Leveraging static analysis for evaluating code-generation models

Sai Anuroop Kesanapalli, Indrani Panchangam, Abhishek Anand, Vishesh Mittal, Kayvan Shah

CSCI 544: Applied Natural Language Processing
Fall 2023

University of Southern California

Table of Contents

Project Background

- Large Language Models (LLMs) like ChatGPT, GitHub Copilot, and Code Llama have reshaped the coding paradigm with real-time code predictions.

Project Background

- Large Language Models (LLMs) like ChatGPT, GitHub Copilot, and Code Llama have reshaped the coding paradigm with real-time code predictions.
- Studies (Ziegler et al., 2022) indicate that 70% of LLM-generated code is discarded by developers, suggesting potential issues.

Project Background

- Large Language Models (LLMs) like ChatGPT, GitHub Copilot, and Code Llama have reshaped the coding paradigm with real-time code predictions.
- Studies (Ziegler et al., 2022) indicate that 70% of LLM-generated code is discarded by developers, suggesting potential issues.
- Our project performs static analysis on code samples derived from XLCoST dataset and completed using CodeLlama-7b-hf and CodeLlama-7b-Instruct-hfmodel models, focusing on **C++** and **Python** languages.

Table of Contents

Targets

Based on our analysis of (Ding et al. 2023) and other relevant works in this domain, we are working on the following *targets* in this project:

Targets

Based on our analysis of (Ding et al. 2023) and other relevant works in this domain, we are working on the following *targets* in this project:

1. **Broadening horizons:**

Targets

Based on our analysis of (Ding et al. 2023) and other relevant works in this domain, we are working on the following *targets* in this project:

1. **Broadening horizons:**
    - Expanding the scope of static analysis to C++ samples using Cppcheck and Code Llama.
    - Reproducing results on Python samples using flake8 and Code Llama.

Targets

Based on our analysis of (Ding et al. 2023) and other relevant works in this domain, we are working on the following *targets* in this project:

1. **Broadening horizons:**
   - Expanding the scope of static analysis to C++ samples using Cppcheck and Code Llama.
   - Reproducing results on Python samples using flake8 and Code Llama.
2. **Code generation improvement through feedback to model and fine-tuning:**

Targets

Based on our analysis of (Ding et al. 2023) and other relevant works in this domain, we are working on the following *targets* in this project:

1. **Broadening horizons:**
   - Expanding the scope of static analysis to C++ samples using Cppcheck and Code Llama.
   - Reproducing results on Python samples using flake8 and Code Llama.
2. **Code generation improvement through feedback to model and fine-tuning:**
   - Feeding the errors obtained by our re-developed static evaluation framework back to the model, in the form of engineered prompts, and re-evaluating the generated code.
   - Fine-tuning the code generation model by re-training it using additional samples enriched with the knowledge obtained from our static analysis.

Targets

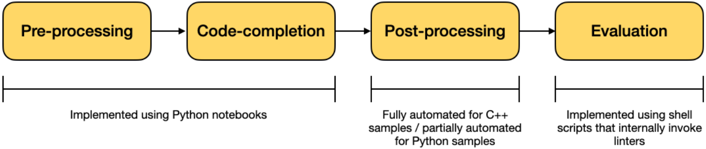
Based on our analysis of (Ding et al. 2023) and other relevant works in this domain, we are working on the following *targets* in this project:

1. **Broadening horizons:**
   - Expanding the scope of static analysis to C++ samples using Cppcheck and Code Llama.
   - Reproducing results on Python samples using flake8 and Code Llama.
2. **Code generation improvement through feedback to model and fine-tuning:**
   - Feeding the errors obtained by our re-developed static evaluation framework back to the model, in the form of engineered prompts, and re-evaluating the generated code.
   - Fine-tuning the code generation model by re-training it using additional samples enriched with the knowledge obtained from our static analysis.

Table of Contents

Pipeline

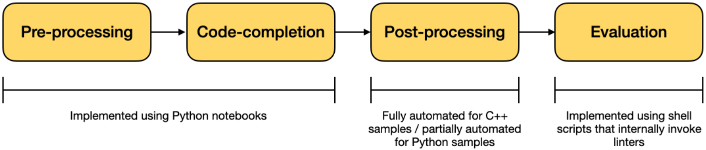
- The model pipeline[†] comprises four stages: **pre-processing**, **code-completion**, **post-processing**, and **evaluation**.



[†] https://github.com/ksanu1998/NLP_Group37

Pipeline

- The model pipeline[†] comprises four stages: **pre-processing**, **code-completion**, **post-processing**, and **evaluation**.

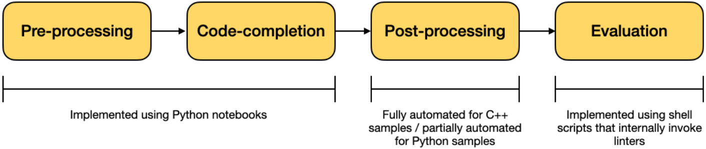


- While quantized versions of the models are more manageable due to their smaller size and faster generation time, for our current and future analyses to be fair across languages and models, we persist with full-size models for our experiments.

[†] https://github.com/ksanu1998/NLP_Group37

Pipeline

- The model pipeline† comprises four stages: **pre-processing**, **code-completion**, **post-processing**, and **evaluation**.



- While quantized versions of the models are more manageable due to their smaller size and faster generation time, for our current and future analyses to be fair across languages and models, we persist with full-size models for our experiments.
- Our experiments were run on T4 GPUs accessible through Google Colaboratory, and on P100 GPUs accessible through Kaggle.

† https://github.com/ksanu1998/NLP_Group37

Table of Contents

Broadening horizons

Expanding the scope of static analysis to C++ samples using Cppcheck and Code Llama

Frequency	Defect ID
54	`syntaxError`
2, 2, 2	`shadowFunction`, `unreadVariable`, `variableScope`
1, 1, 1, 1, 1	`arrayIndexOutOfBoundsCond`, `constParameter`, `missingInclude`, `negativeIndex`, `passedByValue`

Broadening horizons

Reproducing results on Python samples using flake8 and Code Llama

Frequency	Error Code(s)	Description
1301, 189, 15, 7, 6	W291, W293, E225, E275, E712	white-space related
56, 32, 53, 26	E305, E302, W292, W391	blank/newline related
40	E501	line too long
61	W191	indentation contains tabs
17	F821	undefined name
12	F401	imported, unused
11	E999	SyntaxError – cannot generate AST
6	E712	`if` condition related
3	F841	assigned, not used
2	F405	Undefined fuction
2, 1	E741, F403	Miscellaneous

Code generation improvement through feedback to model and fine-tuning

Code generation improvement through feedback to model [Python, 300 samples]

Freq. Before f/b	Freq. After f/b	Difference	Error Code
290	0	290 ↓	W292
48	40	8 ↓	F401
11	4	7 ↓	F821
10	6	4 ↓	E999
3	0	3 ↓	E231
17	16	1 ↓	E501
4	4	0	E741
2	2	0	F403
1	1	0	F523
2	2	0	F821
21	22	1 ↑	E225

Table: Coarse-grained statistics — Improvement with feedback — Python

Code generation improvement through feedback to model and fine-tuning

Code generation improvement through feedback to model [Python, 300 samples]

Repetition Freq.	Error Code
30	F401
19	E225
9	E501
8	E302
4, 4	E741, E999
2, 2	F403, F821
1, 1, 1, 1, 1	F523, F841, W292, E231, F811

Table: Fine-grained statistics — Errors that have repeated despite feedback — Python

Code generation improvement through feedback to model and fine-tuning

Code generation improvement through feedback to model [C++, 300 samples]

Freq. Before f/b	Freq. After f/b	Difference	Defect ID
65	14	51 ↓	`constParameter`
159	114	45 ↓	`unusedFunction`
46	16	30 ↓	`syntaxError`
21	7	14 ↓	`passedByValue`
5	4	1 ↓	`variableScope`
3	3	0	`negativeIndex`
1	1	0	`arrayIndexOutOfBounds`
4	6	2 ↑	`shadowVariable`
11	13	2 ↑	`unreadVariable`

Table: Coarse-grained statistics — Improvement with feedback — C++

Code generation improvement through feedback to model and fine-tuning

Code generation improvement through feedback to model [C++, 300 samples]

Repetition Freq.	Defect ID
111	`unusedFunction`
12	`syntaxError`
5	`unreadVariable`
4	`variableScope`
3	`negativeIndex`
2	`shadowVariable`
1, 1, 1, 1, 1	`constParameter`, `arrayIndexOutOfBounds`, `unusedVariable`, `zerodiv`, `legacyUninitvar`

Table: Fine-grained statistics — Errors that have repeated despite feedback — C++

Remains to be done!

Table of Contents

Analysis

- This preliminary analysis bolsters our hypotheses mentioned during the presentation and proposal that non-EOF errors such as **syntaxError** may be more prevalent in C++ and that OOP-based errors such as **shadowFunction** may be observed.

Analysis

- This preliminary analysis bolsters our hypotheses mentioned during the presentation and proposal that non-EOF errors such as **syntaxError** may be more prevalent in C++ and that OOP-based errors such as **shadowFunction** may be observed.
- Errors like **undefined name** and **unused variables**, as highlighted in (Ding et al., 2023) were also identified in the reproduced results on Python samples.

Analysis

- This preliminary analysis bolsters our hypotheses mentioned during the presentation and proposal that non-EOF errors such as **syntaxError** may be more prevalent in C++ and that OOP-based errors such as **shadowFunction** may be observed.
- Errors like **undefined name** and **unused variables**, as highlighted in (Ding et al., 2023) were also identified in the reproduced results on Python samples.
- Our feedback pipeline seems to yield promising results as the static errors post feedback have reduced in the Python samples by $\sim$ **76%**, based on the coarse-grained statistics we obtained. For C++ samples, this figure is $\sim$ **43%**.

Analysis

- This preliminary analysis bolsters our hypotheses mentioned during the presentation and proposal that non-EOF errors such as **syntaxError** may be more prevalent in C++ and that OOP-based errors such as **shadowFunction** may be observed.
- Errors like **undefined name** and **unused variables**, as highlighted in (Ding et al., 2023) were also identified in the reproduced results on Python samples.
- Our feedback pipeline seems to yield promising results as the static errors post feedback have reduced in the Python samples by $\sim$ **76%**, based on the coarse-grained statistics we obtained. For C++ samples, this figure is $\sim$ **43%**.
- Furthermore, our analysis of Python samples suggests that the repetition errors constitute only $\sim$ **20%** of the errors before feedback, implying that repetitions do not occur often post feedback. For C++ samples, this figure is $\sim$ **45%**.

Analysis

- This preliminary analysis bolsters our hypotheses mentioned during the presentation and proposal that non-EOF errors such as **syntaxError** may be more prevalent in C++ and that OOP-based errors such as **shadowFunction** may be observed.
- Errors like **undefined name** and **unused variables**, as highlighted in (Ding et al., 2023) were also identified in the reproduced results on Python samples.
- Our feedback pipeline seems to yield promising results as the static errors post feedback have reduced in the Python samples by $\sim$ **76%**, based on the coarse-grained statistics we obtained. For C++ samples, this figure is $\sim$ **43%**.
- Furthermore, our analysis of Python samples suggests that the repetition errors constitute only $\sim$ **20%** of the errors before feedback, implying that repetitions do not occur often post feedback. For C++ samples, this figure is $\sim$ **45%**.
- Our results indicate that the feedback works better for Python samples than C++, and is expected as C++ is a stricter language.

Table of Contents

Challenges faced and mitigated

1. Running CodeLlama-7b-hf and CodeLlama-7b-Instruct-hfmodel on T4 GPUs through Google Colaboratory was highly time-consuming, with each instance taking about 100 seconds.

Challenges faced and mitigated

1. Running CodeLlama-7b-hf and CodeLlama-7b-Instruct-hfmodel on T4 GPUs through Google Colaboratory was highly time-consuming, with each instance taking about 100 seconds.
    - Switched to P100 GPUs accessible through Kaggle! AWS and CARC not needed, as thought of initially and mentioned in the status report.

Challenges faced and mitigated

1. Running CodeLlama-7b-hf and CodeLlama-7b-Instruct-hfmodel on T4 GPUs through Google Colaboratory was highly time-consuming, with each instance taking about 100 seconds.
   - Switched to P100 GPUs accessible through Kaggle! AWS and CARC not needed, as thought of initially and mentioned in the status report.
2. Since CodeLlama-7b-hf is not instruction-tuned for chat, the response generated, at times, has duplicate/multiple definitions for some functions.

Challenges faced and mitigated

1. Running CodeLlama-7b-hf and CodeLlama-7b-Instruct-hfmodel on T4 GPUs through Google Colaboratory was highly time-consuming, with each instance taking about 100 seconds.
   - Switched to P100 GPUs accessible through Kaggle! AWS and CARC not needed, as thought of initially and mentioned in the status report.
2. Since CodeLlama-7b-hf is not instruction-tuned for chat, the response generated, at times, has duplicate/multiple definitions for some functions.
   - Addressed through the introduction of the post-processing stage in the pipeline.

Challenges faced and mitigated

1. Running CodeLlama-7b-hf and CodeLlama-7b-Instruct-hfmodel on T4 GPUs through Google Colaboratory was highly time-consuming, with each instance taking about 100 seconds.
   - Switched to P100 GPUs accessible through Kaggle! AWS and CARC not needed, as thought of initially and mentioned in the status report.
2. Since CodeLlama-7b-hf is not instruction-tuned for chat, the response generated, at times, has duplicate/multiple definitions for some functions.
   - Addressed through the introduction of the post-processing stage in the pipeline.
3. Accurate data cleaning in Python was deemed essential but proved tricky due to the risk of introducing subtle errors, especially in indentation.

Challenges faced and mitigated

1. Running CodeLlama-7b-hf and CodeLlama-7b-Instruct-hfmodel on T4 GPUs through Google Colaboratory was highly time-consuming, with each instance taking about 100 seconds.
   - Switched to P100 GPUs accessible through Kaggle! AWS and CARC not needed, as thought of initially and mentioned in the status report.
2. Since CodeLlama-7b-hf is not instruction-tuned for chat, the response generated, at times, has duplicate/multiple definitions for some functions.
   - Addressed through the introduction of the post-processing stage in the pipeline.
3. Accurate data cleaning in Python was deemed essential but proved tricky due to the risk of introducing subtle errors, especially in indentation.
   - We ensured that the observed errors were genuine and not artifacts of our evaluation pipeline.

Table of Contents

Remaining Work

- Fine-tuning the code generation model by re-training it using additional samples enriched with the knowledge obtained from our static analysis.

Remaining Work

- Fine-tuning the code generation model by re-training it using additional samples enriched with the knowledge obtained from our static analysis.
- Cleaning-up code, automating the entire pipeline, and documenting the work.

Table of Contents

References I

[1] Tegawendé F Bissyandé et al. "Popularity, interoperability, and impact of programming languages in 100,000 open source projects". In: *2013 IEEE 37th annual computer software and applications conference*. IEEE. 2013, pp. 303–312.

[2] Tom Brown et al. "Language Models are Few-Shot Learners". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

[3] Harrison Chase. *LangChain*. Oct. 2022. URL: https://github.com/langchain-ai/langchain.

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

References II

[5] Hantian Ding et al. "A Static Evaluation of Code Completion by Large Language Models". In: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 5: Industry Track)*. Toronto, Canada: Association for Computational Linguistics, July 2023, pp. 347–360. DOI: 10.18653/v1/2023.acl-industry.34. URL: https://aclanthology.org/2023.acl-industry.34.

[6] *GitHub Copilot · Your AI pair programmer*. en. URL: https://github.com/features/copilot (visited on 10/09/2023).

[7] Kim Tuyen Le, Gabriel Rashidi, and Artur Andrzejak. "A methodology for refined evaluation of neural code completion approaches". In: *Data Mining and Knowledge Discovery* 37.1 (2023), pp. 167–204.

References III

[8] Augustus Odena et al. "Program Synthesis with Large Language Models". In: *n/a*. n/a. n/a, 2021, n/a.

[9] Sundar Pichai. *An important next step on our AI journey*. en-us. Feb. 2023. URL: https://blog.google/technology/ai/bard-google-ai-search-updates/ (visited on 10/09/2023).

[10] Baptiste Rozière et al. "Code llama: Open foundation models for code". In: *arXiv preprint arXiv:2308.12950* (2023).

[11] Ming Zhu et al. *XLCoST: A Benchmark Dataset for Cross-lingual Code Intelligence*. 2022. arXiv: 2206.08474. URL: https://arxiv.org/abs/2206.08474.

[12] Albert Ziegler et al. "Productivity assessment of neural code completion". In: *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*. 2022, pp. 21–29.

Thank you!