Benchmarking Educational Program Repair

Charles Koutcheme[1], Nicola Dainese[1], Sami Sarsa[1], Juho Leinonen[2], Arto Hellas[1], Paul Denny[2]

[1]Aalto University, [2]University of Auckland

Aalto University School of Science

1. Introduction

Motivation. The emergence of large language models (LLMs) has sparked enormous interest, particularly in their potential application across various educational tasks. One task showing great promise is program repair, where LLMs can offer valuable feedback to students in the form of debugging support and next-step hints.

Problem. Despite the potential of LLMs in program repair, progress in educational AI is hindered by the use of bespoke datasets and different evaluation metrics in research, leading to unreliable direct comparisons of results. This lack of standardization calls for the need to establish benchmarks that enable equitable comparisons between competing approaches.

Contributions. In this work, we address the challenge by proposing a framework for evaluating program repair techniques using LLMs. We introduce a unified evaluation procedure and curate two high-quality publicly available programming datasets suitable for evaluation. We also provide baseline results of fine-tuned code language models, offering a step toward much-needed standardization in the field.

2. Methods

Problem setup. Picture a student tackling a programming assignment. We have the problem description, associated unit tests, and a grader employing these tests for summative feedback.

Task. Faced with a program failing unit tests, your aim is to train/use an LLM to generate a repair, addressing all issues in the incorrect code. Below, we summarize the proposed unified framework.

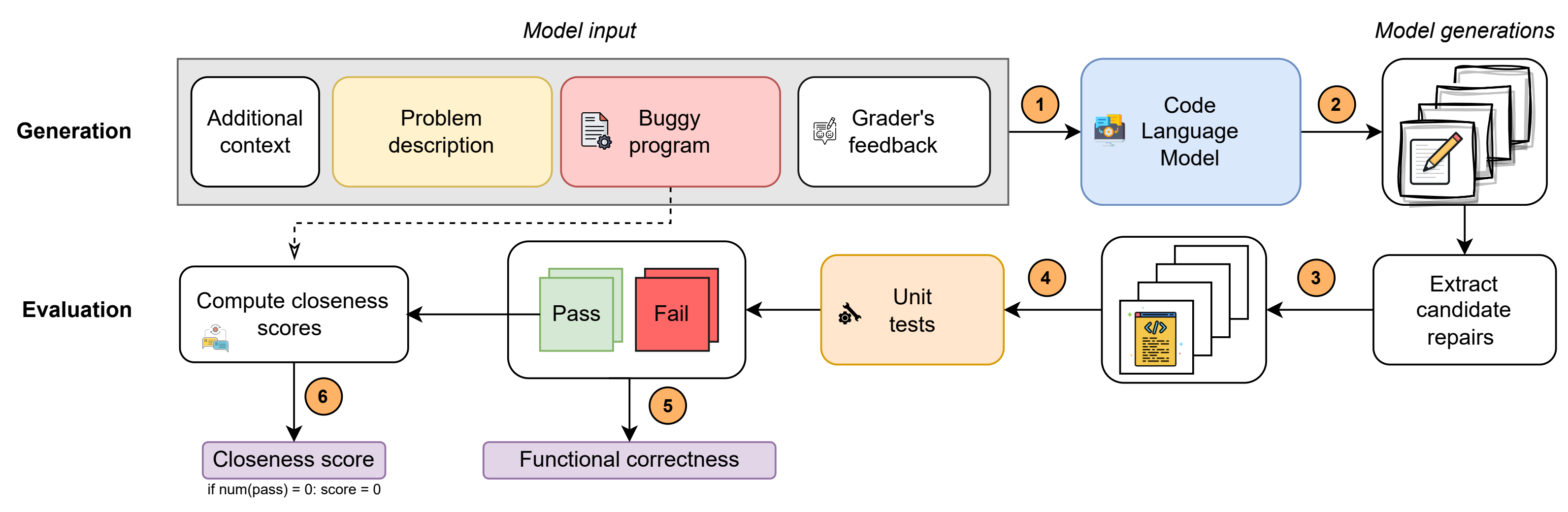

Figure 1: Overview of the problem setup and our evaluation methodology.

Model inputs 1

You can feed an LLM any prompt containing, at a minimum, the problem description and the incorrect student code. Additional contextual information (e.g., instructions) and, optionally, the grader's feedback can also be included in the prompt.

Language Models 1 2

You can evaluate any LLM, whether pretrained or instruction-tuned. You can even fine-tune an existing LLM on your own dataset (what we did).

Generating feedback and extracting candidate repairs 2 3

You would likely generate n outputs from your model, each providing distinct feedback containing at least the candidate repairs (and, optionally, explanations). We will then extract the candidate repairs from these n feedback outputs.

Evaluation procedure 4 5 6

We want the generated programs to be (1) functionally correct, and (2) closely aligned with the original incorrect program (i.e., we want a small distance between the incorrect program and the repairs). We use the following metrics:

(1) Functional correctness - pass@k: Estimates the probability that one of the n candidate repairs passes all unit tests.

(2) Closeness score - rouge@k: A **new metric** estimating the highest ROUGE score among the n candidate repairs.

Datasets

We curated two high-quality programming datasets for evaluating LLMs.

FalconCode

- Language: Python
- Split into three semesters
- Various difficulty levels
- Use case: evaluating LLMs when large amounts of contextual information are available

Singapore

- Language: Python
- Single split
- Homogeneous difficulty level
- Use case: evaluating LLMs when little contextual information is available

3. Experiments

We established baseline performance using pretrained LLMs and fine-tuned them on pairs of *<buggy, correct>* student submissions. This process utilized data from the first semester of the FalconCode dataset, with pairs generated by an Automated Repair Tool. Our study includes models from the **CodeGen** family with 350M and 2B parameters, as well as models from the **StarCoder** family with 164M, 1B, and 3B parameters.

Template for training/prompting our Code Language Models

```
Repair the incorrect program.
<PROBLEM DESCRIPTION>

**Incorrect program**:
<BUGGY PROGRAM>

**Repaired code**:
<CORRECTED PROGRAM> ⓧ
```

4. Results

We report the performance of the supervised baselines on the incorrect submission of the last semester of the FalconCode dataset, split into two levels of difficulty. We also report the performance of the same supervised models on the Singapore dataset.

Table 1: Pass@k for k = $1, 5, 10$

		falconcode_easy			falconcode_hard			singapore		
model	size	k = 1	k = 5	k = 10	k = 1	k = 5	k = 10	k = 1	k = 5	k = 10
starcoder	164M	6.88	12.46	14.68	20.27	35.93	41.85	2.54	7.62	10.46
codegen	350M	12.08	23.19	27.88	13.68	28.28	33.77	3.72	12.92	18.90
starcoder	1B	16.91	30.38	36.06	**26.44**	**44.94**	**50.72**	7.35	21.02	28.77
codegen	2B	16.10	25.99	30.67	13.85	25.92	30.38	11.24	24.97	31.63
starcoder	3B	**19.11**	**31.68**	**37.73**	14.63	29.88	36.51	**12.29**	**29.37**	**36.14**

Table 2: Rouge@k for k = $1, 5, 10$

model	size	k = 1	k = 5	k = 10	k = 1	k = 5	k = 10	k = 1	k = 5	k = 10
starcoder	164M	5.91	10.66	12.57	12.14	23.43	28.32	2.18	6.58	9.07
codegen	350M	10.56	20.14	24.12	9.06	19.32	23.50	2.49	8.66	12.92
starcoder	1B	14.32	25.41	29.93	**17.02**	**30.91**	**35.90**	5.60	15.95	21.68
codegen	2B	13.70	21.91	25.66	10.22	19.14	22.70	7.63	13.59	15.91
starcoder	3B	**15.89**	**26.18**	**31.12**	10.48	21.59	26.54	**7.85**	**19.97**	**25.73**

Observations

Model performance scales with training compute and model sizes. In the same family, performance scales with model sizes and generations. However, across families, smaller LLMs can outperform larger ones.

Fine-tuned language models can overfit their datasets. Small CodeGen models perform better at fixing the 'hard' problems of FalconCode compared to the easy ones, primarily due to overfitting on the training set.

Supervised training can transfer across datasets. Models fine-tuned on the FalconCode dataset can also correct incorrect programs in another dataset, which features distinct problem types and other unique issues.

5. Discussion

Overarching goals. By providing a benchmark for this task, we hope to facilitate future research by allowing easy replication of results.

Future work:

- Transitioning toward a multilingual evaluation benchmark.
- Conducting large-scale evaluations of instruction-tuned and chat models.
- Establishing an online leaderboard.
- Investigating the relationship between program repair quality and feedback quality.

Interested in collaborating? Reach out to us at *charles.koutcheme@aalto.fi*.