Fine-tune LLM for Code Generation

Ayush Chamoli
RPTU Kaiserslautern, Department of Computer Science

Note: *This report contains a project documentation and reflection on the portfolio task submitted for the lecture Engineering with Generative AI in WiSe 2023-24. This report is an original work and will be scrutinised for plagiarism and potential LLM use.*

1 Portfolio documentation

This overview provides a high-level summary of the project's goals, objectives, and scope, setting the stage for the detailed documentation that follows.

1.1 Introduction

1.1.1 Overview of the Portfolio Project

The aim of this portfolio project is to explore the application of generative artificial intelligence for generating code by fine-tuning a large language model for code generation purpose specific to python programming. The project has multiple phases, which includes data collection, data generation and preprocessing, fine-tuning LLMs, and evaluation of their performances.

1.1.2 Objectives

There are a number of objectives in the portfolio project:

1. **Fine-Tuning LLMs**: The objective is to fine-tune a pre-trained LLM for generating python code.
2. **Synthetic Dataset Generation**: The objective is to generate synthetic dataset that is similar to the benchmark dataset using Llama model hosted on AWS.
3. **Performance Evaluation and Analysis**: The objective is to analyse performance of models pre-trained using different data.
4. **Visualization**: The objective is to visualize the result of each model and it's performance to facilitate a better understanding

1.1.3 Base Model

A 4-bit quantized version of Llama 2 7B Chat [1] is used as the base model for further finetuning. Llama 2 7B has been an open source state-of-the-art LLM for text generation task. The 4-bit quantization further helps in improving its efficiency by reducing the memory usage without affecting the performance, which makes it suitable for working on it in resource-constrained environment.

1.1.4 Benchmark Dataset

Most Basic Python Problems (MBPP) dataset [2] is selected as the benchmark dataset for the portfolio project. It consists of 1000 basic python problems where each task has a description, a coding solution and three automated test cases.

1.2 Preprocessing Dataset

In this section, the data preprocessing steps are described. The training data includes 374 tasks and testing data includes 500 tasks. This split is taken from MBPP dataset [2]. A training data looks like the following.

```
Write a function to find the maximum of nth column from the given tuple list.
def max_of_nth(test_list, N):
  res = max([sub[N] for sub in test_list])
  return (res)
['assert max_of_nth([(5, 6, 7), (1, 3, 5), (8, 9, 19)], 2) == 19',
'assert max_of_nth([(6, 7, 8), (2, 4, 6), (9, 10, 20)], 1) == 10',
'assert max_of_nth([(7, 8, 9), (3, 5, 7), (10, 11, 21)], 1) == 11']
```

Listing 1: *text code* and *test_list* attribute of 11th data point in training dataset of MBPP

This cannot be used directly with Llama 2 7B Chat model. It needs to be changed in the following format before Llama 2 can process it.

```
<s> [INST] <<SYS>>
System Prompt
<</SYS>>
User Prompt
[/INST] Model Answer </s>
```

Listing 2: Prompt format of Llama 2 7B Chat

Here, the *System Prompt* is instruction for Llama 2 which is predefined, and *User Prompt* is the prompt that user passes. So the training data and testing data is mapped using mapping functions to add a *prompt* attribute to each dataset respectively as followed.

```
<s>[INST] <<SYS>>
You are a python programming assistant that obeys the constraints and passes the
example test case.
You wrap the code answer without any comments between [PYTHON] and [/PYTHON] tags.
In case a test case is available, it is written inside [TEST] and [/TEST] tags.
<</SYS>>
sample['text']
[TEST]sample['test'][0][/TEST]
[/INST]
[PYTHON]
sample['code']
[/PYTHON]</s>
```

Listing 3: Prompt for a sample training data

```
<s>[INST] <<SYS>>
You are a python programming assistant that obeys the constraints and passes the
```

```
example test case.
You wrap the code answer without any comments between [PYTHON] and [/PYTHON] tags.
In case a test case is available, it is written inside [TEST] and [/TEST] tags.
<</SYS>>
sample['text']
[TEST]sample['test'][0][/TEST]
[/INST]
[PYTHON]
```

Listing 4: Prompt for a sample test data

The idea is to use $[TEST]$ and $[/TEST]$ tags for passing a sample test case and $[PYTHON]$ and $[/PYTHON]$ tags for passing the sample code. This also helps during inference time so as to look only within the tags to retrieve the relevant python code [3].

1.3 Finetuning LLMs

Before Model A: Llama 2 7B Chat is discussed, it's important to shed light on how the model was chosen. Initially, CodeLlama 7B Instruct was chosen. This LLM has already fine-tuned for general-purpose code generation. However, because of the difficulty of running it on resource-constrained environment despite quantization, the model was dropped. Next, Tinyllama [4] was chosen. It only has 1 billion parameters which made it easier to work with it on resource constrained devices. However, there was no significant improvement observed in terms of performance of those model on code generation despite fine tuning it. So finally, Llama 2 7B Chat is chosen as the Model A for the portfolio project.

1.3.1 Model A: Llama 2 7B Chat

Llama 2 7B Chat is chosen as model A and is loaded after quantization into 4-bit. It makes it possible to load the LLM on Google Colab (T4 GPU) environment. Then the following steps are taken.

Repeat for each item in test dataset:

1. *Set max limit of new tokens to* 500 *and current limit to* 50.
2. *Pass 'prompt' attribute of mapped test dataset through tokenizer of Llama* 2 *7B Chat*.
3. *Pass the tokenized prompt through the LLM with maximum new tokens to current limit and top p to* 0.9 *and decode the output. Measure this generation time.*
4. *If second occurrence of* [*PYTHON*] *and* [*/PYTHON*] *tags (the first occurrence is in the prompt itself) is found, go to step* 5, *else increase current limit by* 50, *set the 'prompt' to current decoded output and go to step* 2.
5. *Extract the code inside the second occurrence of* [*PYTHON*] *and* [*/PYTHON*] *tags.*
6. *Add all the items from 'test_list' attribute of current item of test dataset to the code.*
7. *Execute the code and update the statistics. Accuracy is measured as number of successfully executed code divided by total number of items in dataset.*

Listing 5: Calculation of a Model's accuracy on MBPP's test dataset

Apart from this, the checkpoint of result is saved for each item of the dataset so as not to recompute it in case of running the model after resetting the Google Colab session. The model

has an accuracy of 9.4%, a median generation time of 17.31 seconds to generate a response. If only the correct answers are considered, the median generation time is 15.79 seconds.

1.3.2 Model B: Llama 2 7B Chat fine-tuned with MBPP dataset

Model A is loaded with the same 4-bit quantization. Then a LoRA [5] configuration is set for the fine-tuning. Then a Supervised Fine Tuning trainer is set with Model A, the LoRA configuration and training dataset and the model is fine-tuned for 3 epochs. Once the training is complete, the fined tuned Model B is saved.

Model B is then loaded while testing using the same steps as Listing 5. The model has an accuracy of 13.8%, a median generation time of 19.1 seconds to generate a response. If only the correct answers are considered, the median generation time is 12.42 seconds.

1.3.3 Model C: Llama 2 7B Chat fine-tuned with Synthetic dataset

Model C is fine-tuned in the same way as model B is fine-tuned. However, model C is fine-tuned using a synthetic dataset which is generated with the use of AWS Llama using it's API. Firstly, one-shot prompting is used in order to generate 20 basic topics of python for which relevant questions can be generated. The following prompt is used for the same.

```
<s> [INST] Write me 20 beginner topics for python programming which I can test.
Each topic must be between [T] and [/T] tags [/INST]
1. [T]Variables[/T]
2. [T]
```

Listing 6: One-shot prompt for python topic generation using AWS Llama API

In the above instructions, $[T]Variables[/T]$ is passed so as to generate the output from the AWS Llama in a particular format. This leads to generation of 20 topics, where each topic is inside $[T]$ and $[/T]$ and is extracted. These topics are further saved locally as the API generated different response each time.

Next, for each topic, the minimum number of questions is set and two questions are made manually which will be further used for generating atleast that minimum number of questions.

```
synthetic_dataset['Functions'] = {
  "MINIMUM_NUMBER_OF_QUESTIONS" : 100,
  "QUESTIONS" : ["Write a Python program defining a function to add
  two numbers.", "Write a Python program defining a function to
  subtract two numbers"]
}
```

Listing 7: Minimum number of questions and two manual questions for topic 'Functions'

```
<s>[INST] Generate {synthetic_dataset["Variables"]["MINIMUM_NUMBER_OF_QUESTIONS"]}
python programming questions only on the topic of Variables. Each question must be
inside [QUESTION] and [/QUESTION] tags.[/INST]
1. [QUESTION]{synthetic_dataset["Variables"]["QUESTIONS"][0]}[/QUESTION]
2. [QUESTION]{synthetic_dataset["Variables"]["QUESTIONS"][1]}[/QUESTION]
```

```
3. [QUESTION]
```

Listing 8: Two-shot prompt for python's question generation on topic 'Variables' using AWS Llama API

A question is extracted when it is between $[QUESTION]$ and $[/QUESTION]$ tags. The minimum number of questions for a topic is chosen manually as well because it is easier to generate beginner questions for some topics compared to others. The AWS API is called until atleast that minimum number of questions is obtained for a topic. Finally, 1151 synthetic questions are generated which is more than 3 times the number of questions in training dataset.

Finally, for each question, a code is generated using AWS Llama API using the following prompt.

```
<s>[INST]Write a python code to solve the following coding problem that obeys the
constraints and passes the example test cases. Please wrap your code answer between
[PYTHON] and [/PYTHON] tags:
{synthetic_dataset["data"][90]['text']}[/INST]
```

Listing 9: Zero-shot prompt for python's code generation for 91st question in synthetic dataset using AWS Llama API

This generates python code which can be extracted from the text between the $[PYTHON]$ and $[/PYTHON]$. The AWS Llama API can be forced to start by generating the python code by adding $[PYTHON]$ at the end in the prompt in Listing 9. However, this was realized after codes for all the questions were generated. This could have sped up the process while reducing the number of maximum tokens needed for the same.

The code generated by AWS Llama API includes a number of comments, print statements and input statements. To make it similar to MBPP dataset, these parts of the code is cleaned. Apart from this, AWS Llama API is not used in order to generate test cases which are there in MBPP dataset because in case of using it for a chat assistant, it may be the case that a user has no specific test case. So this model is expected to have that edge.

Model C is fine-tuned in the same way as model B is fine-tuned with the difference of the dataset used for training. Model C uses the synthetically generated dataset for training.

Model C is then loaded while testing using the same steps as Listing 5. The model has an accuracy of 13.4%, a median generation time of 19.11 seconds to generate a response. If only the correct answers are considered, the median generation time is 7.21 seconds.

1.3.4 Model D: Llama 2 7B Chat fine-tuned with MBPP and Synthetic dataset combined and shuffled

For model D, the synthetic dataset and MBPP training dataset are mixed and shuffled and fine-tuned in the same way as model B using the mixed dataset.

Model D is then loaded while testing using the same steps as Listing 5. The model has an accuracy of 14.4%, a median generation time of 18.71 seconds to generate a response. If only the correct answers are considered, the median generation time is 7.0 seconds.

1.4 Results

As seen from this visualization, model D has the best performance among all the models being slightly better than model B and model C and a lot better than model A for code generation task.

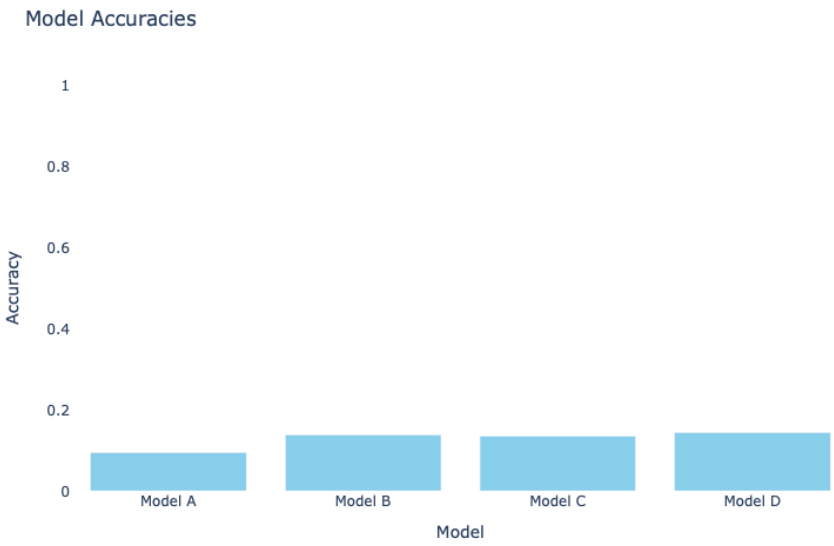

Figure 1: Model Accuracies

In the generation time, it is seen that model A has a lot of tasks which has outlier generation times. It can be concluded that model A failed to use $[PYTHON]$ and $[/PYTHON]$ tags properly in order to terminate the generation quickly. This explains a number of tasks which took more than 300 seconds to generate an output. This behaviour is seen less in fine-tuned models. The median generation time for all the models are close to each other.

In case only the correct responses are considered, it can be observed that the median generation time for model C and D are way better than model A and B.

This leads to the conclusion that model D is best for use case in terms of accuracy and generation times. It not only gives more accurate results than 4-bit quantized Llama 2 7B Chat, but also the time to generate a proper response is faster and thus it can further be used for development of an application that helps in Python Code generation

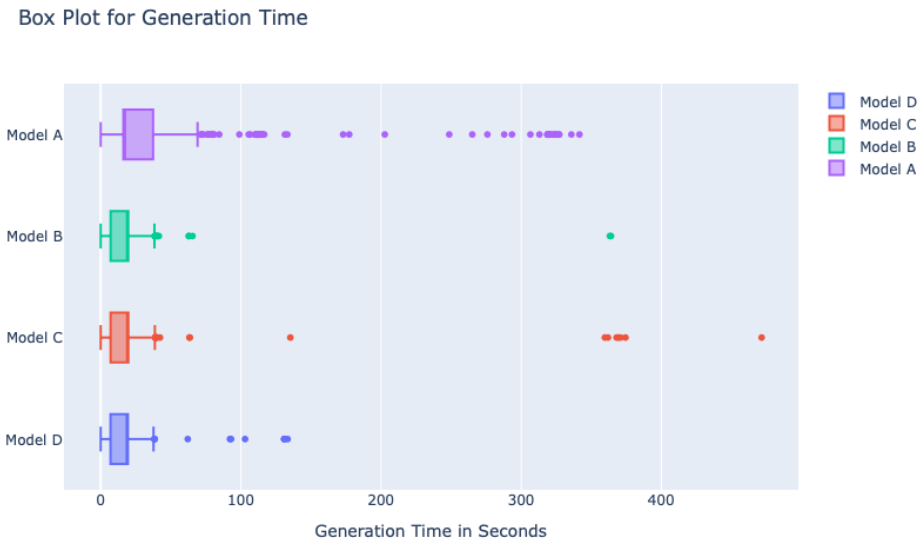

Figure 2: Box Plot for Generation Time of Different Models

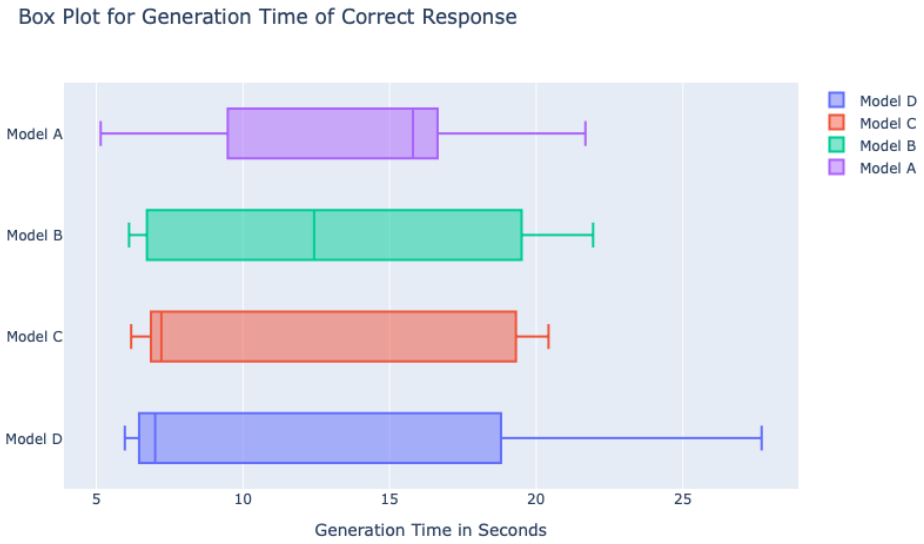

Figure 3: Box Plot for Generation Time of Correct Response of Different Models

2 Reflection

Working on the portfolio project on fine-tuning LLMs for code generation was a journey full of discovery, challenges and learning. In this section, I will write in details about the journey, challenges faced, adjustment made, ethical consideration and other topics.

2.1 Most interesting thing learned

Realizing how the nature of training data affects the fine-tuning process of LLMs for code generation was the most interesting thing I learned. I was surprised by how the model adapted to a particular style of writing code depending upon what kind of data was fed into it. For example, when synthetic data was used for fine-tuning model C without cleaning, the resulting code exhibited an abundance of comments, print statements and input statements. On the other hand, this was not the case for the fine-tuned model B as the training data lacked those elements. This make me emphasize the importance of quality of data for fine-tuning LLMs. Upon cleaning the synthetic data, not only model C performed better, but it also made it faster than model B. This suggests that the quality of the training dataset significantly influences the behaviour of LLMs while generating code.

The other interesting thing I learned involved the recent development of supervised fine-tuning techniques. With methods like LoRA, one can fine turn a LLM for their use-case without needing the exact same amount of hardware. This makes LLMs accessible to not just big companies but it can now also be used on personal projects.

2.2 Most Proud Part of Portfolio

The part of the portfolio project that I am the most proud of is the utilization of AWS Llama API for generating synthetic data. I was worried by the fact that it was required for the synthetic data to be 3 times as large as the training data and with only a limited number of API calls (3000 API call usage). With the use of AWS Llama API, I was able to develop synthetic data which was close to MBPP dataset. The approach for data generation involved changing the parameters and creating prompt that led in generation of more than 1000 unique python programming questions. Because of this, model C and model D had a lot more data to fine-tune with and it helped it achieve better generation time compared to the other two models.

2.3 Adjustment During Implementation Phase

The biggest adjustment made during the implementation phase were the change of initial base model A. First, I chose CodeLlama 7B Instruct as model A. However, I failed to get it working properly in my coding environment. Later, I tried to use TinyLlama 1B as model A. But the model did not really produce decent results after fine-tuning them and experimenting fine-tuning with different hyper parameters. So finally, I chose to use Llama 2 7B Chat as model A and had to make a number of changes in the existing code with TinyLlama to get things working and produce results which I was satisfied with.

Another adjustment made was with the text generation process of each of the model. Initially, the maximum token limit of 500 was being used in any case. However, this was not efficient as

I was only concerned with the code generated by the model which would be in the $[PYTHON]$ and $[/PYTHON]$ tags. There would be times that the model would output the code and follow it up with the explanation of the code before ending it generation itself. This would lead up to not only increase in generation time but also, a lot of text generated by the model was unused and the resources could better be used for generating code for the next question. I finally updated my approach for generation to be in steps where I would terminate generation as soon as the ending $[/PYTHON]$ tag is detected in a step. This led to a huge speed up in generation times.

If I had to do the portfolio project another time, I would have firstly explored the improvement of performance of different LLMs with just the training data of MBPP rather than making changes in the future. Also, while generating, I would make sure to have a mechanism in order to stop the generation early to save resources.

2.4 Ethical Considerations

When working with LLMs for code generation, it is really important to address ethical considerations. This can include bias, fairness and privacy. As the training data plays a huge role in influencing the behaviour of the model, the bias present in the training dataset can find it's way in the code generated by LLM. Apart from it, it is also really important to ensure data privacy when using external dataset for fine-tuning model. When generating synthetic dataset, I made questions that were not involving use of personal data and which were general python programming questions. However, if it were to involve the use of any personal data, I would have made sure to augment the data to account for it and remove any personal information.

2.5 Exciting Lecture Topics

Among the topics covered by all the lectures during the semester, the one about Parameter Efficient Fine Tuning (PEFT) with Low Rank Adaptations (LoRA) by Mariano Kamp of AWS was the most exciting one for me. One part of the reason was because the lecture was taught by an industry expert in the field. The other part of the reason was the usefulness of this topic on the overall portfolio project. I am intrigued by the advantages of LoRA for fine tuning as one only has to train a small number of new weights that are inserted into the model. This not only makes the fine-tuning process faster but it is also memory-efficient and the final model weights are way smaller making it easier to store.

2.6 Future Project Ideas

Based on the content of the lecture taught during the semester and the task I have carried out in this portfolio, I would be interested in developing an AI Chat assistant fine-tuned for agricultural purposes and which is also fluent in multiple languages spoken in India. This will empower the farmers across India as the AI assistant will help in optimizing agricultural practices, providing personalized recommendations and real-time assistance in order to maximize crop yields. The multilingual capability is the most important as it ensure accessibility to the Chat assistant despite the linguistic background. Therefore, the model will be fine-tuned to understand and

generate text not only in English but also in multiple languages spoken in India, such as Hindi, Marathi, Bengali and others.

2.7 Multiple Choice Questions

1. Which of the following options is an example of the environmental aspect of Large Language Models?
    a) Large Language Models are capable of renewable energy research through natural language processing.
    b) Large Language Models help in conservation of endangered species through predictive modeling.
    c) Large Language Models generate a huge amount of carbon emissions while training.
    d) Large Language Models are good assistant in agriculture by providing real time weather forecast.
2. For a student's learning and academic research, which of the following aspect is impacted by Large Language Models?
    a) Large Language Models are irrelevant for students as they are mainly relevant for industrial purpose.
    b) Large Language Models offer limited usefulness for students because of the complexity to understand topics a student learns.
    c) Large Language Models hinder student's development as it increases reliance of automated system and affects critical thinking and creativity.
    d) Large Language Models enhance student's learning by aiding in research and providing access to a large amount of educational resource.
3. How can a small company utilize Large Language Model in a cost-effective manner?
    a) Use cloud-based LLM services with pay-as-you-go pricing.
    b) Work together with a large company to share LLM resources and the costs related to it.
    c) Hire AI experts to deploy in-house LLM application.
    d) Invest in special hardware for LLM training and inference.

2.8 Conclusion

In conclusion, the portfolio project on fine-tuning Language Models for code generation has helped me in deepening my understanding of generative AI for text generation. Through experimentation, iteration and critical thinking, I have gained valuable insights into the challenges and opportunities available by using Large Language Models for real world applications. In future, I am excited to keep up with the trends and finding in generative AI as it will help not only in reshaping text generation tasks but also many other tasks.

References

[1] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023.

[2] Augustus Odena, Charles Sutton, David Martin Dohan, Ellen Jiang, Henryk Michalewski, Jacob Austin, Maarten Paul Bosma, Maxwell Nye, Michael Terry, and Quoc V. Le. Program synthesis with large language models. In *n/a*, page n/a, n/a, 2021. n/a.

[3] 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. Code llama: Open foundation models for code, 2024.

[4] Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. Tinyllama: An open-source small language model, 2024.

[5] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021.