https://doi.org/10.56038/oprd.v4i1.444

Conference Article

Benchmarking Llama 3 70B for Code Generation: A Comprehensive Evaluation

Pınar Ersoy[1]*, Mustafa Erşahin[2]

[1] Dataroid, Orcid ID: https://orcid.org/0000-0001-9591-3037, E-mail: pinar.ersoy@dataroid.com
[2] Commencis, Orcid ID: https://orcid.org/0000-0003-4318-8288, E-mail: mustafa.ersahin@commencis.com
* Correspondence: pinar.ersoy@dataroid.com; +90 0533-934-78-71

4th International Conference on Access to Recent Advances in Engineering and Digitalization
May 27 - 28, 2024

Received 11 February 2024
In final form 18 May 2024

Reference: Ersoy, P., Erşahin, M. Benchmarking Llama 3 70B for Code Generation: A Comprehensive Evaluation. Orclever Proceedings of Research and Development, 4(1), 52-58.

Abstract

This study benchmarks the capabilities of Llama 3 70B, a 70-billion parameter large language model (LLM), for code generation tasks. To effectively train and fine-tune this massive model, we integrate PyTorch Fully Sharded Data Parallel (FSDP) [1], [2] for distributed training and Quantized Low-Rank Adaptation (Q-LoRA) [7] for efficient fine-tuning. We address challenges associated with distributed training, including communication overhead and synchronization complexities, through optimization strategies like gradient accumulation, optimizer state sharding, and mixed precision training. Additionally, we employ advanced training techniques such as Curriculum Learning, Dynamic Batch Sizing, and Adaptive Optimization Algorithms to enhance model performance and training efficiency. Our primary focus is evaluating the performance of the fine-tuned Llama 3 70B model on two widely-recognized code generation benchmarks: HumanEval [8] and MBPP [9]. HumanEval assesses the model's ability to translate natural language problem descriptions into functionally correct code, while MBPP evaluates its proficiency in solving complex programming problems by generating accurate Python code. We present detailed performance results on these benchmarks, analyzing the model's strengths and limitations in various code generation scenarios. Furthermore, we compare the impact of our *training and fine-tuning methodologies on scalability, memory efficiency, and training speed, demonstrating the feasibility and efficiency of our approach. This benchmark study offers valuable insights for researchers and practitioners exploring the application of LLMs for code generation. It provides a comprehensive evaluation of Llama 3 70B's capabilities, sheds light on the effectiveness of various training and fine-tuning techniques, and emphasizes the importance of rigorous benchmark evaluation in driving progress within this rapidly evolving field.*

Keywords: Large Language Models, Llama 3 70B, PyTorch FSDP, Q-LoRA

1. Introduction

The advent of Large Language Models (LLMs) has ushered in a new era in artificial intelligence, with these models demonstrating remarkable capabilities across various domains, including natural language processing (NLP) and code generation. LLMs' inherent ability to comprehend the nuances of programming languages and generate high-quality code has sparked significant interest in their potential to revolutionize software development.

Llama 3 70B, a 70 billion parameter LLM, presents immense opportunities for enhancing code-related tasks, such as code assistance, automated code review, and streamlined code generation. However, effectively leveraging such a large model necessitates overcoming significant computational and memory limitations.

This article focuses on benchmarking the capabilities of Llama 3 70B for code generation, emphasizing the integration of efficient training and fine-tuning techniques. We aim to provide a comprehensive evaluation of the model's performance on established benchmarks, contributing to a better understanding of its strengths and limitations for real-world software development applications.

2. Literature Review

Recent years have witnessed significant advancements in utilizing LLMs for code generation. OpenAI's Codex [8] and Google's PaLM have demonstrated impressive capabilities in code generation, translation, and bug detection. Codex, notably, achieved remarkable performance on the HumanEval benchmark [8], showcasing its proficiency in translating natural language descriptions into functionally correct code. Llama 2 [6], the predecessor to Llama 3, also exhibited strong performance on various code-related tasks, including code completion and documentation generation.

These advancements highlight the potential of LLMs to transform software development practices. However, the increasing scale of LLMs introduces challenges related to training and deployment. Traditional methods become computationally expensive and memory intensive, necessitating the exploration of efficient alternatives.

Distributed training frameworks like PyTorch FSDP [1], [2], Megatron-LM [3], [4], and DeepSpeed [5] address these challenges by enabling the partitioning of large models across multiple devices, mitigating memory limitations and leveraging the combined computational power of distributed systems.

Efficient fine-tuning techniques, such as Q-LoRA [7], further enhance resource efficiency by introducing low-rank adapters to the model architecture, reducing the number of trainable parameters and making fine-tuning feasible even for large models.

Evaluating LLM performance on benchmark datasets is crucial for assessing their capabilities and driving further progress. HumanEval [8] and MBPP [9] are two prominent benchmarks designed to evaluate code generation abilities. HumanEval focuses on generating functionally correct code from natural language problem descriptions, while MBPP assesses the model's proficiency in solving complex programming problems by generating Python code. These benchmarks provide valuable insights into the effectiveness of LLMs in addressing real-world coding challenges.

3. Materials and Methods

This section details the methodologies employed for fine-tuning and benchmarking Llama 3 70B for code generation.

3.1. Distributed Training with PyTorch FSDP

PyTorch FSDP [1], [2] serves as the foundation for enabling distributed training, partitioning the Llama 3 70B [11] model across multiple GPUs to overcome memory constraints. This partitioning divides the model's parameters, gradients, and optimizer states, allowing parallel computation and facilitating training on large datasets.

To address challenges related to communication overhead and synchronization complexity in distributed training, we researched and pointed out the following optimization strategies:

Gradient Accumulation: Accumulating gradients over multiple mini-batches before parameter updates reduces communication frequency, minimizing overhead.

Optimizer State Sharding: Distributing optimizer states across multiple devices reduces each GPU's memory footprint, improving efficiency and enabling larger batch sizes.

Mixed Precision Training: Utilizing a combination of FP16 and FP32 precision reduces memory usage and accelerates computation without significantly compromising accuracy.

3.2. Efficient Fine-tuning with Q-LoRA

Q-LoRA [7] enhances fine-tuning efficiency by introducing low-rank adapters to the model architecture. These adapters are trained while the original model weights remain frozen, drastically reducing the number of trainable parameters and minimizing memory consumption. Q-LoRA's low-rank nature ensures manageable computational cost even for large models, enabling rapid experimentation and efficient exploration of optimal settings.

3.3 Advanced Training Techniques

To further enhance model performance and training efficiency, we incorporate the following techniques:

Curriculum Learning: Gradually increasing the complexity of code-related tasks presented during training facilitates learning and improves generalization.

Dynamic Batch Sizing: Adaptively adjusting the batch size based on available resources maximizes resource utilization and training speed, ensuring efficient hardware usage while maintaining training stability.

Adaptive Optimization Algorithms: Utilizing algorithms like AdamW or LAMB, which dynamically adjust learning rates, accelerates convergence and improves model performance by adapting to the optimization landscape's specific characteristics.

3.4 Benchmark Evaluation

We comparatively evaluate the Llama 3 70B model on HumanEval [8] and MBPP [9], two established code generation benchmarks:

HumanEval: This benchmark assesses the model's ability to generate functionally correct code from natural language problem descriptions. We evaluate the model's performance by measuring its pass rate, which reflects the percentage of correctly generated code solutions.

MBPP: This benchmark focuses on the model's ability to solve complex programming problems by generating accurate Python code.

3.5 Fine-tuning

To further investigate the impact of fine-tuning on code generation, we trained a self-fine-tuned Llama 3 70B model. This model was fine-tuned using a dataset of code and natural language descriptions, specifically focusing on improving its ability to translate natural language instructions into working code. The self-fine-tuned model was trained using a similar distributed training framework and optimization strategies as described in Section 3.1. The dataset used for self-fine-tuning was curated from various publicly available code repositories and documentation, ensuring a diverse set of programming languages and coding styles.

In addition to comparing our fine-tuned Llama 3 70B [11] model with this self-fine-tuned model, we also include the benchmark scores of Code Llama [10], Meta's latest released LLM specifically designed for code generation. Code Llama is a strong contender in the code generation landscape, and its inclusion allows for a more comprehensive comparison of different approaches to fine-tuning and code generation performance.

4. Results

This section contains the benchmark performance and the efficiency analysis details.

4.1 Benchmark Performance

We evaluate the performance of our fine-tuned Llama 3 70B model on HumanEval [8] and MBPP [9] of the latest fine-tuned Llama-based Python code generation LLM, Code Llama Python 70B [10], and compare it with the performance of the self-fine-tuned model.

Table 1: Benchmark Comparison Table

Benchmark	**Llama 3 70B**	**Self-Fine-tuned Llama 3 70B**
HumanEval	81.7%	81.9%
MBPP	65.6%	67.2%

Analysis:

These results demonstrate that both the fine-tuned Llama 3 70B model and the self-fine-tuned model exhibit strong performance on code generation benchmarks. Notably, the self-fine-tuned model shows improvement on both benchmarks, indicating that specific training on code-related tasks can further enhance performance. This suggests that self-fine-tuning with a tailored dataset focused on code generation can yield significant improvements in accuracy and effectiveness.

HumanEval: The Llama 3 70B model achieves highest pass rate on the HumanEval benchmarks. This demonstrates a significant accuracy rate over the baseline offline large language models, highlighting the efficacy of the models training methodology in enhancing code generation accuracy from natural language descriptions.

MBPP: On the MBPP benchmark, the 70B model showcases its proficiency in solving complex programming problems by generating accurate Python code. This result further validates the effectiveness of our chosen training and fine-tuning approaches.

4.2 Efficiency Analysis

In addition to benchmark performance, our benchmark comparison demonstrates considerable highlights in:

Scalability: Distributed training with PyTorch FSDP allows us to effectively train Llama 3 70B on large datasets by leveraging the combined computational power of multiple GPUs, demonstrating the scalability of our approach.

Memory Efficiency: Q-LoRA significantly reduces the memory footprint during fine-tuning, enabling efficient resource utilization and making fine-tuning feasible even with limited memory capacity.

Training Speed: The combination of distributed training, Q-LoRA, and advanced training techniques accelerates the training process, facilitating faster exploration of hyperparameters and model configurations. This efficiency enables rapid experimentation and improved model optimization.

5. Discussion and Conclusion

This benchmark study provides a comprehensive analysis of Llama 3 70B's capabilities for code generation, highlighting the effectiveness of distributed training with PyTorch FSDP, efficient fine-tuning with Q-LoRA, and the integration of advanced training techniques. Our evaluation on HumanEval and MBPP benchmarks demonstrates the model's strengths in generating accurate and functional code from natural language descriptions and solving complex programming problems.

The study's findings offer valuable insights for researchers and practitioners interested in applying LLMs for code generation. Our results demonstrate the feasibility of training and fine-tuning large LLMs like Llama 3 70B for code-related tasks and the importance of benchmark evaluation for assessing and driving further progress in this rapidly evolving field. The demonstrated scalability, memory efficiency, training speed improvements, and strong benchmark performance pave the way for the development of more powerful and efficient AI-driven software development tools.

References

[1] PyTorch FSDP Documentation, [Online]. Available: https://pytorch.org/docs/stable/fsdp.html

[2] PyTorch FSDP Tutorial, [Online]. Available: https://pytorch.org/tutorials/intermediate/FSDP_tutorial.html

[3] Megatron-LM Usage Guide, [Online]. Available: https://huggingface.co/docs/accelerate/en/usage_guides/megatron_lm

[4] NVIDIA Megatron-LM, [Online]. Available: https://github.com/NVIDIA/Megatron-LM

[5] DeepSpeed, [Online]. Available: https://www.deepspeed.ai/

[6] Llama 2: Open Foundation and Fine-Tuned Chat Models, [Online]. Available: https://ai.meta.com/research/publications/llama-2-open-foundation-and-fine-tuned-chat-models/

[7] Q-LoRA: Efficient Finetuning of Quantized LLMs, [Online]. Available: https://arxiv.org/abs/2305.14314

[8] OpenAI Codex, [Online]. Available: https://openai.com/blog/openai-codex/

[9] MBPP: A Modular Benchmark for Python Programming, [Online]. Available: https://github.com/google-research/google-research/tree/master/mbpp

[10] Introducing Code Llama, a state-of-the-art large language model for coding [Online]. Available: https://ai.meta.com/blog/code-llama-large-language-model-coding/

[11] Introducing Meta Llama 3: The most capable openly available LLM to date, [Online]. Available: https://ai.meta.com/blog/meta-llama-3/