2021 Summer Research Program

Instructors: Gustavo Sandoval, Brendan Dolan-Gavitt, Siddharth Garg

August 18, 2021

1. Team Members:

- Shumeng Jia - sj3233 - N18767285
- Zhipeng Zhang - zz2995 - N12016592

2. Title:

A Bug-Injection GPT2 Model fine-tuned on open source C and C++ projects.

3. Problem Description:

The goal of this project is to create realistic bugs by using the pre-trained language model to learn a transformation from non-buggy code to buggy code. Since it will be trained on real bugs and is based on a model that has a good understanding of C source code we should be able to create highly realistic bugs that we can then use to test fuzzing and other bug-finding tools (or even to help train people to recognize bugs when auditing software).

4. Literature:

- Related works:
  https://arxiv.org/abs/1812.10772
  https://software-lab.org/publications/fse2021.pdf
- GPT2 and Transformer:
  https://arxiv.org/abs/1706.03762
  https://www.cs.ubc.ca/~amuham01/LING530/papers/radford2018improving.pdf
  https://jalammar.github.io/illustrated-transformer/
  https://jalammar.github.io/illustrated-gpt2/
- Tutorials:
  https://huggingface.co/nvidia/megatron-gpt2-345m
  https://huggingface.co/blog/how-to-generate
- Github REST API Documents:
  https://docs.github.com/en/rest

5. Data collection:

- Gather a list of the top C/C++ projects by number of stars on GitHub. To select the most active and popular projects, we count number of watches between 06/01/2020 ~ 05/31/2021 from GitHub Archive and retain top 1000 projects.
- For each project, get a list of fixed bugs from the GitHub issue tracker. Identify the git commit that fixes the bug. To make this easier, we will only look at issues that were automatically closed via a commit containing a message like "Fixes Issue #123".
- Use GCC command to preprocess the code, so that the comment and other English sentence will be cleaned.
- Collect patches (a diff of the code before and after the fix) that fix those bugs.
- Context: 5-10 lines around the commit line
- Generate the final csv dataset with CSV library in python.

6. Dataset:

Our dataset is a .csv file that contains 225202 datapoints, 80% are used for training. Each datapoint has four columns, see this example:

	id	before	buggy_code	patched_code

0	cf2166483
	908b35d60
	7e2431549
	cc3141028
	3a80
		static void LaunchDoom(void *unused1, void *unused2)
		static txt_button_t *GetLaunchButton(void)
		{
			char *label;
			switch
			(gamemission)
			{
				const char *label;
				switch
				(gamemission)
				{

The first column is bug id, with this id we can verify the raw commit message where this bug came from and distinguish different datapoint. Then we saved diff of the code in three columns, `[before]` has the codes that appear before the bugs and remain unchanged during fix. `[buggy_code]` and `[patched_code]` contain the bugs and patches respectively. The reason why we save data like this is to reduce the size of the file and make it more easier to know where the bug is exactly so the complete bugs can be obtained by concatenating `[before]` and `[buggy_code]`, the complete patches can be obtained by concatenating `[before]` and `[patched_code]`. This format can also be processed in pytorch easily

7. Model:

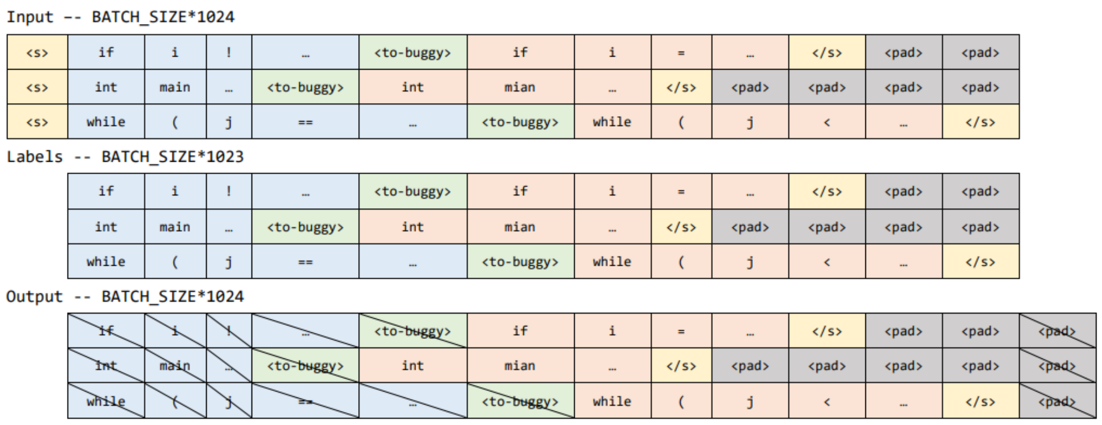
- Model Structure:
  The pre-trained Megatron GPT2 model uses a sequence length of 1024 and an embedding size of 1280. It contains 36 transformer decoder blocks and has 20 attention heads for each attention layer. During fine-tuning, we only updated the parameters in the last decoder and the output linear layer.

- Tokenizer:
  Our model uses byte pair encoding, the pre-trained tokenizer contains 52001 different tokens including six special tokens:
  `"<s>": 0, "<pad>": 1, "</s>": 2, "<unk>": 3, "<mask>": 4, "<|endoftext|>": 52000.`
  We add one additional special token: `"<to-buggy>": 52001` as a delimiter between the patched code and buggy code, so the size of vocabulary is 52002.

8. Training Details:

- Inputs and Outputs:

  The model uses a sequence length of 1024, which means we can have source code context for around 500 tokens long. Each input sequence is composed of the following parts: `<s> + [patched_code] + <to-buggy> + [buggy_code] + </s>`. Then all the sequences are padded to the same maximum length. Since GPT2 model generates one token at a time, the label is exactly the same as the input but moved one word to the right. Therefore, we discard the last token in output sequence since we don't have corresponding label for that token. All the tokens before the delimiter are also discarded when calculate loss because we only care about the bug injection.



Input -- BATCH_SIZE*1024

<s>	if	i	!	…	<to-buggy>	if	i	=	…	</s>	<pad>	<pad>
<s>	int	main	…	<to-buggy>	int	mian	…	</s>	<pad>	<pad>	<pad>	<pad>
<s>	while	(	j	==	…	<to-buggy>	while	(	j	<	…	</s>

Labels -- BATCH_SIZE*1023

if	i	!	…	<to-buggy>	if	i	=	…	</s>	<pad>	<pad>
int	main	…	<to-buggy>	int	mian	…	</s>	<pad>	<pad>	<pad>	<pad>
while	(	j	==	…	<to-buggy>	while	(	j	<	…	</s>

Output -- BATCH_SIZE*1024

~~if~~	~~i~~	~~!~~	~~…~~	~~<to-buggy>~~	if	i	=	…	</s>	<pad>	<pad>	~~<pad>~~
~~int~~	~~main~~	~~…~~	~~<to-buggy>~~	int	mian	…	</s>	<pad>	<pad>	<pad>	<pad>	~~<pad>~~
~~while~~	~~(~~	~~j~~	~~==~~	~~…~~	~~<to-buggy>~~	while	(	j	<	…	</s>	~~<pad>~~

- Loss function:

  `torch.nn.CrossEntropyLoss()`

- Optimizer:

  `torch.optim.AdamW()`

- Hyperparameters:

```
batch size: 8
epoch: 34
learning rate: 2e-5
weight decay: 0.01
```

9. Evaluation:

- Metrics**:**

  We monitor a BLEU score for up to 4-grams using uniform weights (called BLEU-4) for both training and validation.

- Generative method:

  For bug generation with Transformers, our input will be patched code end with the transfer delimiter. We use beam search to do next token selection, number of beams is 5. To evaluate the generated bugs, we calculated four values: BLEU score between the actual buggy code and generated buggy code (BLEU-Model), BLEU score between actual buggy code and source code (BLEU-Baseline), the difference between BLEU-Model and BLEU-Baseline (delta), and the proportion of the realistic bugs among all generated bugs. A positive delta value indicates:

compare to thee patched code, the generated mutant is more similar to the actual bug.

10. Preliminary results:

EPOCH: 34
train loss: 0. 0023349307011812925
train bleu: 0. 9899062428457659
eval loss: 0. 013691503554582596
eval bleu: 0. 944643047271352

After 34 epochs, the BLEU score of training dataset can achieve 0.9899, the BLEU score of validation dataset can achieve 0.944. When look at some generated samples in detail, we found that the model does learn some kind of bug patterns, but it does not always put the desired bug patterns in the desired place. Sometimes we expect a different data type bug, however the model may inject a bug that changes the variable. Also, since the input of model is just a part of code, which is not syntactically correct, the output also might not be syntactically correct.

generate bleu: 0.8711585179924592
generate delta: 0.01676530297004161
generate actual bugs: 0.0003996447602131439

After 34 epochs, the BLEU score of the generated buggy code achieves 0.87, 0.3‰ of them are exactly the same as actual bugs. Delta now becomes a positive number, which means the predicted code represents a translation that is more similar to the buggy code when compared to the original input.

11. Remaining work:

- At present, our dataset only contains context that is close to the commit line (about 5-10 lines around) and the function name, which means the input source code itself is not compliable and not syntactically correct. We are collecting the content in the entire source code right now and plan to gather a new dataset. A small dataset which includes fifty thousand bugs has been shared on /scratch/zz2995, but more bugs can be collected and limited to the time, we haven't done it.
- When the new dataset is ready, we can re-train the model and input a complete program file or a complete function to see how well it generates compliable buggy code.
- Using exactly different repositories to train and evaluate the model might be a good idea and this requires inputting different repository list to the data collection program. However, pay attention to the torvalds/linux repository. Collecting too many bugs from one repository might influence the performance of the model.

12. Source code:

Github repo