Adapting Techniques from Software Testing to Benchmark and Utilize LLMs for Code

Mrigank Pawagi, Viraj Kumar

Indian Institute of Science, Bengaluru

GuardRails: Automated Suggestions for Clarifying Ambiguous Purpose Statements

mrigank.in/GuardRails

Presented at the 6th Annual COMPUTE Conference by ACM India.

Introduction and Context

Large Language Models (LLMs) can generate code from natural language prompts [3]. Although this code is not always accurate, an early study showed that LLM-generated code outperforms novice programmers on simple code-writing tasks [5]. More recent work shows continued improvement, including on more complex programming tasks [8, 10]. As a result, there have been calls to urgently review "our educational practices in the light of these new technologies" [2]. One such review of code-writing tasks has been put forward by Raman and Kumar [11], based on the 6-step recipe proposed by Felleisen et al. [4] to help novice programmers design functions. We focus on two of these steps: Step 2 (Signature, Purpose Statement, Header) and Step 3 (Functional Examples). Before implementing a function, programmers are encouraged to write a *purpose statement* i.e., a short, natural-language explanation of what the function computes. However, a purpose statement may be *ambiguous* i.e., it may fail to specify the intended behaviour when two or more inequivalent computations are plausible on certain inputs.

Motivating Example

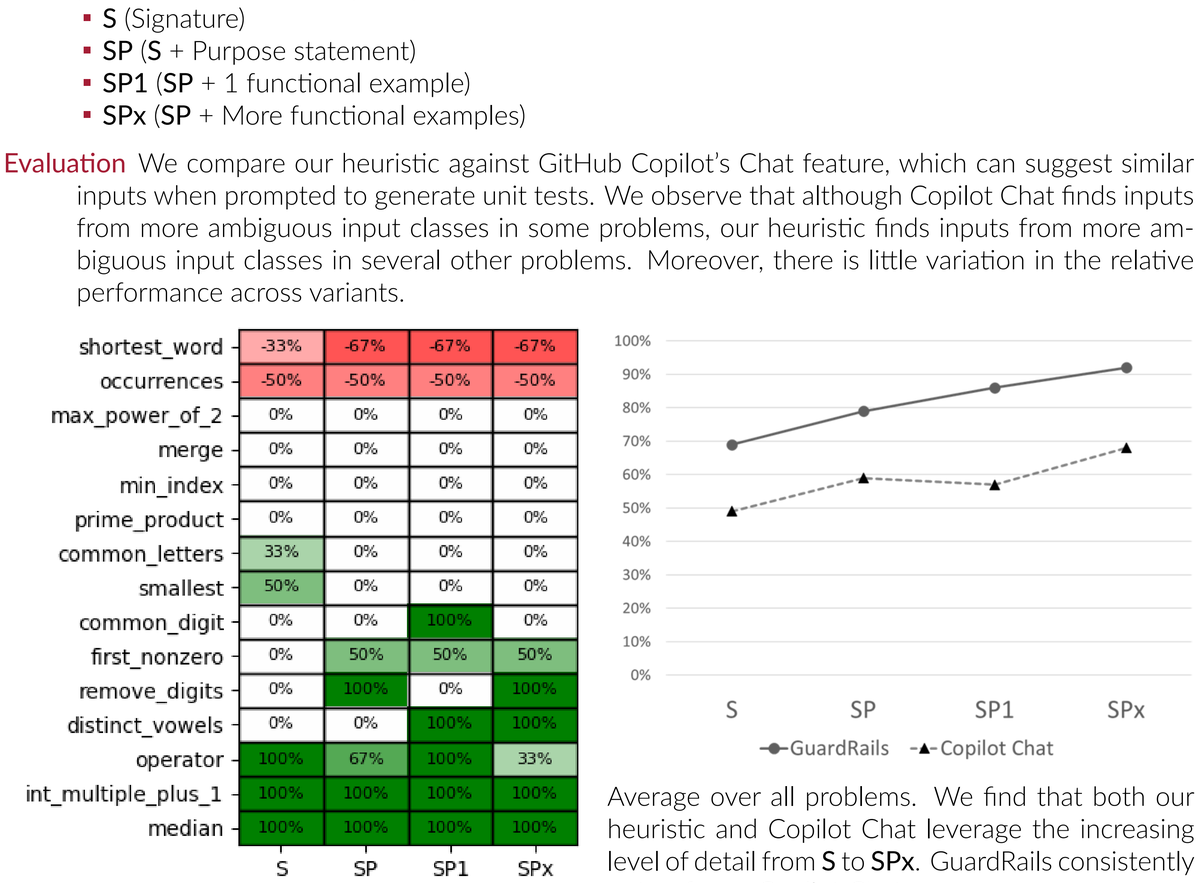
```
def first_nonzero(nums: list[float]) -> float:
    """ Return the first non-zero value in
      nums.

    >>> first_nonzero([0.0 , 3.7 , 0.0])
    3.7
    """
```

This is an example of an incomplete function definition in Python. It has a purpose statement (called the *docstring*) which contains a functional example. The purpose statement is ambiguous because it does not specify the intended behaviour on lists with no non-zero elements.

Contributions

Heuristic We propose a novel heuristic that suggests such inputs using Large Language Models (LLMs). Using these suggestions, the programmer may choose to clarify the purpose statement (e.g., by providing a *functional example* that specifies the intended behaviour on such an input).



Open Dataset To assess the quality of inputs suggested by our heuristic, and to facilitate future research, we create an open dataset of purpose statements with known ambiguities. For each function, we provide four variants of prompts.

- S (Signature)
- SP (S + Purpose statement)
- SP1 (SP + 1 functional example)
- SPx (SP + More functional examples)

Evaluation We compare our heuristic against GitHub Copilot's Chat feature, which can suggest similar inputs when prompted to generate unit tests. We observe that although Copilot Chat finds inputs from more ambiguous input classes in some problems, our heuristic finds inputs from more ambiguous input classes in several other problems. Moreover, there is little variation in the relative performance across variants.

Average over all problems. We find that both our heuristic and Copilot Chat leverage the increasing level of detail from S to SPx. GuardRails consistently outperforms Copilot Chat.

Tool We provide an open-source implementation of our heuristic as an extension to Visual Studio Code for the Python programming language, where purpose statements and functional examples are specified as docstrings and doctests respectively. We believe that this tool will be particularly helpful to novice programmers and instructors.

Synthesizing Thorough Test Cases for LLM Code Generation Benchmarks using Property-Based Testing

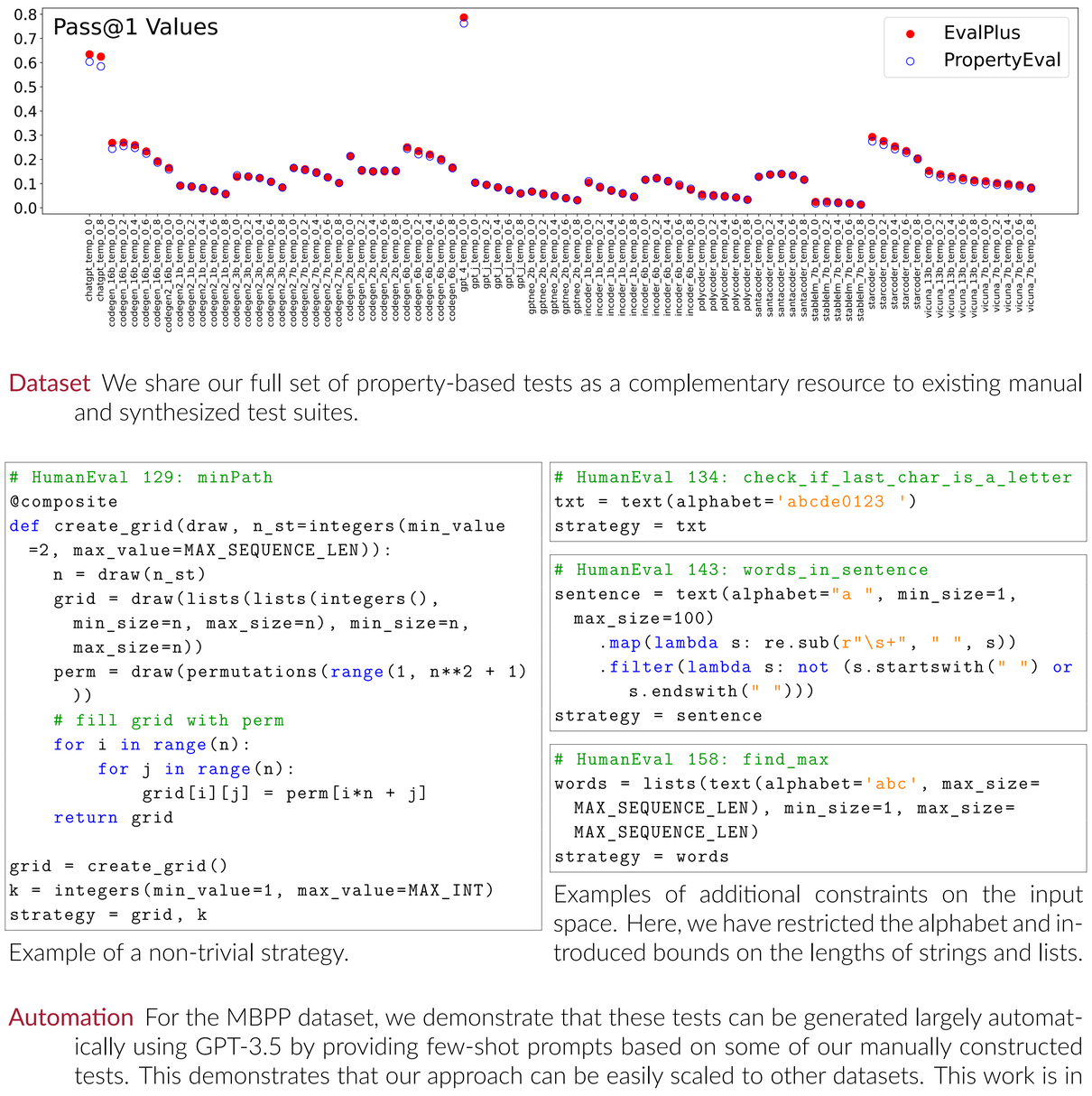
mrigank.in/PropertyEval

Introduction and Context

An important application of Large Language Models (LLMs) is for generating code based on natural-language prompts [3]. Indeed, a growing fraction of professionally developed software is generated by such models [6]. The accuracy of these code generation models is not perfect, but is nevertheless improving [10]. These improvements are tracked using standard programming benchmarks, including HumanEval [3], MBPP [1], and APPS [7]. Each item in a programming benchmark typically consists of (1) a natural-language prompt that describes a coding task, (2) a model solution for that task, and (3) a set of test cases. Recent work by Liu et al. [9] has shown that the test cases for the popular programming benchmark, HumanEval, are inadequate for thoroughly evaluating the correctness of LLM-generated code. Instead, LLM- and mutation-based methods have been proposed to synthesize more thorough test cases.

Contributions

Thoroughness We show that it is possible to further improve the thoroughness of synthesized test cases via Property-Based Testing (PBT), leveraging the canonical solutions within programming benchmarks. For the HumanEval dataset, since adequate property-based tests cannot be automatically generated using rule-based tools, we carefully construct these tests manually. We show that our approach using PBT allows us to synthesize as thorough test cases as those generated using type-aware mutations in Liu et al.'s EvalPlus [9]. However, our approach can be easily adapted to other contexts.



Dataset We share our full set of property-based tests as a complementary resource to existing manual and synthesized test suites.

```
# HumanEval 129: minPath
@composite
def create_grid(draw, n_st=integers(min_value
  =2, max_value=MAX_SEQUENCE_LEN)):
    n = draw(n_st)
    grid = draw(lists(lists(integers(),
      min_size=n, max_size=n), min_size=n,
      max_size=n))
    perm = draw(permutations(range(1, n**2 + 1)
      ))
    # fill grid with perm
    for i in range(n):
        for j in range(n):
            grid[i][j] = perm[i*n + j]
    return grid

grid = create_grid()
k = integers(min_value=1, max_value=MAX_INT)
strategy = grid, k
```

Example of a non-trivial strategy.

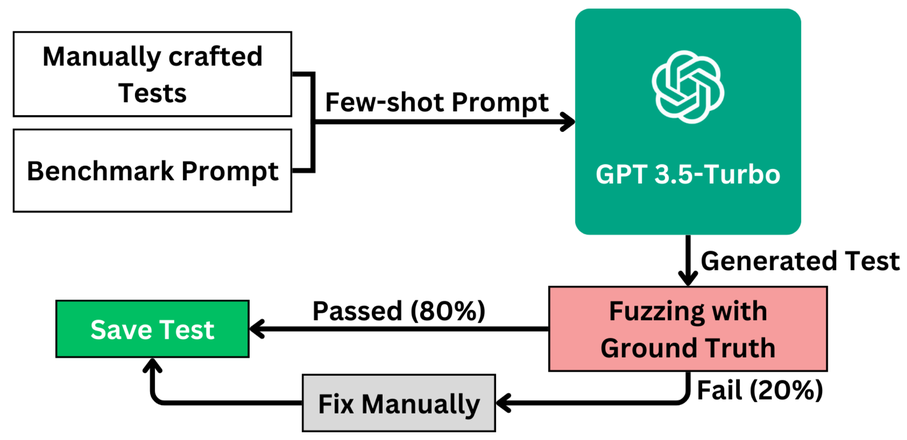
```
# HumanEval 134: check_if_last_char_is_a_letter
txt = text(alphabet='abcde0123 ')
strategy = txt
```

```
# HumanEval 143: words_in_sentence
sentence = text(alphabet="a ", min_size=1,
  max_size=100)
    .map(lambda s: re.sub(r"\s+", " ", s))
    .filter(lambda s: not (s.startswith(" ") or
      s.endswith(" ")))
strategy = sentence
```

```
# HumanEval 158: find_max
words = lists(text(alphabet='abc', max_size=
  MAX_SEQUENCE_LEN), min_size=1, max_size=
  MAX_SEQUENCE_LEN)
strategy = words
```

Examples of additional constraints on the input space. Here, we have restricted the alphabet and introduced bounds on the lengths of strings and lists.

Automation For the MBPP dataset, we demonstrate that these tests can be generated largely automatically using GPT-3.5 by providing few-shot prompts based on some of our manually constructed tests. This demonstrates that our approach can be easily scaled to other datasets. This work is in progress.



References

[1] Jacob Austin et al. "Program Synthesis with Large Language Models". In: *CoRR* abs/2108.07732 (2021). arXiv: 2108.07732. url: https://arxiv.org/abs/2108.07732.

[2] Brett A. Becker et al. "Programming Is Hard - Or at Least It Used to Be: Educational Opportunities and Challenges of AI Code Generation". In: *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. SIGCSE 2023. Toronto ON, Canada: Association for Computing Machinery, 2023, pp. 500–506. isbn: 9781450394314. doi: 10.1145/3545945.3569759. url: https://doi.org/10.1145/3545945.3569759.

[3] Mark Chen et al. "Evaluating Large Language Models Trained on Code". In: (2021). arXiv: 2107.03374 [cs.LG].

[4] Matthias Felleisen et al. *How to design programs: an introduction to programming and computing*. MIT Press, 2018.

[5] James Finnie-Ansley et al. "The Robots Are Coming: Exploring the Implications of OpenAI Codex on Introductory Programming". In: *Australasian Computing Education Conference*. 2022, pp. 10–19.

[6] *GitHub Copilot for Business is now available*. https://github.blog/2023-02-14-github-copilot-for-business-is-now-available/. Accessed: 2024-01-31.

[7] Dan Hendrycks et al. "Measuring Coding Challenge Competence With APPS". In: *NeurIPS* (2021).

[8] Yujia Li et al. "Competition-level code generation with alphacode". In: *arXiv preprint arXiv:2203.07814* (2022).

[9] Jiawei Liu et al. "Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation". In: *arXiv preprint arXiv:2305.01210* (2023).

[10] OpenAI. *GPT-4 Technical Report*. 2023. arXiv: 2303.08774 [cs.CL].

[11] Arun Raman and Viraj Kumar. "Programming Pedagogy and Assessment in the Era of AI/ML: A Position Paper". In: *Proceedings of the 15th Annual ACM India Compute Conference*. COMPUTE '22. Jaipur, India: Association for Computing Machinery, 2022, pp. 29–34. isbn: 9781450397759. doi: 10.1145/3561833.3561843. url: https://doi.org/10.1145/3561833.3561843.