Distributed LLM Benchmarking

Thomas Knickerbocker, Owen Ratgen

Topic/Title:

Deployment and Metric Visualizations of an LLM Model in a Distributed Environment Using Kubernetes, Docker, Karpenter, and Grafana

Abstract:

With the recent surge of Large Language Model (LLM) from models like ChatGPT, Gemini, etc…, it is increasingly important to make these models widely available to anyone at any given time. This project's purpose is to expose us to deploying a heavy duty application, and have the performance of the application stay consistently high no matter how many users are using it at any given moment [Top Large Language Models].

Background & Related Works:

Large Language Models (LLMs) such as GPT and BERT have grown in importance in numerous AI applications due to their ability to understand and generate human-like text responses [BERT]. These models require significant computational resources, making efficient deployment crucial, especially when dealing with distributed environments. Containerization technologies like Docker have simplified application deployment by isolating dependencies into light and reusable containers [Docker]. Meanwhile, containerization platforms like Kubernetes manage these containers across clusters for scalability and resilience [Kubernetes]. This combination allows systems to achieve low-latency and high-availability, crucial for real-time services [Mastering Kubernetes]. Software like Karpenter can be used to efficiently scale kubernetes nodes up or down [Karpenter].

With respect to data visualization for LLMs and other data-intensive applications, data visualization that integrates with Docker containers like Prometheus can be leveraged to provide admins time-series information [Prometheus].

In addition, the scaling of LLMs in distributed environments remains an ongoing area of research. Federated learning can be used to distributedly train machine-learning models, and can be adapted to LLMs with packages like FederatedScope-LLM [FederatedScopeLLM]. Faster hardware such as GPUs may also be utilized to further optimize performance in production [Optimizing ML Models for Production]. To monitor systems, tools like Prometheus and Grafana have been widely adopted to collect and visualize key performance metrics such as latency and throughput, and display them in an easy to comprehend manner [Grafana].

Motivation:

For any heavy duty application, it is essential for it to have low-latency in order to make sure a user doesn't lose any interest while accessing the application. The technologies we are going to use will help us utilize cloud resource management libraries [Docker and Kubernetes] in order to mitigate server outages, autoscale the number of worker nodes [Getting Started with Karpenter; Scaling Large Language Models] in order to follow through on the promise of low latency and high performance within our application, metric visualization [Grafana] in order to visualize the process of how our load management is working, and data management [Setting up DynamoDB] in order to keep user data secure and also retrain the model based on the inputs.

Problem Statement:

Our LLM needs to be distributedly hosted in order to provide users a low-latency experience, and we must be able to visualize our performance metrics in order to understand how to improve UX.

Our Approach:

1. Environment Setup: Create a distributed environment using Kubernetes and Docker
2. Apply Autoscaling: Use Karpenter so the application is able to automatically allocate more resources for load management and load balancing when the demand becomes to high
3. Model Selection: Find a suitable LLM for our project [Kubernetes for Machine Learning; Models - HuggingFace]
4. LLM Parameter Variation: Analyze different parameters for the LLM along with resource allocation to see which model will give us the best performance on output and low latency [Intro to Distributed LLM Training]
5. Integrate Data Management: Setup a DynamoDB through AWS to ensure that our data stays somewhere safe and easily accessible
6. Test Environment Setup: Create an environment for the LLM application that simulates realistic workload scenarios
7. Analyze Performance Metrics: Keep constant data for the latency, throughput, and scalability on our application using Prometheus and Grafana to see if we need to reassess anything [Performance Optimization of Machine Learning Models in Docker Containers; Challenges and AI and LLMs in Cloud Computing: Challenges and Opportunities. Advances in Computer Sciences]

Timeline:

- Week 1-2: Literature Review, become familiar with numerous LLM models, setup Kubernetes and Docker Environments
- Week 3-4: Apply autoscaling, select an LLM model, and play around with the model's parameters
- Week 5-6: Integrate Data Management System, get the test environment working, and setup performance metrics visualization
- Week 7: Analyze performance metrics and figure out what needs to be done to ensure our goals for the project
- Week 8: Work on final report

References:

1. Top Large Language Models:
https://vectara.com/blog/top-large-language-models-llms-gpt-4-llama-gato-bloom-and-when-to-choose-one-over-the-other/

2. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding: https://arxiv.org/abs/1810.04805
3. Docker Official Documentation: https://www.docker.com/
4. Kubernetes Official Documentation: https://kubernetes.io/docs/home/
5. Mastering Kubernetes: https://overcast.blog/mastering-kubernetes-for-machine-learning-ml-ai-in-2024-26f0cb509d81
6. Getting Started with Karpenter: https://karpenter.sh/docs/getting-started/getting-started-with-karpenter/
7. Prometheus: Up & Running: https://prometheus.io/docs/introduction/overview/
8. Federated Scope LLM: Federated Learning for LLMs. doi: https://doi.org/10.1145/3637528.3671573
9. Optimizing ML Models for Production: https://medium.com/@rahulholla1/optimizing-machine-learning-models-for-production-7b8f2cd4314f
10. Grafana: https://grafana.com/
11. Scaling Large Language Models: https://arxiv.org/abs/2005.14165
12. Setting up DynamoDB: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/SettingUp.html
13. Kubernetes for Machine Learning: https://www.oreilly.com/library/view/kubernetes-for-machine/9781492041501/
14. Models - HuggingFace: https://huggingface.co/models
15. Intro to Distributed LLM Training: https://gradient.ai/blog/distributed-llm-training-part-1 OR https://www.appypie.com/blog/llm-training-with-distributed-systems
16. McAuley, D. (2024). AI and LLMs in Cloud Computing: Challenges and Opportunities. Advances in Computer Sciences. https://academicpinnacle.com/index.php/acs/article/view/217
17. Performance Optimization of Machine Learning Models in Docker Containers: https://medium.com/ml-optimization