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

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

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
- **URL:** https://aclanthology.org/2024.eacl-long.72.pdf

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 was downloaded locally to `models/pycodegpt-100m/`. This model serves as our **MAPT (Modality-Agnostic Pre-Training)** base. A MAPT model has already been pre-trained on a large, mixed corpus of natural language and code.

  The key advantage of using a MAPT base is that the model *already* understands the general syntax of Python and the semantics of English. Our training is therefore a *continual pre-training* or *fine-tuning* step (MRPT) that teaches the model to differentiate between the two modalities. This is far more efficient and effective than training a model from scratch.

2.3 Tokenization and Training

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

1. **Data Augmentation:** We first expanded our training data by creating paraphrased (semantically similar, but textually different) versions of the docstrings. The goal was to make the model more robust and less reliant on specific phrasing. This was achieved by running:

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

2. **Tokenization:** This step converted the raw (docstring, code) pairs from the augmented dataset into a numerical format the model understands. This process formats the data using special tokens to explicitly separate the natural language modality from the programming language modality. This was run with:

```
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:** We launched the training, which was configured for our specific methodology. For the **CODE-CLM objective**, the trainer was set to calculate loss only on code tokens (ignoring docstring tokens). For **PES**, the trainer recognized the duplicated Python keywords and trained their new embeddings independently from their natural-language counterparts. This was run via:

```
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. Patched the `CodeGeeX` framework by copying our `codegeex_changes` files.

```
# From the text2code_mrpt directory:
cp ./codegeex_changes/codegeex/benchmark/humaneval-x/
   ↪ evaluate_humaneval_x.py ./CodeGeeX/codegeex/benchmark/
   ↪ humaneval-x/
```

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

3.2 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 modified `geneval.sh` to support a new 3-stage pipeline:
  1. **Generate (N*k) Samples:** We generate a larger pool of samples (e.g., 400 samples instead of 200) by using the `-gfac 2` flag.
  2. **Filter to k Samples:** The new `filter_generations.py` script is called. This script groups the 400 samples by task ID. For each task, it computes a distance matrix using Levenshtein distance. It then calculates an "agreement score" for each sample based on how many other samples it is "close" to.
  3. **Evaluate k Samples:** Only the top k (e.g., 200) samples with the highest agreement scores (the most consistent, "low-uncertainty" cluster) are saved and passed to the `CodeGeeX` evaluator.
- **Execution Commands:**

```
# 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 Phase 3: Evaluation Results

This section outlines the performance metrics used and presents a direct comparison of our extended model against the baseline results reported in the original paper.

4.1 Metric Definitions

- **pass@k:** This metric measures the probability that at least one of k generated code samples correctly passes all unit tests for a given problem. For example, `pass@10` is the chance of finding a correct solution within 10 attempts.
- **Incremental pass@k (micropass@k):** This is a more granular metric proposed by the authors that evaluates the model's ability to complete partial code snippets in addition to full-function generation. The final score is a *micro-average* of all sub-problems, grouped by their original task ID.

4.2 Results and Comparison

We ran our final model, which incorporates both **Data Augmentation** and **Uncertainty Filtering**, against the HumanEval dataset. The table below compares our results to those reported in the paper for the equivalent `PyCodeGPT-100M (CODE-CLM, partial)` model.

Table 1: Comparison of HumanEval `pass@k` Scores

Metric Type	k-Value	Original Paper	Our Model (Augmented + Filtered)
Standard	pass@1	12.80	9.18
Standard	pass@10	16.47	18.02
Standard	pass@100	26.25	29.27
Incremental	pass@1	17.13	15.18
Incremental	pass@10	32.83	35.69
Incremental	pass@100	52.42	59.73

4.3 Analysis of Results

The results are conclusive. Our model **underperformed** the original paper on single-shot generation (`pass@1` and incremental `pass@1`).

However, our model **outperformed** the original paper at `pass@10` and `pass@100` for both the standard and incremental metrics. This suggests our approach successfully generates a more diverse and high-quality set of candidates, which is highly valuable for practical applications where a user can select from multiple generated solutions.

5 Conclusion

We have successfully replicated the core methodology of the EACL 2024 paper, "Text-to-Code Generation with Modality-relative Pre-training." In doing so, we navigated a complex environment setup and reproduced the authors' `PES + CODE-CLM` training and evaluation pipeline.

Furthermore, we successfully designed and implemented two novel extensions: a data augmentation pipeline using docstring paraphrasing and an evaluation-time uncertainty filtering technique. Our final results demonstrate the value of these extensions. While

our model showed a decrease in `pass@1` performance, it achieved superior results for `pass@10` and `pass@100` on both the standard and incremental HumanEval benchmarks. This indicates that our combined approach produces a more robust and varied set of high-quality solutions, ultimately improving the model's practical utility in multi-candidate generation scenarios.

Project Repository

All source code for this project are available at the following GitHub repository:

https://github.com/24kTanmay/text2code_cs787/tree/master