Enhancing Code Generation in Interactive Systems through Additional User Information

Yifan Liu [1] and Eldan Cohen [2]

[1] Department of Computer Science, [2] Department of Mechanical & Industrial Engineering, University of Toronto

BACKGROUND

- **LLMs** have demonstrated strong performance in generating **Python code** from **natural language instructions** [1]
- Previous research focused on enhancing accuracy in **single-turn systems**: multi-agent [2] or prompt engineering [3]
- A lack of research on how **additional information** provided by software engineers after initial failure can improve generation in an **interactive, multi-turn system**

Research Question

How do different types of additional user information impact the correction of generated code with various causes?

We categorize the causes of incorrect code generation into two types:

1. **prompting issues**: initial user instructions are unclear (see Figure 1)
2. **generation issues**: LLMs produce errors despite clear instructions (see Figure 2)

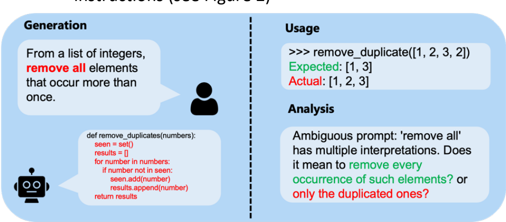

Figure 1: Prompting Issue

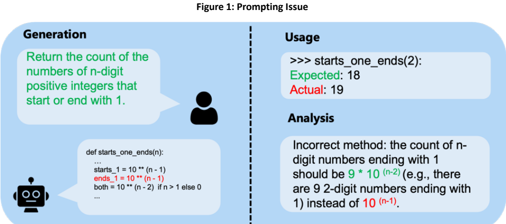

Figure 2: Generation Issue

METHODOLOGIES

The system operates as follows (see Figure 3):

1. The LLM generates the **initial code solution** based on a user input
2. The user provides feedback on the solution's **correctness** and **additional information** for re-generation (simulated in our experiment as illustrated below)

The **additional user information** is defined as follows:

- **Self-reflection [a]**: LLM analyzes the reasons for incorrectness independently
- **Usage Examples [b]**: Correct input - output pairs
- **Failure Examples [c]**: Failed input – output (expected / actual) pairs
- **Incorrect lines [d]**: Incorrect lines identified by the user
- **Incorrect reasons [e]**: Reasons for the incorrectness provided by the user
- **Hints [f]**: Implementation hints from the user
- **Prompt Clarifications [g]**: Clarifications on unclear prompt parts provided by the user

Note: [a], [b], [c] apply to all issue types. [d], [e], [f] are specific to generation issues, and [g] is specific to prompting issues. [b], [c] involve selecting a fixed number of examples from the dataset, while [d], [e], [f], [g] are curated manually by human labelers.

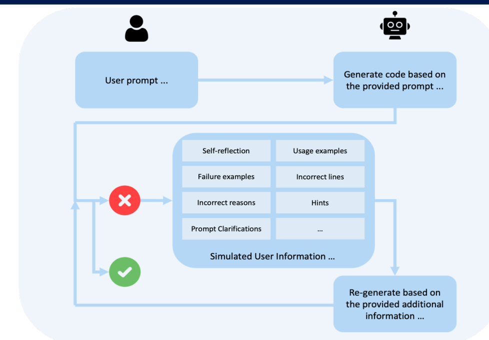

Figure 3: Diagram of the interactive system for code generation and correction

EXPERIMENTS & RESULTS

Experiment Setups:

- Conducted experiments on the **HumanEval** dataset [1], with 164 Python coding questions
- Generated **two paraphrases** to enrich the dataset and ran each **three times** for robustness

  Reported **average percentage of successful correction** of three runs across different information types / causes of incorrect generations
- Tested different **number of usage / failure examples** and **different combination of [d], [e], [f]** (for generation issue only)

Results:

	[1]	[2]	[3]	[4]	[5]	[6]	[7]
Prompting Issues (sample size = 45)	26.7%	46.7%	60.0%	/	/	/	**73.3%**
Generation Issues (sample size = 59)	35.6%	47.5%	62.7%	52.5%	62.7%	**66.1%**	/

Table 1: Average percentage of successful corrections across three runs. The best results are highlighted. "/" indicates the information type is not applicable.

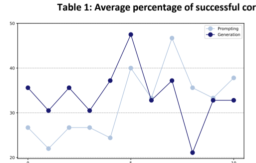

Figure 4: Average correction rate w.r.t. number of usage examples

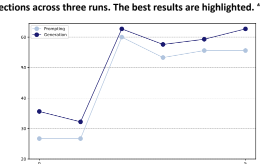

Figure 5: Average correction rate w.r.t. number of failure examples

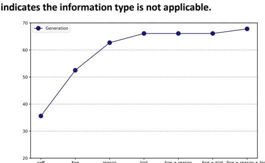

Figure 6: Average correction rate w.r.t. different combinations of additional info.

DISCUSSIONS

- **Prompting issues**: clarifications on prompt are most effective. Further work could investigate different forms of clarifications & how to prompt user for these
- **Generation Issues**: implementation hints are most effective. Adding incorrect lines / reasons may help. Need to test different level of code understanding
- **Usage / Failure examples** are simple yet effective, failure examples are slightly better. Providing **~5 usage examples** or **~2 failure examples** are the optimal.
- Future work could develop an **automated system** to **guide users on the additional information needed** based on the conclusions drawn
- Need a **simple way of interaction** to gather desired information that align with ideal simulated ones

REFERENCES

[1] Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. D. O., Kaplan, J., ... & Zaremba, W. (2021). Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

[2] Huang, D., Bu, Q., Zhang, J. M., Luck, M., & Cui, H. (2023). Agentcoder: Multi-agent-based code generation with iterative testing and optimisation. *arXiv preprint arXiv:2312.13010*.

[3] Denny, P., Kumar, V., & Giacaman, N. (2023, March). Conversing with copilot: Exploring prompt engineering for solving cs1 problems using natural language. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (pp. 1136-1142).