Benchmarking of Code Generative LLMs

Mirza Masfiqur Rahman[1,2], Ashish Kundu[1], Elisa Bertino[2]
[1]Cisco Research [2]Purdue University

SCAN ME

Abstract

- Generative LLMs have proven to be valuable code generators, thus enabling code copilots and meeting several requirements in software engineering.
- But few questions remain-
  - How good is an LLM in software projects?
  - How secure is the code generated by them?
  - Can they work in a team?
- In this research, we address these questions by proposing a holistic benchmarking framework.

Overview

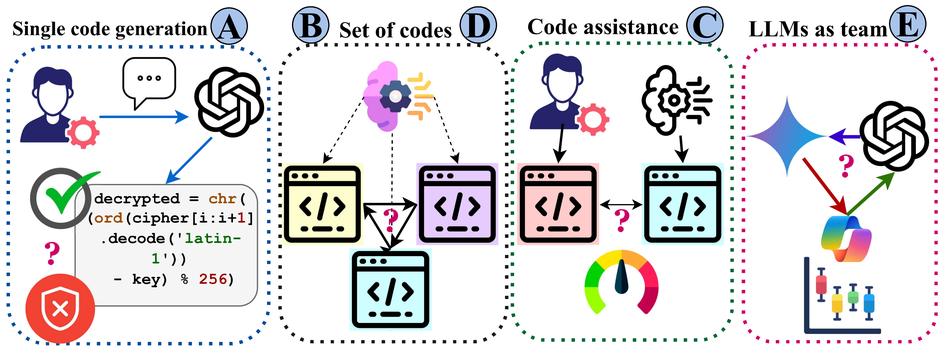

Figure 1: Overview of automated benchmarking

Framework

In our benchmark framework, we holistically consider five entities:

(A) Single code segment quality,

(B) Quality of set of code segments from a single task-specific prompt,

(C) LLM performance as a Co-pilot to assist sub-completed code,

(D) Code comprehensiveness of a single LLM,

(E) Performance of LLMs as a team.

Benchmark Workflow

- The workflow graph represents dependency among predefined code repository, a human, or an autonomous agent.
- The input from code repositories requires a pre-defined, conversational-style text prompt.

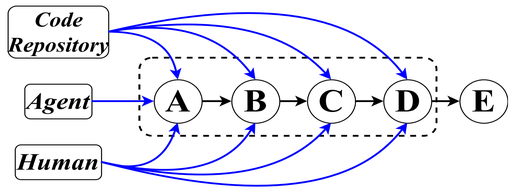

Figure 2: Benchmark workflow graph. Blue edges denote prompts to evaluate an entity.

Benchmark Dependency

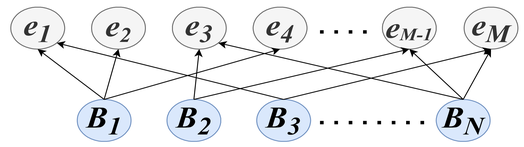

Figure 3: Benchmark dependency graph

- A benchmark is expressed as a set of metrics.
- one might want to calculate another arbitrary set of metrics that falls under more than one benchmark.

Benchmarking Algorithm. Algorithm 1 presents our approach for the automated benchmark.

- Considers two different score aggregators: (i) For code $\mathcal{A}^C(\cdot,\cdot)$, and (ii) For LLM $\mathcal{A}^{\mathcal{L}}(\cdot,\cdot)$
- A generalized property verifier $\mathcal{F}(\cdot,\cdot)$ that can verify properties on a continuous scale.

Algorithm

Algorithm 1 Holistic Benchmarking Algorithm

Input: Properties $\Phi : \{\phi_1, \phi_2, \ldots \phi_d\}$, Generated codes $\zeta : \{C_1, C_2, \ldots C_n\}$, LLMs $\Lambda : \{\mathcal{L}_1, \mathcal{L}_2, \ldots \mathcal{L}_p\}$
Require: Score Aggregators $\Rightarrow$ Code: $\mathcal{A}^C(\cdot,\cdot)$, LLM: $\mathcal{A}^{\mathcal{L}}(\cdot,\cdot)$; Property verifier $\mathcal{F}(\cdot,\cdot)$
Output: Code score set $[e]_n$, LLM score set $[\beta]_p$

```
for a ← 1 to p do
    β^{L_a} ← 0                          ▷ Initially all LLM scores are 0
end for
for i ← 1 to n do
    e^i ← 0                              ▷ Initially i-th code score is 0
    for j ← 1 to d do
        S^i_j ← F(C^{L_a}_i, φ_j), e^i ← A^C(e^i, e^i_j)
        β^{L_a} ← A^L(β^{L_a}, e^i)
    end for
end for
```

Evaluation Model

$$B_{\mathcal{L}}(t_i) = \sum_{j=1}^{m} w_{ij} \hat{f}(t_i, e_j), \quad B_{\mathcal{L}} = \frac{1}{n} \sum_{i=1}^{n} B_{\mathcal{L}}(t_i)$$

- $E = \{e_1, e_2, \ldots, e_M\}$ is the set of evaluation metrics to assess the generated code snippet quality. Evaluation function $f : T \times E \to R$ maps each task t_i and evaluation metric e_j to a real-valued score.
- w_{ij} denote the weight assigned to evaluation metric e_j. Here, $w_{ij} \geq 0$, $\sum_{j=1}^{m} w_{ij} = 1$ for each task t_i.
- $B_{\mathcal{L}} : T \to R$ works as a function of the performance scores obtained across all tasks and evaluation metrics.

Conclusion & Future Research

- We propose a benchmark framework for LLMs based on their code generation capabilities and their ability to be a comprehensive development tool.
- We are developing a platform to integrate multiple open-source and black-box models and property verifiers such as security, code correctness, and hallucination.
- In future, we plan to incorporate more complex tasks, such as static analysis, in the benchmark framework.