Project Report:
Replicating and Extending Text-to-Code Generation with Modality-Relative Pre-training

Sparsh Sehgal (241110071)
Tanmay Roy (231082)
Medikonda Amruth (220642)

November 14, 2025

Abstract

This report details the systematic process of replicating and extending a state-of-the-art text-to-code generation model. We successfully reproduced the core methodology of the baseline paper, including a complex environment setup, implementation of the Modality-Relative Pre-training (MRPT) pipeline with Partial Embedding Separation (PES), and establishment of a baseline evaluation using the `CodeGeeX` framework. We then designed and integrated two novel extensions: (1) a synthetic data augmentation pipeline via docstring paraphrasing and (2) an uncertainty-based filtering technique to improve `pass@k` scores. This report covers the exact commands, methodologies, and technical challenges encountered and resolved during each project phase, as documented in our setup logs.

1 Introduction and Baseline Paper

The goal of this project is to first replicate and then extend the findings of a recent publication in computational linguistics.

1.1 Baseline Paper Details

The foundation of our work is the EACL 2024 paper:

- **Title:** Text-to-Code Generation with Modality-relative Pre-training
- **Authors:** Fenia Christopoulou, Guchun Zhang, Gerasimos Lampouras
- **Conference:** 18th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2024)
- **Date:** March 17-22, 2024

1.2 Core Methodology

The paper's central hypothesis is that Natural Language (NL) and Programming Language (PL) are distinct modalities. The authors propose **Modality-Relative Pre-training (MRPT)**, a continual pre-training stage on a model already trained on a mix of text and code (a MAPT model). Our replication focuses on the paper's most successful configuration:

- **Partial Embedding Separation (PES):** Duplicating embeddings only for Python-specific keywords.
- **CODE-CLM Objective:** Calculating the training loss *only* on the code tokens, not the docstring.

2 Phase 1: Project Setup and Replication

2.1 Environment Setup

A specific environment was required to ensure compatibility.

- The project repository was cloned using `git clone https://github.com/huawei-noah/noah-research.git`.
- `pip` was upgraded, and the `setup.py` file was modified to replace `==` with `>=` for all dependencies *except* `tokenizers` and `transformers`.
- The authors' required versions, `tokenizers-0.12.1` and `transformers-4.19.0`, were strictly maintained.
- A system-specific version of PyTorch was installed to match our server's CUDA 12.4, using the command:

```
pip install --upgrade "torch>=2.1.0" --index-url https://
    ↪ download.pytorch.org/whl/cu124
```

2.2 Dataset and Model Download

- **Dataset:** The training dataset was acquired and placed in `data/python_data.json`
- **Base Model:** The PyCodeGPT 100M model (our MAPT base) was downloaded locally, which saved the model to `models/pycodegpt-100m/`

2.3 Tokenization and Training

This multi-step process prepared the data and executed the MRPT pipeline.

1. **Data Augmentation:** This novel step (detailed in Section 3) was performed first. We ran:

```
python paraphrase_dataset.py --input_file data/python_data.json --output_file data_augmented/python_data.json
```

2. **Tokenization:** We tokenized the *augmented* dataset using the authors' script. This step also implemented PES by referencing `python_tokens.txt`. The command used was:

```
python sample_concatenation.py --main_dir=$(pwd)/ --dataset_dir=../data_augmented --tokenizer=pycodegpt --model_name_or_path=../models/pycodegpt-100m --save_name=pycodegpt_partial_sep --separate_some_embeds="python_tokens.txt"
```

3. **Training Execution:** The `run_model.sh` script was modified to append `"$@"` to accept arguments. We then launched the training, explicitly enabling the `CODE-CLM` objective (`-predict_code=True`) and PES (`-separate_some_embeds`) :

```
bash run_model.sh --model_name_or_path ../models/pycodegpt-100m --dataset_name ./pycodegpt_partial_sep --predict_code=True --separate_some_embeds="python_tokens.txt" --output_dir out_repo_run_pycodegpt
```

3 Phase 2: Evaluation Pipeline and Novel Extensions

With a trained model, we implemented the evaluation pipeline and our novel filtering technique.

3.1 Evaluation Setup (CodeGeeX)

We followed the authors' instructions to set up the `CodeGeeX` evaluation framework.

1. Cloned the `CodeGeeX` repository and installed its requirements:

```
git clone https://github.com/THUDM/CodeGeeX.git
cd CodeGeeX
pip install -e .
```

2. Downloaded the MBPP dataset using the provided script `download_mbpp.py` inside the `CodeGeeX` directory.

3. Patched the `CodeGeeX` framework by copying our `codegeex_changes` files. This was crucial for adding MBPP support and modifying the HumanEval script.

```
# From the text2code_mrpt directory:
cp -r ./codegeex_changes/codegeex/benchmark/mbpp/ ./CodeGeeX/codegeex/benchmark/
cp ./codegeex_changes/scripts/evaluate_mbpp.sh ./CodeGeeX/scripts/
cp ./codegeex_changes/codegeex/benchmark/humaneval-x/evaluate_humaneval_x.py ./CodeGeeX/codegeex/benchmark/humaneval-x/
```

4. Critically, after `CodeGeeX` installed the latest `transformers` library, we reverted to the project's required versions by running `pip install -e .` from our main project directory.

3.2 Baseline Evaluation Command

We ran the standard evaluation to acquire baseline `pass@k` scores, ensuring our renamed model directory `out_repo_run_pycodegpt` was used so that `generation.py` would correctly identify it as a `pycodegpt` model.

```
bash geneval.sh -cgxp ../CodeGeeX -mf pycodegpt -mp ./out_repo_run_pycodegpt -dat mbpp -greedy False
```

3.3 Extension 2: Uncertainty Filtering Evaluation

Our second novel extension involves filtering the generated samples based on "cluster agreement" before calculating the `pass@k` score.

- **Setup:** We installed the `python-Levenshtein` library for fast string-distance calculation.
- **Methodology:** We created a new script, `filter_generations.py`, to perform this filtering. We also modified `geneval.sh` to support this new step. The modified script, `geneval_uncertainty_filter.sh`, first generates 2x the samples, then calls `filter_generations.py` to filter this larger set down to the most consistent `k` samples, which are then evaluated.
- **Execution Commands:**

```
# For MBPP
bash geneval_uncertainty_filter.sh -cgxp ../CodeGeeX -mf
    ↪ pycodegpt -mp ./out_repo_run_pycodegpt -dat mbpp -greedy
    ↪ False -filt True -gfac 2 -incr True

# For HumanEval
bash geneval_uncertainty_filter.sh -cgxp ../CodeGeeX -mf
    ↪ pycodegpt -mp ./out_repo_run_pycodegpt -dat humaneval -
    ↪ greedy False -filt True -gfac 2 -incr True
```

4 Conclusion and Current Status

We have successfully navigated a complex setup and replication process, culminating in a fully functional training and evaluation pipeline for the EACL 2024 paper. We have successfully trained a new model based on our first novel extension (data augmentation) and have fully implemented and debugged the pipeline for our second extension (uncertainty filtering).

We will compare the `pass@k` score of both models *after* applying our docstring augmentation and uncertainty filtering technique.