Retrieval-Augmented Generation Based Code Generative System with Unlimiformer

Chuangji Li, Shizhuo Li, Alan Wang
{chuangjl, shizhuol, minyangw}@andrew.cmu.edu

Abstract

To address existing challenges on Large Language Models on code generation tasks and to improve the reliability of generative code, in this project, we designed a retrieval augmented generation (RAG) based code generation system, that incorporates in system code execution and evaluation, and regeneration of code in case of error. The RAG database is constructed by large python code dataset. In this project, we incorporated Code Llama as the generative model as well as the benchmark for experiments. Experiments are conducted based on two testing datasets of python code, and results are compared with the benckmark.

Introduction

Intersection of natural language processing (NLP) and programming code generation is a research area crucial for advancing software development and enhancing how humans interact with computers. It focuses on using large language models (LLMs) to interpret human language and produce executable code, aiming to reduce the gap between human intent and machine action. Despite progress, the sector faces challenges such as ensuring the accuracy of the generated code, incorporating external data sources, and ensuring the models' effectiveness across different programming challenges.

Our project is situated within this area of research, inspired by recent developments but also recognizing the challenges inherent in generating code from natural language descriptions. The objective is to enhance the generation of Python code from natural languages.

We replicated the findings of the seminal work by Code Llama, which sets a benchmark of this project. Our test on code llama with some prompts shows it insufficient performance some code generation task, that the output code may be not valid, may not compile or execute correctly. Our project is then build on top of code llama, and is to improve the generated code quality and validity.

Subsequently, we plan to integrate the principles of Retrieval-Augmented Generation (RAG) into the Code Llama framework. RAG, by leveraging a vast repository of code snippets and documentation, introduces an external knowledge dimension to the model, potentially enriching the generated code's accuracy, relevance, and efficiency. Followed by the RAG, we constructed a code execution system, this allows evaluation of the output code generated by the LLM after retrieval by executing the python code. We conducted evaluation on our system after its construction. The architecture of our system is shown in next section.

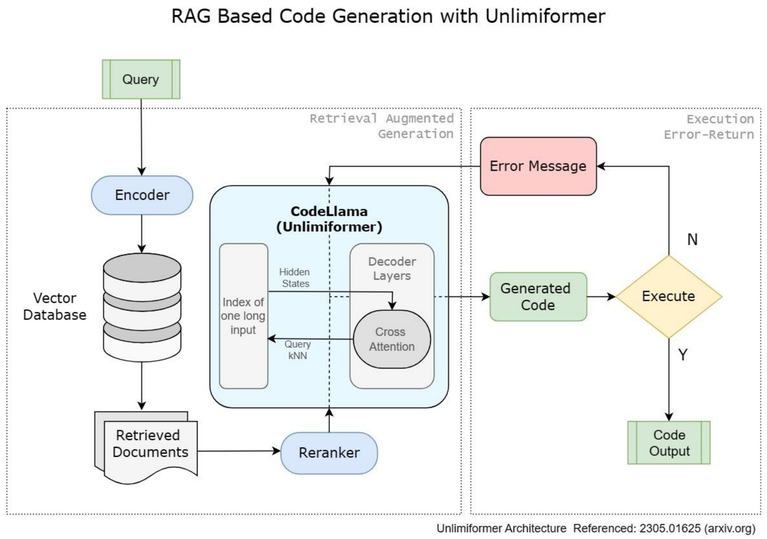
Architecture

Our system consists the Retrieval Augmented Generation (RAG) System and a code execution system. The RAG consists of a query encoder, a vector database, a retriever, a reranker and Code Llama.

The encoder is to encode the natural language query input, and prepare for retrieval process in the vector database. The retriever is retrieve top closely related documents, after reranking, prompts are then constructed by the question and retrieved python code as reference. Prompts are fed to code llama for code generation.

Followed by the RAG, we constructed a code execution system, this allows evaluation of the output code generated by the LLM after retrieval by executing the python code. Error messages are collected and may be fed back to the LLM and are incorporated in the prompt for regeneration. The upper limit of regeneration is decided by user.

We also adapted unlimiformer, a sliding windowed attention based transformer that accepts unlimited length of input, for the generative model, in order to avoid the issue when prompt gets very long under the previous mechanism. Architecture of our system detail is shown below.



Unlimiformer Architecture Referenced: 2305.01625 (arxiv.org)

Database

Our RAG database is constructed by subset of large python code dataset, with natural language descriptions. Here is a summary of dataset that vector database is subsets on.

Dataset	Description
CodeNet	Project CodeNet is a large-scale dataset with approximately 14 million code samples, each of which is an intended solution to one of 4000 coding problems.
CodeSQA	Python dataset with 20000 question-answer pairs, including signature, docstring, body
CodeContests	It consists of programming problems including test cases in the form of paired inputs and outputs, as well as both correct and incorrect human solutions in a variety of languages.
CodeXGlue	CodeXGLUE includes a collection of 10 tasks across 14 datasets and a platform for model evaluation and comparison.
CodeNaLa	2379 training and 500 test examples manually annotated and 598,237 mined intent-snippet pairs. Every example has a natural language intent and its corresponding python snippet.

Experiment

We evaluated our RAG on code generation tasks using benchmark datasets HumanEval and MBPP, and compared the result with CodeLlama-Python-7b produced. We choose evaluate pass@1 and pass@10 for HumanEval and MBPP. Results are shown below.

Model	HumanEval		MBPP	
	pass@1	pass@10	pass@1	pass@10
CodeLlama7b - *Reproduction*	35.3%	64.6%	44.7%	66.9%
CodeLlama7b - *Reported in Paper*	38.4%	70.3%	47.6%	70.3%
RAG-Based, CodeLlama7b	38.3%	66.8%	43.1%	65.1%
RAG-Based, ErrMsg, CodeLlama7b	38.5%		40.8%	
RAG-Based, ErrMsg, Unlimiformer, CodeLlama7b				

Discussion

We have a increase of pass rate after adding the RAG system on HumanEval dataset comparing with Codellama without RAG, showing that our system is to a extent efficient. Where after adding RAG, pass rate decreased for MBPP, probably due to database does not contain enough information useful to coding questions in MBPP.

Adding Error message feedback system does not create observable improvement of performance. One possibility is error message does not always reflect true error of the function.
The next step of this project will be enlarging the vector database, re-evaluating the systems, and adding unlimiformer to the model.

Contact

Chuangji Li, Shizhuo Li, Alan Wang
{chuangjl, shizhuol, minyangw}@andrew.cmu.edu
Carnegie Mellon University
Pittsburgh , PA, 15213

References

1. Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2024. Code llama: Open foundation models for code. [2308.12950] Code Llama: Open Foundation Models for Code (arxiv.org)
2. Amanda Bertsch, Uri Alon, Graham Neubig, and Matthew R. Gormley. 2023. Unlimiformer: Long range transformers with unlimited length input. [2305.01625] Unlimiformer: Long-Range Transformers with Unlimited Length Input (arxiv.org)
3. Shirley Anugrah Hayati, Raphael Olivier, Pravalika Avvaru, Pengcheng Yin, Anthony Tomasic, and Graham Neubig. 2018. Retrieval-based neural code generation. [1808.10025] Retrieval-Based Neural Code Generation (arxiv.org)