT o3	-	✓
GPT o4-mini	-	✓
GPT 4o	-	✗
Llama 3.1	405B	✗
Qwen 3	235B	✓

10 models in total: 6 with reasoning capabilities

Key Takeaways of Qualitative Study

- LLMs are good at identifying dependencies, but still struggle with tasks that require deeper semantic understanding and multi-step reasoning.
- Reasoning models consistently outperform non-reasoning ones
- Performance drops significantly in the presence of complex control structures, longer function bodies, and backward or non-sequential dependency patterns.

Spotlight

🥑 CoRe: Benchmarking LLMs' Code Reasoning Capabilities through Static Analysis Tasks

Danning Xie, Mingwei Zheng, Xuwei Liu, Jiannan Wang, Chenpeng Zhang, Lin Tan, Xiangyu Zhang

- A **high-quality** benchmark for evaluating LLMs on **fundamental static analysis tasks**: including *data dependency*, *control dependency*, and *information flow*
- CoRe is **multi-lingual**: C/C++, Java, Python
- It includes **12,553 human-verified**, **diverse** instances
- Evaluated on **10 state-of-the-art LLMs** & Qualitative analysis

https://corebench.github.io/