(Joseph) Cole Ramos
07400
16 February 2022

Weeks 3 to 5 - Milestone Report

Dates: 3 February 2022 to 16 February 2022

Progress:

Before the meeting: 3 February 2022 and earlier

- Looked for various static analyzers that could be used to generate the annotations
- Most tools I found did not seem to be promising, but https://pmd.github.io/ , a Java static analyzer, seemed like a good bet

First faculty mentor meeting: 3 February 2022

- Clarified various aspects of the project and information I had read about from recommended papers
    - Though paper (1) seemed to show annotations on their own are insufficient to improve performance without using the 2-step approach, my mentor said this was likely due to the type of model they were using, and large language models like the one we will use should do better
- Clarified various aspects of the state of the art in current program generation models: auto-regressive models and BERT models
- Got a few recommendations for static analyzers to look into:
    - Semmle / CodeQL - Github-owned tool that allows you to make SQL style queries to learn properties about a program
    - https://github.com/google-research/python-graphs - "package is for computing graph representations of Python programs for machine learning applications"
- The main goals for the project at the moment are to find a program annotator/ static analyzer to use, and to find some output-benchmark/ evaluation suite, such as HumanEval
    - The main goal is to find some type of simple weakness in current models (e.x. "The models repeatedly make mistakes in using variables that have not been initialized"), then finding a tool that can annotate that type of information (e.x. add labels for when variables are/ aren't initialized) to augment the training data

Between the meetings:

- Looked through various static analyzer tools and read paper recommendations from first meeting
    - CodeQL seems to be the best option: very flexible, well documented, and used by other researchers

- Read paper (2)
    - Most the paper's techniques likely won't apply to this project: this paper mostly seemed to be focused on task of evaluating code instead of generating it
    - However, it did help me understand the capabilities of Semmle/ CodeQL
- Found and skimmed paper (3)
    - Uses CodeQL to analyze security weaknesses in code generated by Github Codex
    - Could be interesting for later work, and I may be able to copy parts of their methodology for CodeQL, but most the errors they identify besides null-checks will be too difficult to try for a first attempt
- Not a lot of quantitative data available on the types of errors made by current large language models. The best data I could find was this table in Paper (1)
    - This notably was a Java test dataset: the HumanEval dataset is in Python and also has a bit different of a format/ problem statement

Table 1: Percent of Static Checks Passed

	GPTNeo125M	GPTNeo1.3B	CODEX	CODEGPT	GNN2NAG	CNG	NSG
No undeclared variable access	89.87%	90.36%	88.62%	90.94%	47.44%	19.78%	**99.82%**
Valid formal parameter access	NA	NA	NA	NA	25.78%	11.03%	**99.55%**
Valid class variable access	NA	NA	NA	NA	15.40%	12.75%	**99.53%**
No uninitialized objects	93.90%	91.73%	90.82%	94.37%	21.20%	21.56%	**99.01%**
No variable access error	90.36%	90.51%	88.86%	91.32%	28.92%	17.92%	**99.69%**
Object-method compatibility	**98.36%**	98.09%	98.35%	97.84%	21.43%	12.23%	97.53%
Return type at call site	97.38%	98.01%	**98.53%**	97.83%	23.86%	16.40%	98.01%
Actual parameter type	87.03%	86.36%	92.28%	88.71%	9.27%	16.09%	**97.96%**
Return statement type	84.05%	85.09%	88.13%	85.23%	12.34%	9.51%	**90.97%**
No type errors	87.25%	88.13%	91.42%	88.10%	16.31%	13.56%	**97.08%**
Return statement exists	99.61%	99.80%	98.44%	99.57%	94.02%	**99.92%**	97.10%
No unused variables	96.42%	96.46%	96.82%	**97.64%**	20.95%	24.29%	93.84%
Percentage of parsing	98.18%	98.13%	96.41%	97.08%	**100.0%**	**100.0%**	**100.0%**
Pass all checks	65.26%	64.88%	47.49%	67.73%	17.34%	12.87%	**86.41%**

- The documented main documented errors for Github's Codex model (Paper (4)) were mainly in parsing the intent of the docstrings: not something my annotation approach could really fix

Second faculty mentor meeting: 10 February 2022

- Mentor has access to his own pretrained large language model he uses for his research
    - This simplifies my task to mainly be providing annotated training data and evaluating the model on some type of test dataset. I do not need to set up a new pipeline or other framework for the model.
- If the annotator ever finds code that is incorrect in the training data (Fails the annotator's check), then we should just ignore it: generally speaking we do not have a way to tell the model that an example is "wrong", as it learns to simply output text following the patterns of the text it was trained on

    - The closest parallel to negative examples would be attempts in Natural Language Processing to keep models from saying offensive things, but this type of negative examples is also not within the scope of the project
- At this point, we are focusing on the problem of either reducing the model's chances of making the specific mistake of using undeclared variables or preventing type errors
    - Variable errors has the advantage of potentially being easy to induce: run the HumanEval dataset, but replace a lot of the variables in the function arguments/ docstrings to be uncommon variables names

References:

1. Neural Program Generation Modulo Static Analysis - https://arxiv.org/pdf/2111.01633.pdf

2. CodeTrek: Flexible Modeling of Code Using an Extensbile Relational Representation - https://www.cis.upenn.edu/~pardisp/ICLR22_CodeTrek.pdf

3. Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions - https://arxiv.org/pdf/2108.09293.pdf

4. Evaluating Large Language Models Trained on Code - https://arxiv.org/pdf/2107.03374.pdf