Scaling Down Multilingual Language Models of Code

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

The democratization of AI and access to code language models is a pivotal goals in the field of artificial intelligence. Large Language Models (LLMs) have shown exceptional capabilities in code intelligence tasks, but with a high computational costs. This paper addresses these challenges by presenting a comprehensive approach to scaling down Code Intelligence LLMs. We focus on training smaller code language models, which lowers the computation cost of inference and training. We extend these models to diverse programming languages, enabling code completion tasks across various domains.

1 Introduction

In recent years, Large Language Models (LLMs) have emerged as powerful tools with applications spanning various domains. Notably, their effectiveness in code intelligence tasks has been remarkable, leveraging the structured nature of programming languages. Accessibility to code LLMs is one of the important goals of AI development as seen by the increasing amount of open-source models and datasets. However, this may not be enough, as the barrier to tuning and using LLM is higher than just access to their weights. For the practitioner, the choices of model architecture and learning algorithms are not obvious, and exploring these options is costly due to the high computation costs. The aim of this project is to address the accessibility gap and further efforts towards democratizing the use and development of code LLMs. In this work, We extend the capabilities of a mono-lingual code LLM originally trained on Python to encompass a diverse range of programming languages including Java, Rust, Ruby, and Swift. This expansion is achieved efficiently by employing Parameter Efficient training techniques and various training optimization methods.

2 Methodology

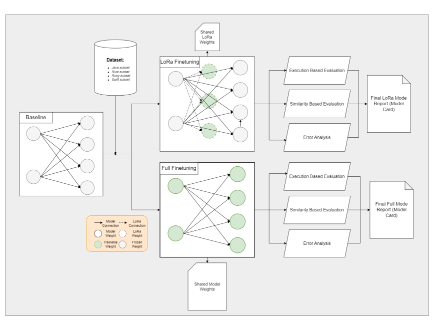

Figure 1: Project Diagram

The first step in the pipeline in Figure 1 is the selection of the baseline model to be fine-tuned. Processing the training dataset is the next step. The sampling of training data from the TheStack corpus (2), file filtering, and generation of the training and validation splits were all standardised in this step. The fine-tuning stage in which we compare between full fine-tuning and LoRa fine-tuning. The fine-tuning can be completed on a free Google Colab T4 GPU in less than 10 hours, depending on the method and parameters chosen. Finally, we create an evaluation report with different results and analyses, then share the generated model and dataset cards along with the trained weights.

3 Fine-tuning Small code language models on Low Resource Languages

To test our approach, we fine-tuned our baseline model in four different programming languages: Java, Ruby, Rust, and Swift. These languages were chosen because they represent varying levels of availability in the Stack dataset. We train, test, and share four distinct LoRa adapters using our preprocessed datasets. We used the CodeGen-Mono baseline (3) and fine-tune the model on the processed datasets. The Pass@10 rates are illustrated in Figure 2, which depicts the evaluation of the four models along with the monolingual and multilingual baselines. The evaluation was carried out across 161 code completion problems from MultiPl-E (1), spanning five different programming languages. As anticipated, the monolingual baseline achieves the highest Pass@10 rate in Python. Conversely, the multilingual baseline, trained on a vast corpus of 119.2B tokens encompassing C, C++, Go, Java, JavaScript, and Python, scores notably lower in Python (12.42) and Java (7.59). Better performance in Java can be seen in our Java model, which achieves a Pass@10 rate of 8.23 despite being fine-tuned on a relatively smaller dataset of 100M Java tokens. The Ruby, Rust, and Swift models all have the best performances in their respective languages.

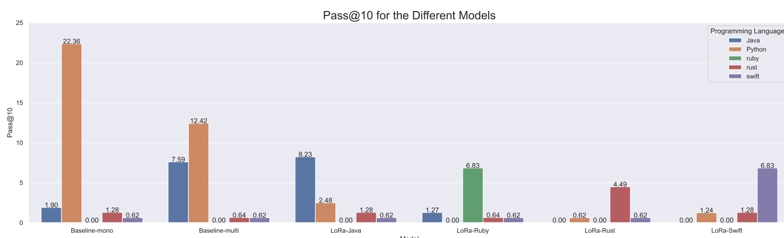

Figure 2: Pass@10 Rates by Fine-tuned models and Baselines

4 Conclusions

The journey towards democratizing the field of artificial intelligence and code language models is a broad mission that requires addressing challenges on various fronts. This work has presented a concerted effort to bridge the gap between advanced AI technologies and their practical usability, particularly in the domain of code intelligence. By focusing on accessibility, usability, and empirical understanding, we have contributed to the ongoing narrative of democratization in AI.

References

[1] Federico Cassano, John Gouwar, Daniel Nguyen, Sydney Nguyen, Luna Phipps-Costin, Donald Pinckney, Ming-Ho Yee, Yangtian Zi, Carolyn Jane Anderson, Molly Q Feldman, Arjun Guha, Michael Greenberg, and Abhinav Jangda. Multipl-e: A scalable and polyglot approach to benchmarking neural code generation. *IEEE Transactions on Software Engineering*, 49(7): 3675–3691, 2023. doi: 10.1109/TSE.2023.3267446.

[2] Denis Kocetkov, Raymond Li, Loubna Ben Allal, Jia Li, Chenghao Mou, Carlos Muñoz Ferrandis, Yacine Jernite, Margaret Mitchell, Sean Hughes, Thomas Wolf, Dzmitry Bahdanau, Leandro von Werra, and Harm de Vries. The stack: 3 tb of permissively licensed source code, 2022.

[3] Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. Codegen: An open large language model for code with multi-turn program synthesis, 2023.