NEURAL INFORMATION PROCESSING SYSTEMS

CoRE: Benchmarking LLMs' Code Reasoning Capabilities through Static Analysis Tasks

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

Datasets & Benchmarks Track

✨ Spotlight ✨

PURDUE UNIVERSITY

Motivation

- LLMs are widely used for **coding tasks**
- LLMs are **prompted with high-level objectives** (e.g., identify buggy lines) or **used directly as static analyzers**

- It requires a deep understanding of the program semantics:
  - **how values propagate through statements,**
  - **how control structures govern program execution,**
  - **how different parts of the program influence one another.**

- Existing benchmarks: in **end-to-end fashion** (e.g., whether the bug is fixed)
- No **direct fine-grained assessment of LLM's core program analysis skills**

```
1  // untrusted input
2  String header =
       getHeader("Content-Type");
3  // tainted expression
4  String expr = "%{"+header+"}";
5  // dangerous sink
6  evaluateOGNL(expr);
```
(a) Taint Analysis

```
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 }
```
(b) Fuzzing

```
1  if (codec != null)
2    in = new LineReader
3      (codec.InputStream(file), job);
4  else {
5    if (start != 0) {
6      skipFirstLine = true;
7      --start;
8      f.seek(start);
9    }
10   in = new LineReader(f, job);
11 }
12 if (skipFirstLine)
13   start += in.read();  // NPE
```
(c) Fault Localization

CoRE

- **High-quality,** human-verified
- **Multi-lingual**: C/C++, Java, Python
- Evaluates LLMs on **fundamental static analysis tasks**
- Leverages **semantics-aware diverse sampling** strategy
  - Ensures *both complexity and diversity*
- Consists of **12,553 diverse task** instances from 180 programs
- Balanced distribution of **LoC**: 21–40, 41–60, 61–80, 81–100

	CoRe			CoRe Lite		
Task Type	Positive	Negative	Total	Positive	Negative	Total
Data Dependency	1,814	2,800	4,614	209	381	590
Control Dependency	1,693	1,959	3,652	239	250	489
Information Flow	2,291	1,996	4,287	291	214	505
Total	5,798	6,755	**12,553**	739	845	**1,584**

Information flow

Captures how the value of one variable can influence another *through data or control dependencies*.

- **Explicit**: direct assignment
- **Implicit**: control conditions determine which value a variable receives

```
1  x = 0
2  if x == 0:
3    y = 1
4  else:
5    y = 0
6  z = y + 1
```
x → y → z

Data Dependency

The value of one variable depends on the value of another

Control Dependency

Whether the execution of one statement is governed by another

Pairwise Query

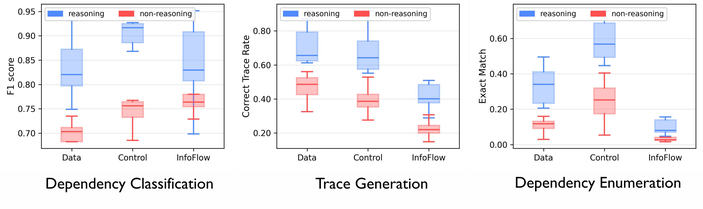
- Given two variables/lines,
  - **Classification**: ask whether a specific dependency exists
  - **Trace Generation**: if so, provide a trace
- Metrics: Precision/Recall/F1, Correct Trace Rate (CT)

Target-centric Query

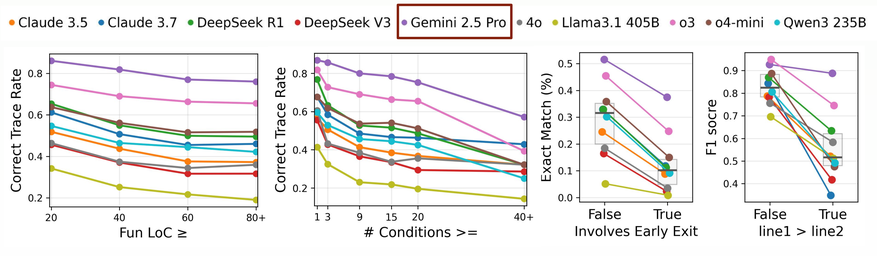
- Given one variable/line,
  - **Dependency Enumeration**: list all other variables/lines that have the specified dependency relation over it
- Metrics: Exact Match (EM), Precision/Recall/F1

Key Results

- Models perform well on dependency classification, with reasoning models achieving **≥80% F1.**
- Dependency source enumeration is the **most difficult**, with most reasoning models **<40%.**



- Model performance declines as the **program complexity increases** and involves **multi-step reasoning.**



Prompt Example (Simplified)

[INSTRUCTIONS]

You are a program-analysis assistant. Perform a **static** data-dependence analysis on a given code snippet, treating each branch or loop condition as potentially taking any outcome, without using semantic or symbolic execution to prune paths.

1. Data Dependence Definition
Data dependence captures the influence of data flow between variables.
We denote each variable instance as `(var,lineNumber)`, meaning the variable `var` defined or updated at `lineNumber`.
Direct Data Dependence: A variable instance `(varB,lineB)` is **directly** data-dependent on `(varA,lineA)` if …
A variable instance `(varB,lineB)` is data-dependent on `(varA,lineA)` if there is a **transitive** (indirect) chain of **direct** data dependencies from `(varA,lineA)` to `(varB,lineB)`.

2. Output Format
<json format explanation>
<In Pairwise Query only> You only need to provide **one** valid chain of data dependence if you conclude a data dependence exists, even if multiple possible chains exist.

3. Intraprocedural Data Dependence
All dependence analysis is performed within **a single function** …

4. Example Code Snippet
<6 examples with code, question, output in specified format, and explanation>

[YOUR TURN]

```Python
import math,string,itertools...
...
def main():
  ...
  for t in aa[1:]:
    if a < x:
      if nt <= t:
        x1 -= lc * s
        xr += (rc+xc) * s
        lc -= nc
        rc += xc
        heapq.heappush(rq,x)
      else:
        nx = rq[0]
        nc = c[nx]
        s = nx - x
        lc += xc
        ...
  return '\n'.join(r)
```

https://corebench.github.io/

Pairwise Query

Question: Does `(rq,12)` have data dependence over `(lc,10)` in function `main`? If so, provide a trace.
Output:

Ground Truth

```
{
"DataDependence": true,
  "Trace": [
      { "from": ["rq", 12], "to": ["nx", 14] },
      { "from": ["nx", 14], "to": ["nc", 15] },
      { "from": ["nc", 15], "to": ["lc", 10] }
  ]
}
```

Target-centric Query

Question: Which variable instances have data dependence over `(lc,10)` in function `main`? List all such variables.
Output:

Ground Truth

```
{
  "DataDependenceSources": [
      ["lc", 17],
      ["xc", …],
      ["nc", 15],
      ["nx", 14],
      ["c", …], …
  ]
}
```