CS 6158 Project Proposal

Expanding Benchmarks for LLM-generated Program Efficiency

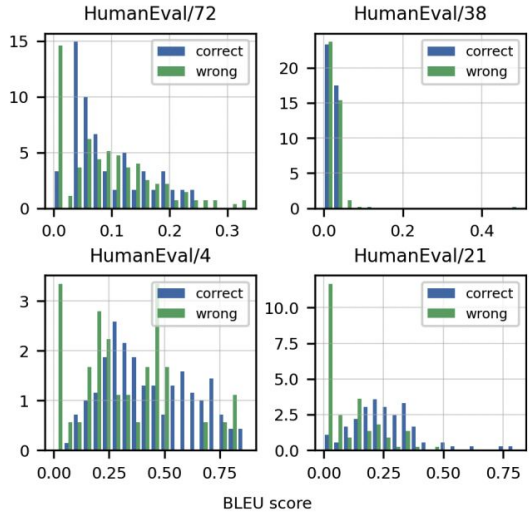
Arnav Joshi (M.Eng) and
Aditya Vinodh (MPS)

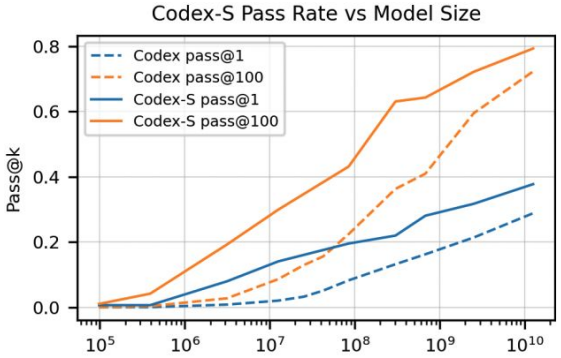
Overview

1. Research Problem
2. Methodology
3. Evaluation
4. Anticipated Challenges
5. Suggestions
6. Deliverables

Research Problem

- Large Language Models like GPT-4 have advanced code generation, enabling automation in writing, optimizing, and debugging code.
- Existing assessments primarily evaluate syntactic, functional correctness and accuracy of generated code.
- Efficiency, in terms of runtime, memory usage, and computational complexity, is often ignored in these evaluations.
- LLMs can produce multiple versions of the same algorithm, not all of which are optimized for resource efficiency.





This study aims to develop a comprehensive benchmark for the efficiency of LLM-generated programs by examining performance metrics such as runtime, memory consumption, and algorithmic complexity.

Methodology

1. Metric Definitions

i. Runtime performance (execution time)
ii. Memory usage (profiling tools)
iii. Redundant variable storage (static code analysis)
iv. Algorithmic efficiency (Big-O complexity)
v. Others

2. Data Curation

a. Gather prompt-code pairs for comparison
b. Identify highly efficient, "gold label" code to benchmark against LLM-generated code

3. Metric Comparison

a. Evaluate and compare LLM-generated code to each other AND benchmark code using defined metrics

4. Refining Metrics

a. Analyze LLM performance gaps
b. Determine which metrics best predict inefficient or suboptimal code

Evaluation

1. **Metrics Accuracy**
   - Validate efficiency metrics (runtime, memory, variable storage, algorithmic complexity) by comparing LLM-generated code with each other.
   - Success: Metrics reflect differences in efficiency through sorting, graph algorithms, dynamic programming, etc.

2. **Baseline Comparison**
   - Compare LLM-generated code to human code using the defined metrics.
   - Success: Framework effectively identifies suboptimal LLM-generated code and highlights areas of efficiency or inefficiency.

3. **Dataset Quality**
   - Ensure dataset covers a wide range of domains with efficient human solutions.
   - Success: Dataset is useful and applicable for future research and development.

Anticipated Challenges

Identifying Efficiency Metrics: Defining meaningful and measurable criteria for program efficiency requires careful selection

Dataset Curation: Difficulty finding datasets to perform evaluation that contain a range of problems spanning domains and solutions that are optimal.

Automating Static and Dynamic Analysis: Evaluating efficiency involves both static analysis and dynamic analysis. Building automated tools will be a significant technical challenge.

Creating a Prompting & Testing Framework/Automation: Technical challenges & software design involved in creating a framework to run tests, similar to EvalPlus' architecture.

Suggestions

Metrics:
- Runtime performance (measured through execution time)
- Memory usage (tracked using profiling tools)
- Redundant variable storage (through static code analysis)
- Algorithmic efficiency (using Big-O complexity analysis)

Dataset Curation: Public data from competitive programming platforms such as LeetCode & Codeforces are ideal. Other datasets like academic datasets may also be useful.

Automating Static and Dynamic Analysis: Tools like pylint can automatically statically test code. Benchmarking suites such as hyperfine or benchmark.js could be used for dynamic testing.

Testing Framework: Technical challenges & software design involved in creating a framework to run tests, similar to EvalPlus' architecture.

Any ideas?

Deliverables

1. Extended EvalPlus

A working extension of EvalPlus, capable of evaluating LLM-generated code based on efficiency metrics.

2. Efficiency Dataset

A curated dataset of programs for benchmarking and further research on code efficiency.

3. Final Report

A comprehensive report detailing methodology, results, and insights on the efficiency of LLM-generated code.

Enhance AI-assisted programming by providing tools and methodologies for optimizing code generation efficiency.

References

Chen, Mark et al. “Evaluating Large Language Models Trained on Code.” *ArXiv* abs/2107.03374 (2021): n. pag.