Mixture of Expert Model for Code Generation

TEAM 11
Adithya Kameswara Rao akameswa@andrew.cmu.edu
Santhoshkumar Panneerselvam spanneer@andrew.cmu.edu
Yihao Jia yihaoj@andrew.cmu.edu
Yirui Zhu yiruiz@andrew.cmu.edu

1 Introduction

Deep learning, as a specialized branch within the machine learning domain, exhibits an increasingly commanding prowess in content generation. While the consensus suggests that complex models tend to achieve superior outcomes, the necessity for advanced GPUs and the associated costs render such endeavors viable only for behemoth corporations. In this context, the Mixture-of-Experts (MoE) [1] method emerges as a pioneer that democratizes deep learning by circumventing the prohibitive costs associated with simpler models but equivalent performance. Essentially, MoE segments the problem space into discrete sub-regions, each addressed by specialized, simpler models or 'experts,' rather than employing a monolithic model for the entire domain. With adept integration, such an MoE framework can achieve performance parity with larger counterpart models, albeit in a more economically feasible manner.

Taking the realm of code generation as a point of reference, and as depicted in Figure 1, we propose a MoE framework that deftly selects the C language expert model for output generation if the input code snippet is C language. Analogously, when presented with Python code, the system channels the task to the expert model versed in Python for processing. This sophisticated architecture empowers the system to harness the prowess of distinct expert models tailored to various programming languages, thereby significantly enhancing the precision of outputs over the conventional baseline model.

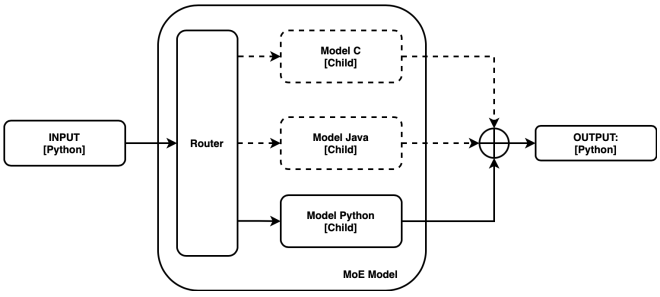

Figure 1: Applying MoE Architecture on Code Generation

In our pursuit, we endeavor to meticulously examine the Mixture-of-Experts (MoE) paradigm as it pertains to code generation. Presently, Large Language Models (LLMs) tasked with code generation typically function as monolithic systems across a variety of programming languages. Although these models are capable of generating competent outputs, their efficacy may wane when tasked with specific programming languages. For instance, a sophisticated model might exhibit superior performance in generating code for Python as opposed to Java and C++, despite all these languages falling well within its realm of expertise. Such observations compellingly suggest that adopting the MoE framework for code generation can be both intellectually sound and potentially advantageous. Given that code generation naturally divides into discrete sub-domains—namely, different programming languages—the assignment of specialized experts within an MoE model for each language offers a significant opportunity to surpass the capabilities of a singular, complex model in handling multiple programming languages.

2 Literature Review

MoE stands as a cutting-edge approach, leveraging a sophisticated model ensemble to handle diverse and complex tasks efficiently. The powerful fusion of MoE and a code-gen model directs specialized expert models, elevating its capabilities in code generation for different programming languages.

2.1 Mixture-of-Experts (MoE)

Mixtral [1]: The "Mixtral of Experts" paper introduces the Mixtral 8x7B model, an advanced sparse MoE language model built upon the Mistral 7B architecture. It features eight feedforward blocks per layer and selects two experts for each token. Overall, it boasts 47B parameters but utilizes only 13B during inference. Thus, this model excels in computational efficiency and performance, surpassing benchmarks set by models like GPT-3.5 and Llama 2 70B in areas including mathematics, code generation, and multilingual translation.

Gemini [2]: Gemini 1.5, boasting significantly improved performance and greater speed over its predecessor, Gemini 1.0, primarily attributes its advancements to the incorporation of the MoE framework into its foundational architecture. As claimed by Google, Gemini 1.5 outperforms 87% in benchmark experiments through the integration of the MoE approach within models such as GShard-Transformer, Switch-Transformer, M4, among others.

SegMoE [3]: SegMoE, introduced by Segmind, is an innovative open-source framework designed to enhance text-to-image generation. It combines multiple generative image models into more comprehensive and efficient systems, utilizing Stable Diffusion's architectural principles enhanced with the sparse MoE layers. This addition optimizes expert selection for each token, aiming to significantly boost image quality and prompt accuracy.

2.2 Code Generation Model

SantaCoder [4]: SantaCoder is a 1.1 billion parameter decoder-only transformer architecture designed for code generation tasks, trained on Java, JavaScript, and Python from The Stack dataset. Building upon earlier efforts in the BigCode community, the model integrates Multi Query Attention (MQA) and Fill-in-the-Middle (FIM) techniques, along with preprocessing filters. After 600,000 iterations of training, SantaCoder outperforms larger models on code generation and infilling tasks across Java, JavaScript, and Python programming languages.

Phi-2 [5]: Phi-2 is a large language model based on the Transformer architecture, comprising approximately 2.7 billion trainable parameters. Its training data encompassed the same sources utilized for the earlier Phi-1.5 model, supplemented by an additional corpus consisting of synthetically generated natural language processing (NLP) texts and filtered web content. Phi-2 demonstrated performance nearing the state-of-the-art among models constrained to fewer than 13 billion parameters on commonsense reasoning, language comprehension, and logical inference capabilities.

3 Baseline Model Ideas

Our strategy encompasses two primary phases: construction and evaluation. In the construction phase, our efforts will focus on 1) Selecting a code generation model of exceptional versatility, capable of accommodating a broad spectrum of programming languages; 2) Tailoring this model through precise fine-tuning to create dedicated experts for each coding language; 3) Integrating these specialized models into a cohesive MoE model.

For the evaluation phase, we will employ the MultiPL-E system [6], which acts as a multi-programming language evaluation framework, for model benchmarking. Basically, MultiPL-E employs pass rates as a metric, evaluating the probability that a model's output will successfully resolve a specified problem within a given number of attempts. This metric offers a quantifiable gauge of a model's proficiency in producing functionally accurate code across diverse programming languages and tasks.

Upon the conclusion of this project, a thorough analysis will be articulated, encapsulating the efficacy of the MoE model in juxtaposition with the conventional universal model.

4 Dataset Description

Through preliminary research, our datasets will be drawn from the BigCode [7] database, which contains 30 programming languages. The original dataset, initially sized at approximately 102TB, underwent filtering processes, including near-deduplication and license filtering, resulting in a reduced size of 3TB. The filtering methodology follows the approaches outlined in Codex[8] and involves removing files that meet the following criteria:

1. Average line length exceeding 100 characters.
2. Maximum line length surpassing 1,000 characters.
3. Alphanumeric character fraction less than 25%.
4. Remove auto-generated files like config, etc.

Dataset Structure

Each data instance corresponds to a file, with its content stored in the "content" feature.

Data Fields

Column Name	Type	Description
content	string	file content
size	integer	uncompressed file size
lang	string	programming language
avg_line_length	float	average line length
max_line_length	integer	maximum line length
alphanum_fraction	float	fraction of alphanumeric characters
hexsha	string	unique git hash
max_stars_repo_path	string	path in the repository with the maximum stars
max_forks_repo_path	string	path in the repository with the maximum forks
max_issues_repo_path	string	path in the repository with the maximum issues
max_stars_repo_name	string	name of the repository with the maximum stars
max_forks_repo_name	string	name of the repository with the maximum forks
max_issues_repo_name	string	name of the repository with the maximum issues
max_stars_repo_head_hexsha	string	repository head hexsha for repo with maximum stars
max_forks_repo_head_hexsha	string	repository head hexsha for repo with maximum forks
max_issues_repo_head_hexsha	string	repository head hexsha for repo with max issues
max_stars_repo_licenses	string	repository licenses for repo with maximum stars
max_forks_repo_licenses	string	repository licenses for repo with maximum forks
max_issues_repo_licenses	string	repository licenses for repo with maximum issues
max_stars_count	integer	number of stars in repository with maximum stars
max_forks_count	integer	number of forks in repository with maximum forks
max_issues_count	integer	number of issues in repository with maximum issues

For the fine-tuning process, we will select a dataset of 4 programming languages based on the compute feasibility and evaluation procedures. Once selected, a subset (with a statistically significant sample size) of the dataset for respective programming languages will be chosen and worked with.

References

[1] **Mixtral of Experts**, arXiv:2401.04088

[2] **Gemini 1.5:** `https://blog.google/technology/ai/google-gemini-next-generation-model-february-2024`

[3] **Segmind's SegMoE:** `https://blog.segmind.com/introducing-segmoe-segmind-mixture-of-diffusion-experts/`

[4] **SantaCoder: don't reach for the stars!**, arXiv:2301.03988

[5] **Phi-2:** `https://huggingface.co/microsoft/phi-2`

[6] **MultiPL-E: Scalable and Extensible Approach to Benchmarking Code Generation** arXiv:2208.08227

[7] **BigCode:** `https://www.bigcode-project.org/docs/about/the-stack/`.

[8] **Evaluating Large Language Models Trained on Code**, arXiv:2107.03374