Infrastructure and Deployment Tuning Strategies & Chunking Methods

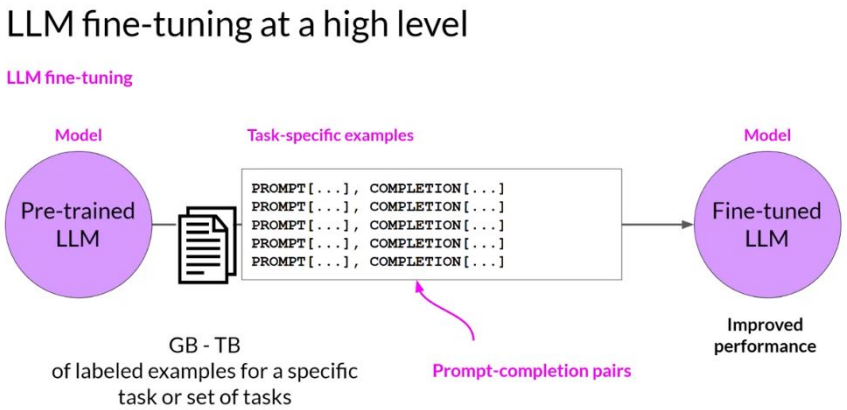
Hamza Salem - Innopolis university

Agenda

- Key strategies for optimal deployment
- Inference Acceleration Tools
- Tools and technologies for speeding up model inference
- Monitoring and Observability
- Chunking

Tuning

Fine-tuning large language models (LLMs) is a crucial process in adapting these models for specific tasks and improving their performance.



Tuning strategies

1. **Instruction Fine-Tuning:** Instruction fine-tuning involves training the model on datasets that include pairs of instructions and expected outputs. This method helps the model understand how to respond to specific prompts, enhancing its ability to follow human instructions effectively. For instance, if the goal is to improve summarization, the training data would consist of **instructions like "summarize this text" followed by the corresponding summaries**.

2. **Full Fine-Tuning**: This method updates all the model's weights during training, resulting in a new version of the model tailored to the specific dataset. **Full fine-tuning requires significant computational resources and memory, as it processes all gradients and optimizers**.

3. **Transfer Learning**: Fine-tuning is a type of transfer learning where a pre-trained model is further trained on a new dataset. **This allows the model to adjust its weights to better fit the new task while retaining the knowledge gained from the original training**.

Best Practices for Fine-Tuning

To ensure successful fine-tuning, several best practices should be followed:

Data Quality and Quantity: The dataset used for fine-tuning must be clean, relevant, and sufficiently large to avoid issues like overfitting.

Hyperparameter Tuning: Experimenting with different settings for learning rates, batch sizes, and training epochs is crucial for optimizing model performance and preventing overfitting or underfitting.

Regular Evaluation: Continuous assessment of the model's performance during training helps identify necessary adjustments and ensures that the model generalizes well to new data.

Key strategies for optimal deployment

Deploying LLMs effectively requires careful planning and consideration of various factors. Here are some key strategies to ensure optimal deployment:

1. **Model Selection**: Choose models wisely based on cost efficiency, such as using mixtural models that combine different LLMs for different parts of a query.

Implement cascading models that use a series of models, each trained on a different task, to optimize costs.

Set probability thresholds for model output to avoid unnecessary queries to higher-cost models.

Key strategies for optimal deployment

2. **Input Management**

Strategically truncate input sequences to reduce memory usage, as longer inputs demand more tokens.

Utilize models designed for cost-efficiency like the Router, which can route tasks to the most suitable and affordable model based on the query.

Key strategies for optimal deployment

3. **Memory Optimization**

Employ model pruning to remove unimportant weights from the network and reduce model size.

Leverage model quantization to decrease parameter precision and further reduce memory requirements.

Apply knowledge distillation to transfer knowledge from a large model to a smaller one, optimizing for size and performance

Key strategies for optimal deployment

4. **Deployment Strategies**

Consider cloud-based or on-premise deployment based on factors like data security and scalability requirements.

Choose appropriate hardware (GPUs, TPUs, CPUs) based on latency needs and cost constraints

The difference between CPU, GPU, and TPU is that the CPU is the brain of any computer system and is responsible for performing and managing the computer's logical, arithmetic, and I/O operations. On the other hand, a GPU is an additional processor capable of performing complex mathematical operations and advanced graphics rendering. Lastly, TPUs are Google's custom-made powerful processors built specifically to work on its projects, i.e, TensorFlow. Of the three processor units, TPU is the most recent processor built to train and learn machine learning models faster.

Key strategies for optimal deployment

5. **Continuous Monitoring and Optimization**

Experiment with different approaches to find the optimal cost-performance balance for your specific use case.

Stay updated on the evolving LLM landscape and new cost-effective models and optimization techniques.

Ensure ethical considerations like data privacy and responsible use remain top priorities alongside cost optimization.

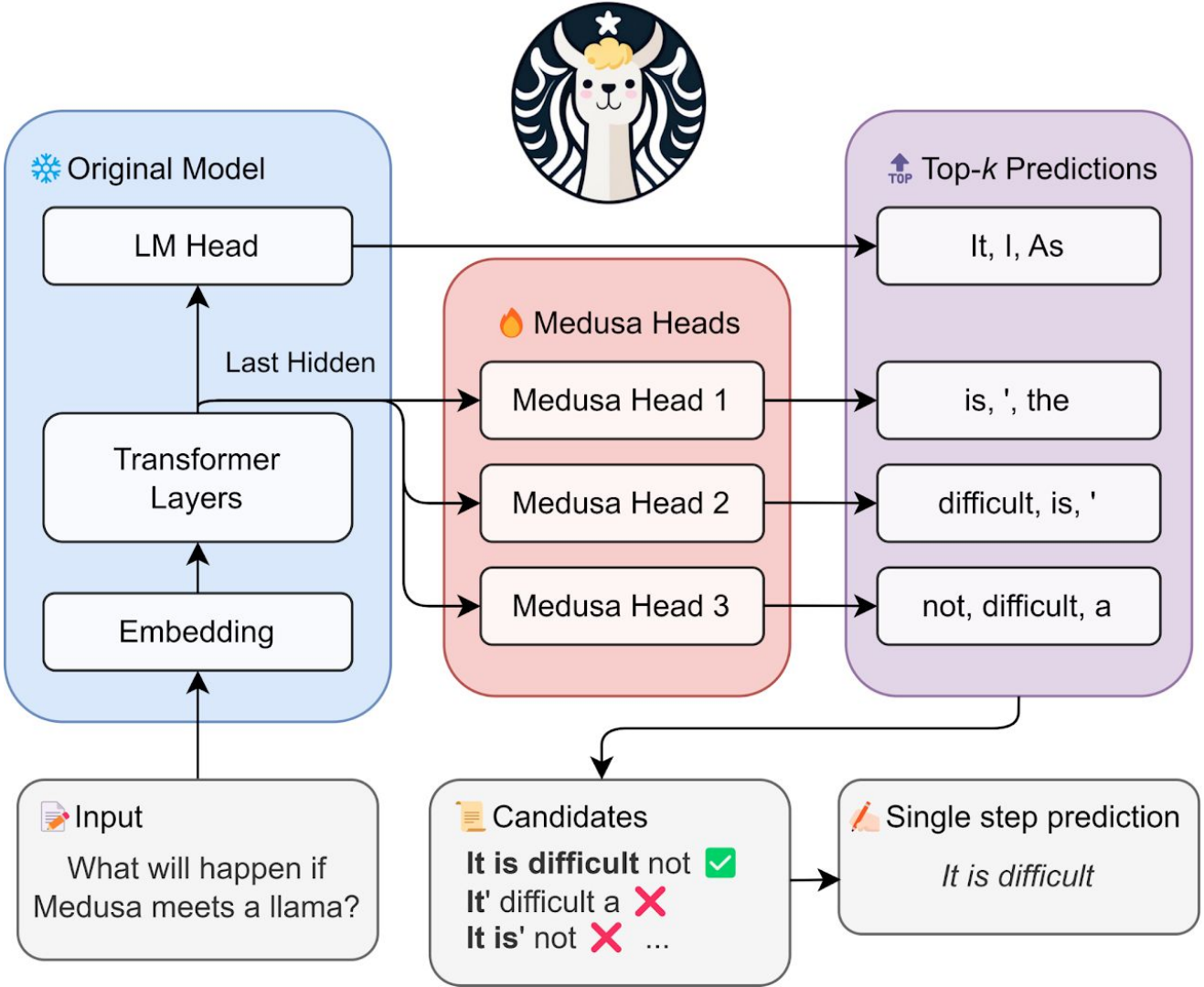
Inference Acceleration Tools for Large Language Models (LLMs)

1. Medusa: Augments LLM inference by adding extra decoding heads to predict multiple subsequent tokens in parallel.

Avoids the challenges of maintaining a separate draft model required by speculative decoding

Provides a simple and effective way to speed up LLM inference

https://github.com/FasterDecoding/Medusa


Original Model
LM Head
Last Hidden
Transformer Layers
Embedding
Medusa Heads
Medusa Head 1
Medusa Head 2
Medusa Head 3
Top-k Predictions
It, I, As
is, ', the
difficult, is, '
not, difficult, a
Input
What will happen if Medusa meets a llama?
Candidates
It is difficult not
It' difficult a
It is' not ...
Single step prediction
It is difficult

Inference Acceleration Tools for Large Language Models (LLMs)

2. Lookahead: A generic framework that introduces a multi-branch strategy to accept several tokens per forward step.

Uses a Trie-based retrieval and verification mechanism to guarantee correct output without approximation

Achieves significant inference speedups of 2.66x to 6.26x in real-world deployment

https://github.com/hao-ai-lab/LookaheadDecoding

Key Components of LLM Monitoring and Observability

Three Pillars of Observability

- Logs: These capture detailed events and actions within the system, providing a chronological record for troubleshooting.
- Metrics: Quantitative measurements such as latency, throughput, and error rates that help monitor the overall health and performance of the LLM.
- Traces: These track execution paths through various system components, aiding in identifying the causes of performance issues and understanding interactions between different parts of the system

Key Components of LLM Monitoring and Observability

Benefits of LLM Observability

- Improved Performance: Continuous monitoring allows for real-time assessment of performance metrics, enabling quick identification of deviations and enhancing overall application performance.
- Faster Issue Diagnosis: End-to-end visibility helps in pinpointing the root causes of issues, significantly reducing downtime and improving response times to problems.
- Better Explainability: Enhanced insights into model operations promote transparency, allowing stakeholders to understand and trust the decisions made by LLMs.
- Increased Security: Monitoring can detect anomalies and potential security vulnerabilities, helping to safeguard sensitive data and maintain application integrity.
- Efficient Cost Management: Observing resource consumption enables organizations to optimize resource allocation based on actual usage patterns, leading to cost savings.

Chunking/Text Splitters

What is chunking?

Splitting text to small chunk that can be input to LLMs.

Why?

- Reduce token usage
- Give system accurate text as input prompt
- ??

Chunking theories/Methods

How to split ?

- By characters?

Chunk size?

- Fixed ?
- dynamic?

https://js.langchain.com/v0.1/docs/modules/data_connection/document_transformers/

How to chunk?

- Book:
  - By page ?
  - By topic?
  - By paragraph?
  - By sentence?
- Web page:
  - By Code/Tag
  - By context?
- Conversion/chat?
  - By message?
  - By Time?

Summary

Must to have in your project:

- Retrieval Augmented generation (Vector database)(RAG)
- Chunking..

What you can build in your project other than Chunking?

- Memory?
- Adding parallel draft models?
- Router?

Exercise 1

Create Flask app/API that have :

- Endpoint for Text Chunking
  - Path: /chunking
  - Method: POST
  - Input: A string of text.
  - Output: An array of strings, where each string represents a context chunk derived from the input text.
- Endpoint for Question Answering
  - Path: /answer
  - Method: POST
  - Input: A string representing a question and an array of context chunks (strings).
  - Output: A string that is the response to the question based on the most relevant context chunk retrieved from the array.

You can find the input text in the notes of this slide

Submit CURL example to @enghamzasalem before Second lecture finish.

Exercise 2

Create an API endpoint that splits or chunks text using semantic chunking based on contextual similarity. After the initial chunking, the code should analyze the chunks, grouping them where necessary and providing a suggested threshold for each group based on the similarity gaps between adjacent chunks.

Resources

- https://www.perplexity.ai/search/introduction-to-tuning-strateg-SfSklpvIRJuBSka0lkLgOA
- https://js.langchain.com/v0.1/docs/modules/data_connection/document_transformers/
- https://github.com/hao-ai-lab/LookaheadDecoding
- https://github.com/FasterDecoding/Medusa
- https://www.perplexity.ai/search/lookahead-a-generic-framework-v8jq.gmzREyfNM7ZIema5A