Large Language Models Inference optimizations

Sid Raskar
Data Scientist
Pacific Northwest National Laboratory

Contributors : Krishna Teja Chitty-Venkata, Murali Emani

Presentation Agenda

LLM Inference Motivation

- How LLM Inference is becoming more important and more computationally demanding workload

Optimization Techniques

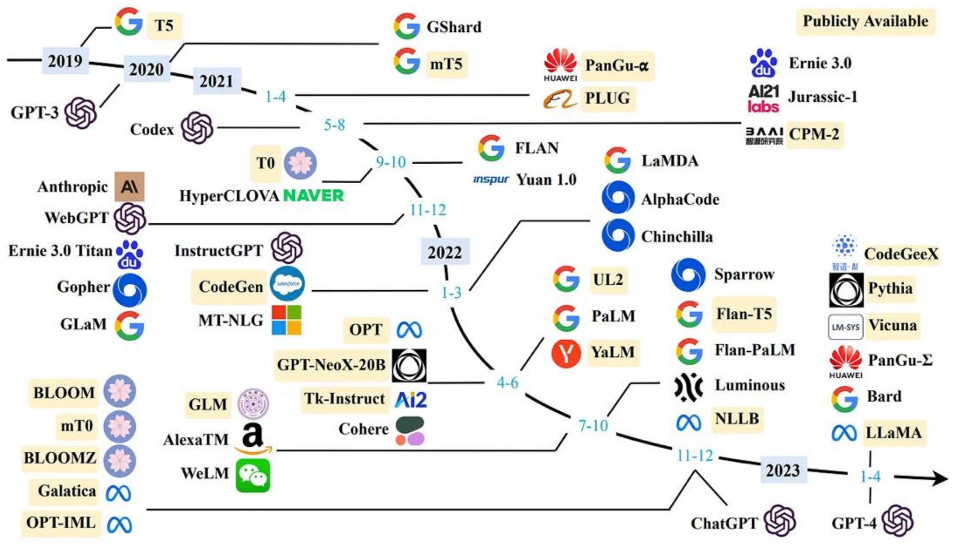
- Transformer Model Optimization Strategies
  - GQA, MoE
- LLM Algorithm Optimization Methods
  - Pruning, Quantization, KV Cache, Speculative Decoding
- Memory management and Hardware Parallelism Methods
  - PaggedAttention, Tensor, Pipeline and Expert Parallelism

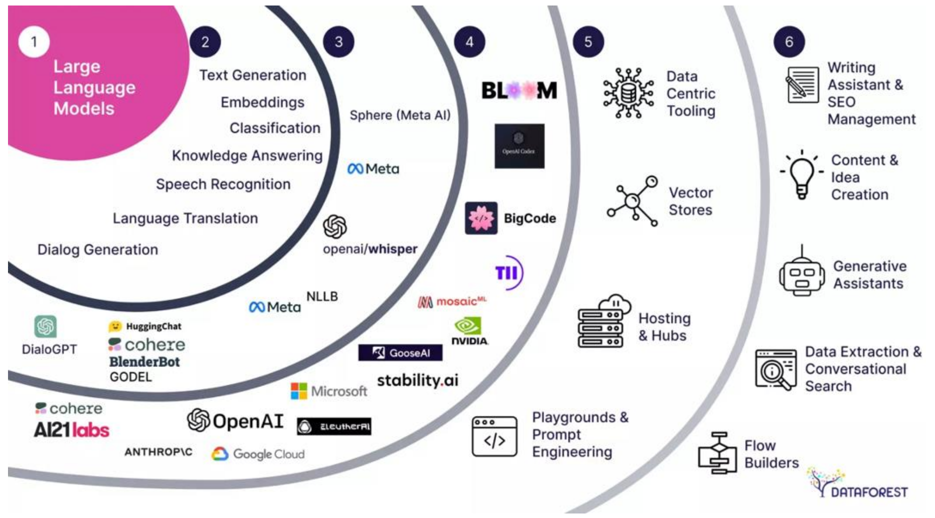
Hands on Agenda

- Run a sample LLaMA-3-8B using HuggingFace and vLLM Inference

Large Language Models (LLMs)

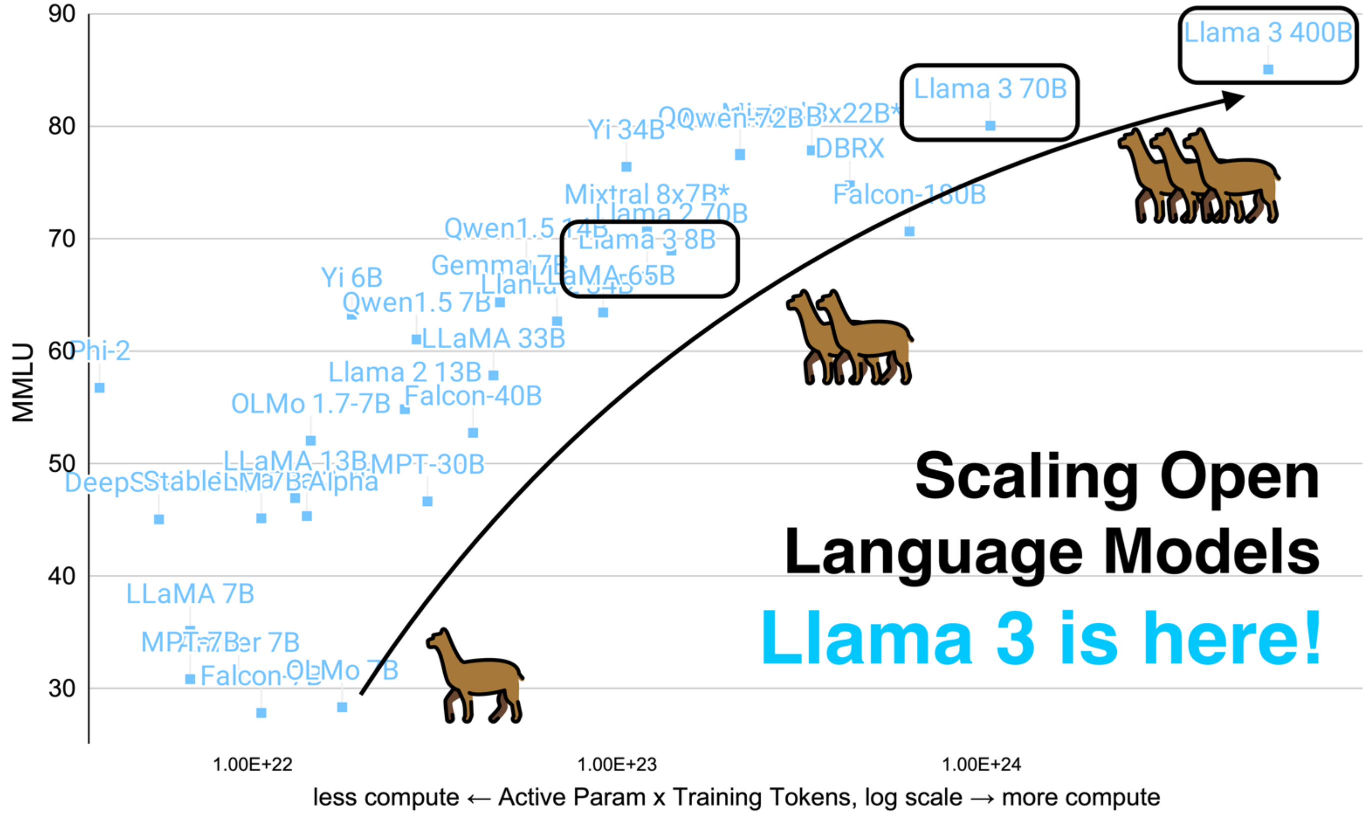
- LLM is a deep learning algorithm that's equipped to summarize, translate, predict, and generate human-sounding text to convey ideas and concepts.
- They leverage vast amounts of data and sophisticated algorithms to perform a wide range of tasks
- They rely on a massive number of parameters, which allows them to capture intricate language patterns and context.
- Examples of popular LLMs include OpenAI's GPT, Google's BERT, and Meta's LLaMA.





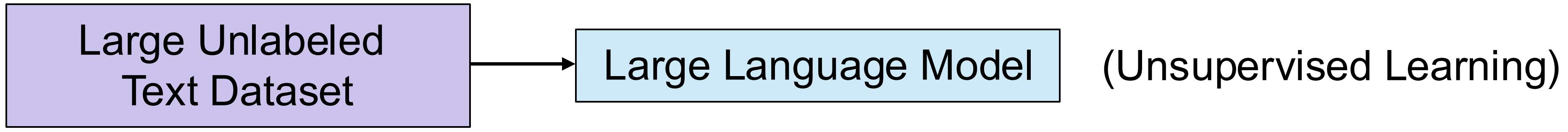
Need for LLM Inference Optimizations

- LLMs, with billions of parameters, can be slow during inference due to the computational load required to process large amounts of data.
- Many applications require real time responses from LLMs, which can be challenging.
- The high computational demands of LLMs translate into significant operational costs.
- LLM Inference optimization methods reduces energy consumption and hardware requirements, making deployments more cost-effective

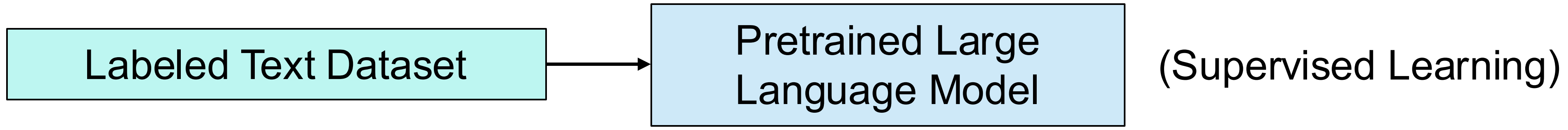


https://www.interconnects.ai/p/llama-3-and-scaling-open-llms

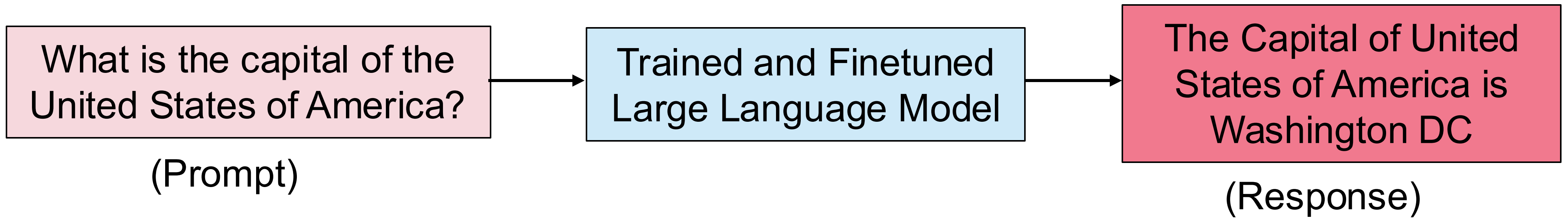
LLM Pretraining



LLM Finetuning

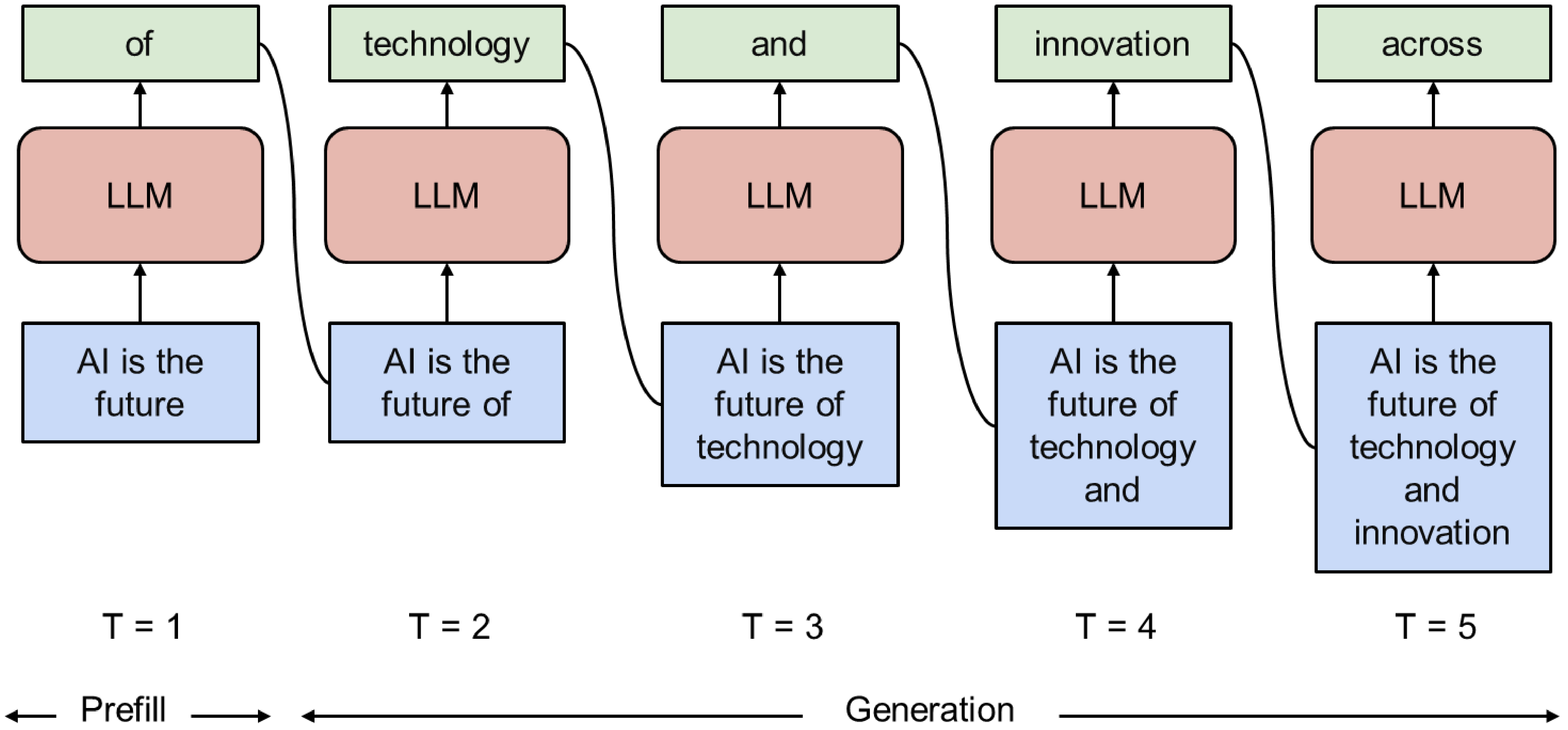


LLM Inference


What is the capital of the United States of America?
(Prompt)
Trained and Finetuned Large Language Model
The Capital of United States of America is Washington DC
(Response)

ATPESC10 ECP EXASCALE COMPUTING PROJECT Argonne NATIONAL LABORATORY

LLM Inference 101

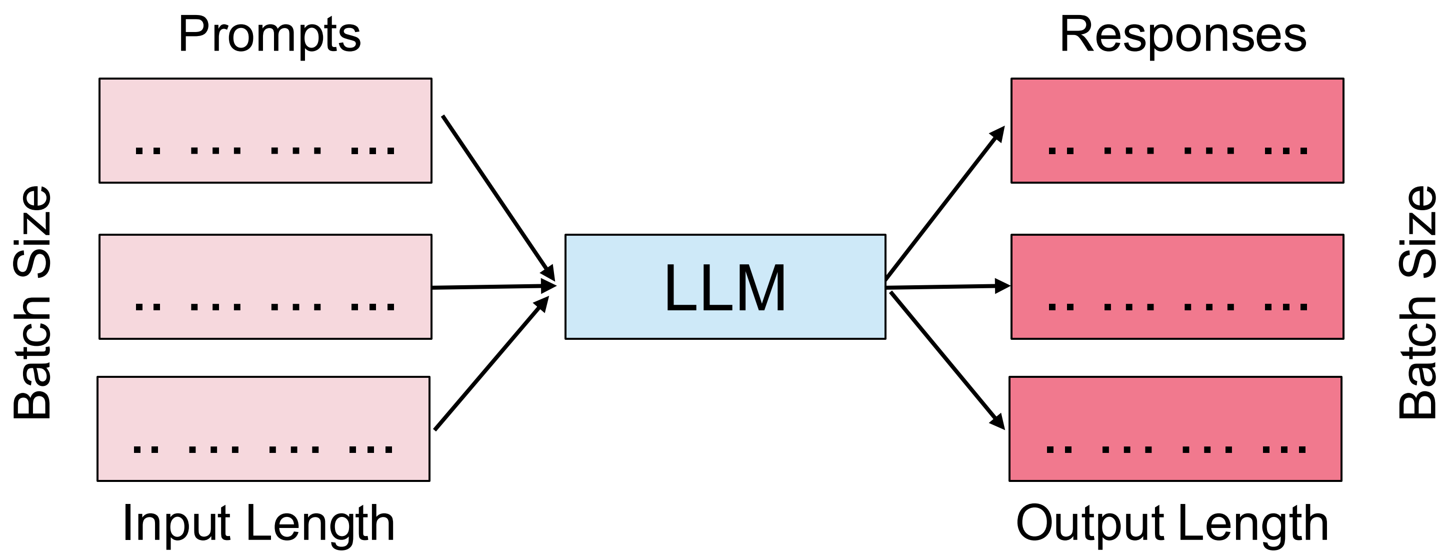


LLM Inference – Basic Terms

Input Length: Input Length refers to the total number of tokens given to an LLM as input prompt for a single query.

Output Length: Output Size, also referred to as maximum new tokens is the number of tokens produced by the model as a response to a single input prompt.

Batch Size: Batch Size refers to the number of input sequences processed and new output sequences produced simultaneously.



Performance Metrics

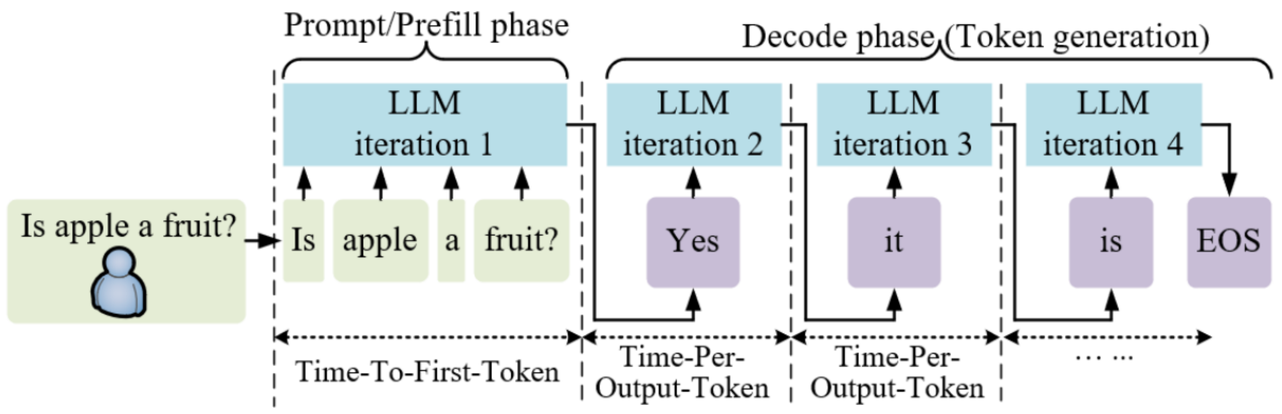
Perplexity quantifies the model's level of surprise when encountering new data to generate a new token. A lower perplexity indicates better performance

Throughput as the total number of tokens (both input and output) processed by the hardware per second.

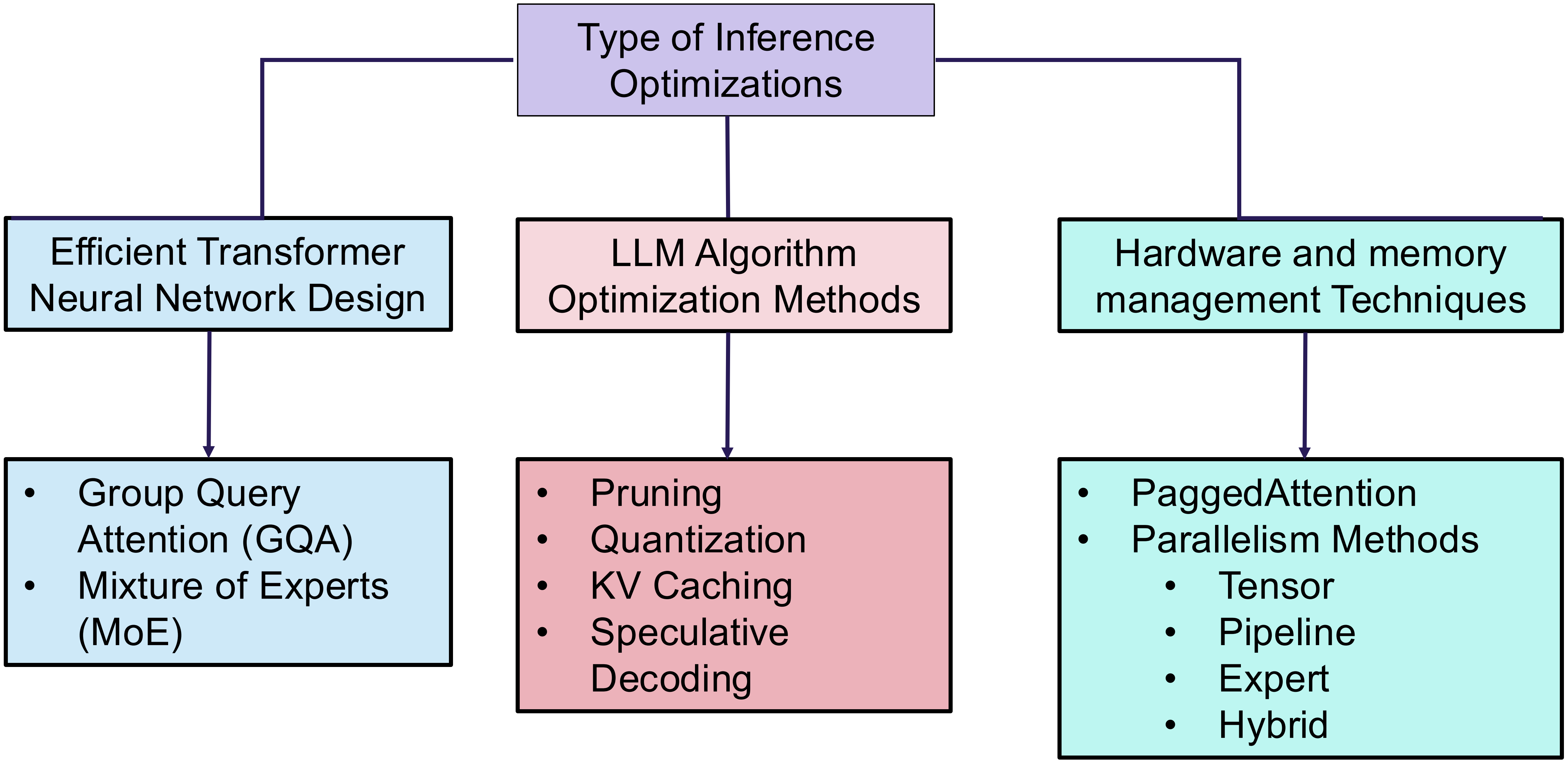
$$\text{throughput} = \frac{\text{Batch Size} \times (\text{Input} + \text{Output Tokens})}{\text{End-to-End Latency}}$$

Time to First Token (TTFT) is the amount of time required to produce the first output token after receiving an input prompt.

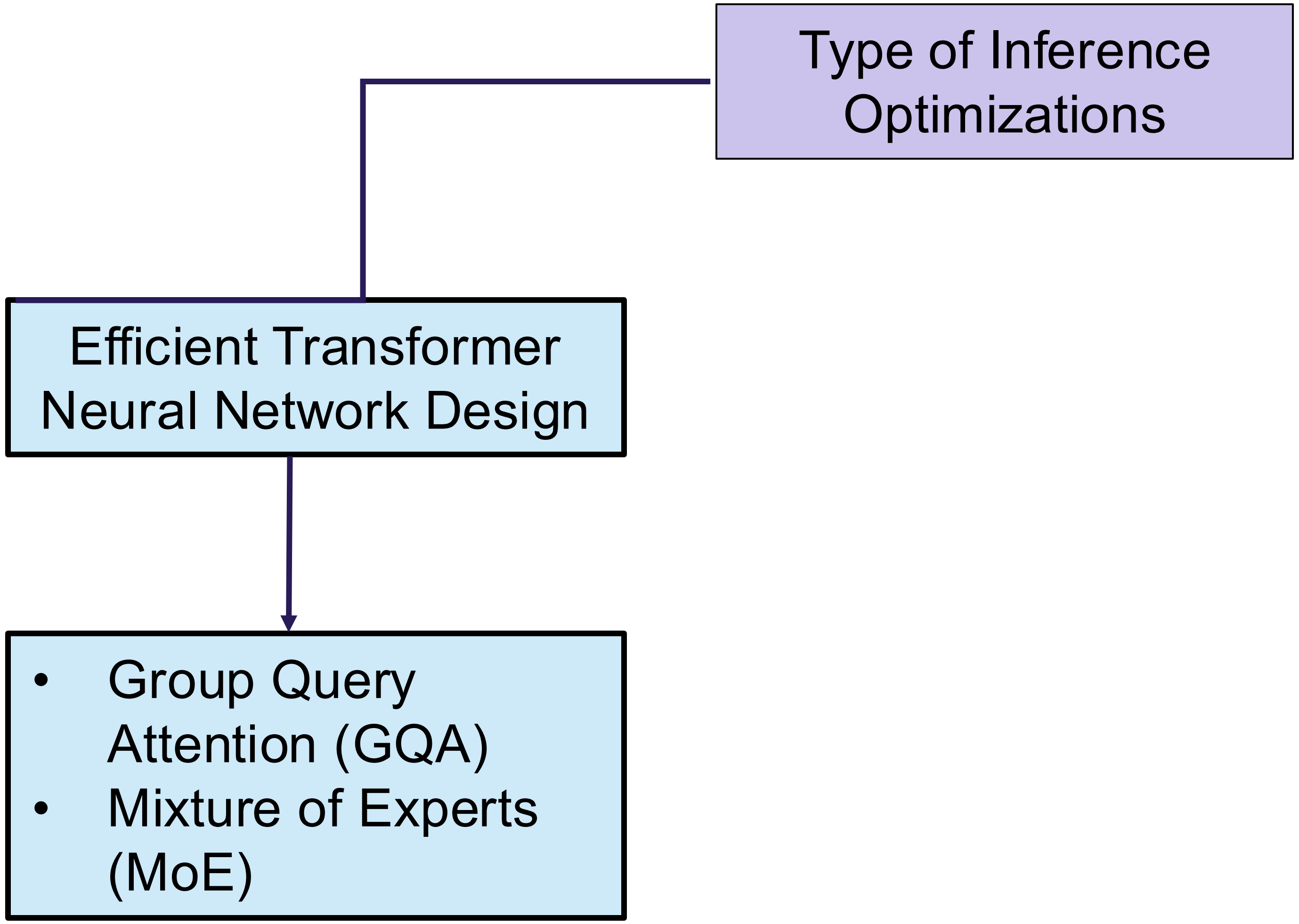
Time Per Output Token refers to the average time interval between generating consecutive tokens.



Classification of LLM Inference Methods


Type of Inference Optimizations
Efficient Transformer Neural Network Design
LLM Algorithm Optimization Methods
Hardware and memory management Techniques
• Group Query Attention (GQA)
• Mixture of Experts (MoE)
• Pruning
• Quantization
• KV Caching
• Speculative Decoding
• PaggedAttention
• Parallelism Methods
• Tensor
• Pipeline
• Expert
• Hybrid

Classification of LLM Inference Methods

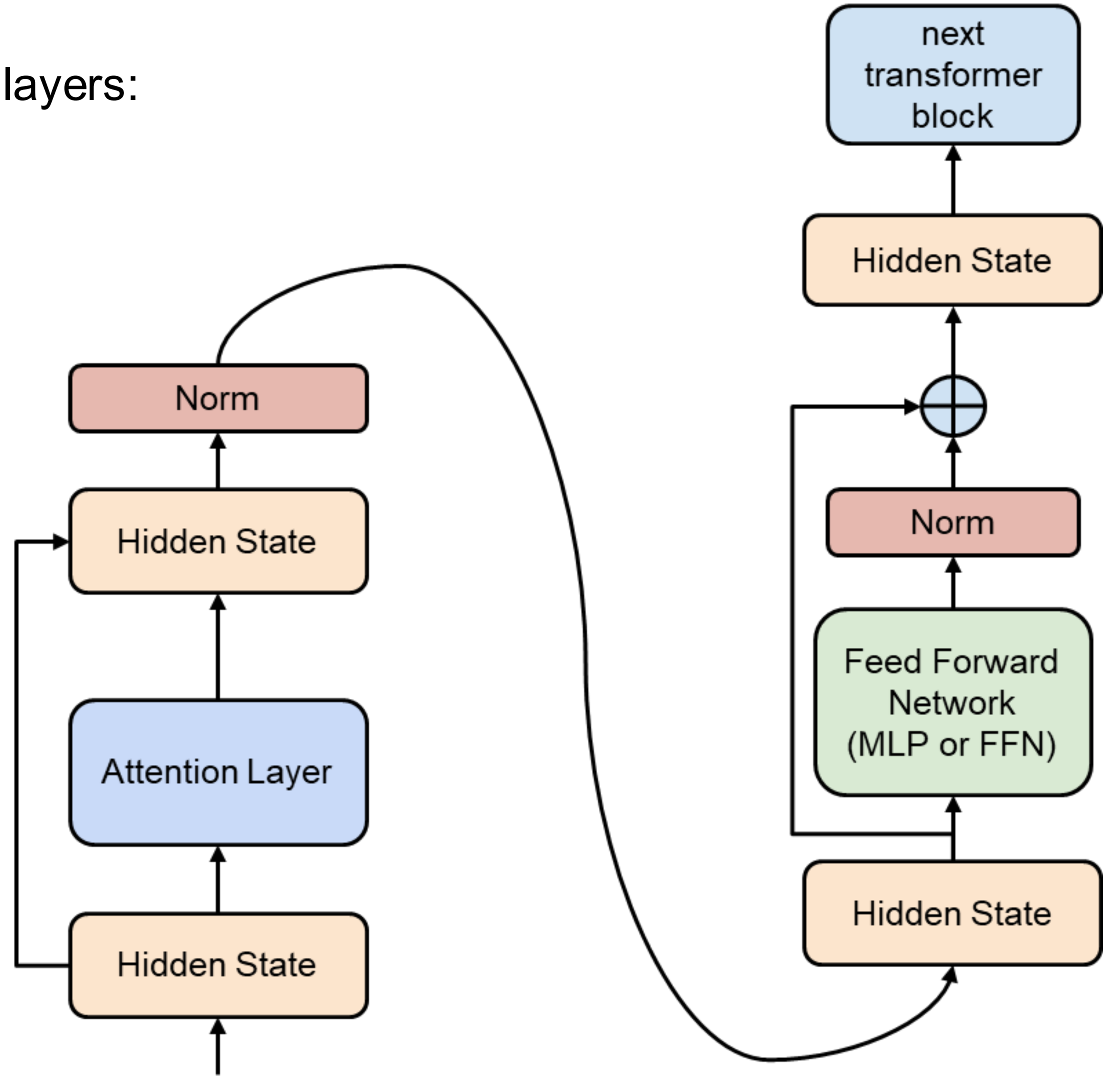

Type of Inference Optimizations
Efficient Transformer Neural Network Design
• Group Query Attention (GQA)
• Mixture of Experts (MoE)

Transformer Model

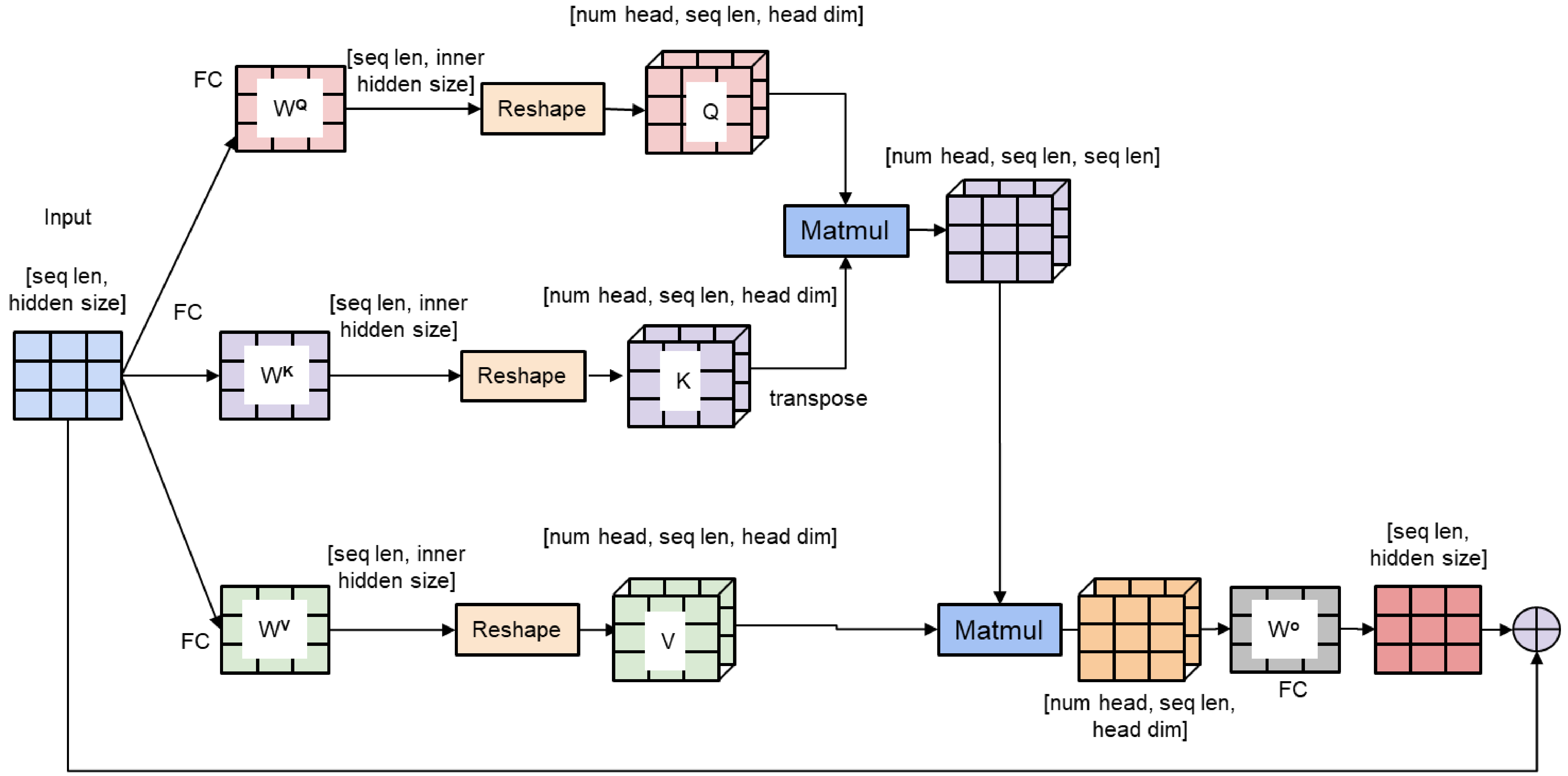
Transformer Neural Network is comprised of two important layers:

1) Attention Layer

2) Feed Forward Network (FFN)

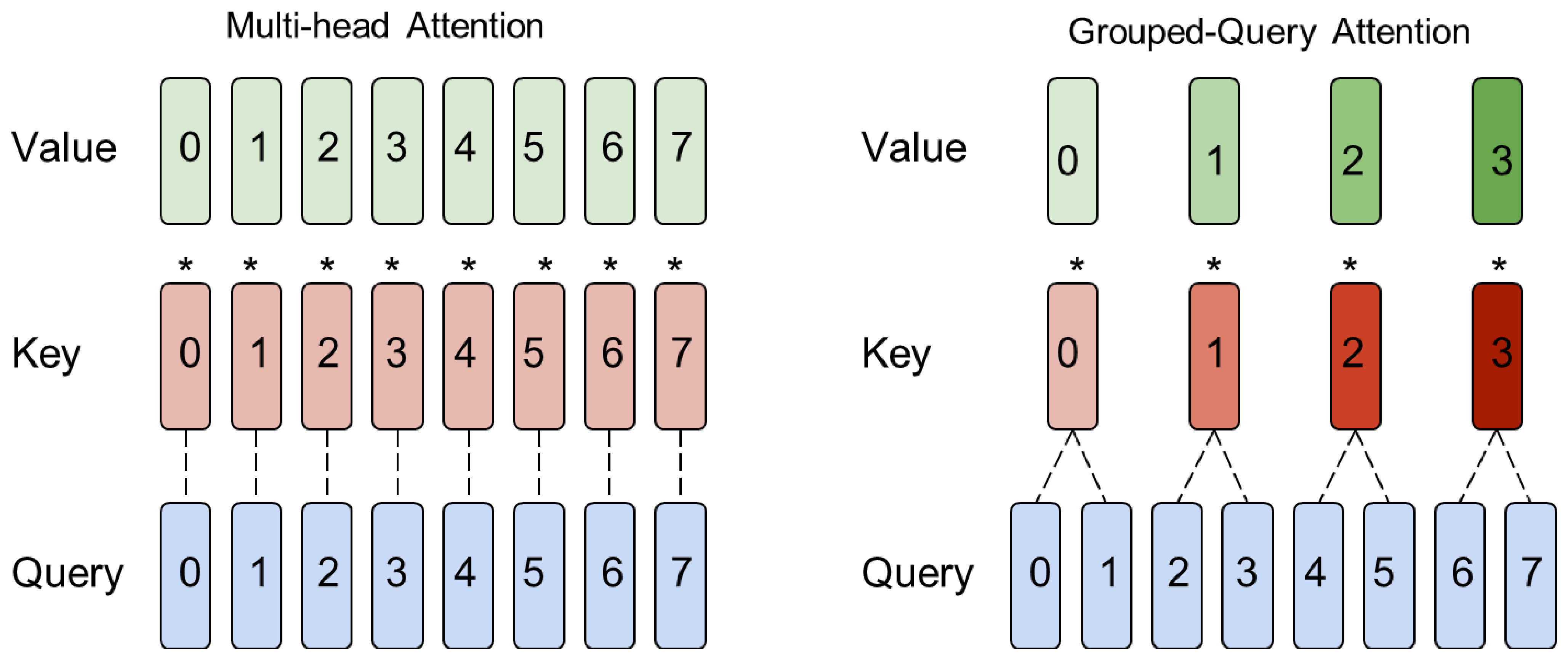


Multi Head Self Attention



[num head, seq len, head dim]
FC
W^Q
[seq len, inner hidden size]
Reshape
Q
[num head, seq len, seq len]
Input
Matmul
[seq len, hidden size]
FC
W^K
[seq len, inner hidden size]
[num head, seq len, head dim]
Reshape
K
transpose
[num head, seq len, head dim]
[seq len, inner hidden size]
[seq len, hidden size]
FC
W^V
Reshape
V
Matmul
W^o
FC
[num head, seq len, head dim]

Multi-head Attention (MHA) vs Group Query Attention (GQA)



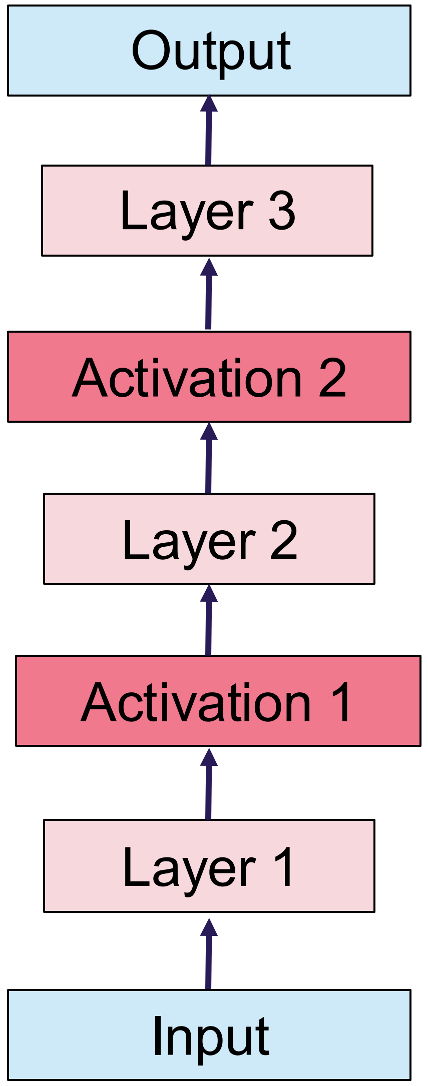
- GQA reduces memory and compute by a factor of "group size"

Dense Model

Dense LLMs are the traditional layer-by-layer transformer models

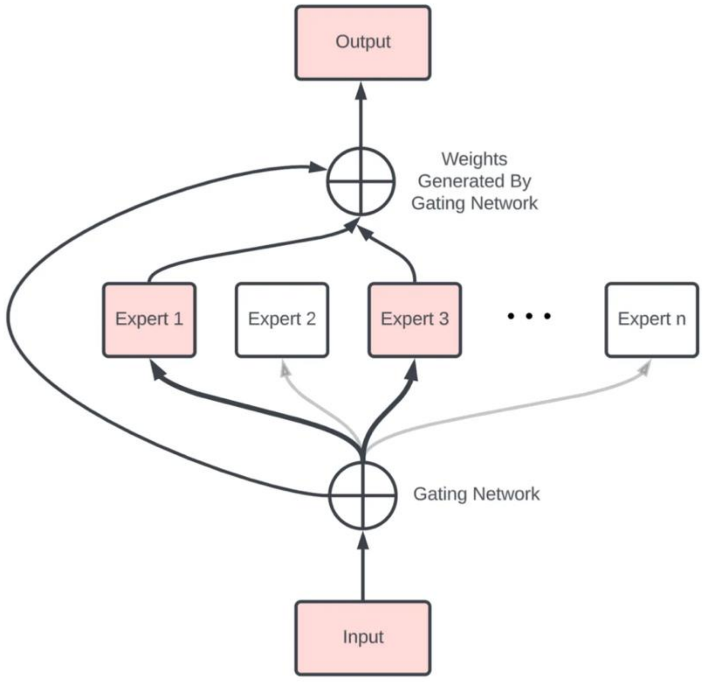
In Dense models, all the layers are connected in series and the activations are processed layer-by-layer

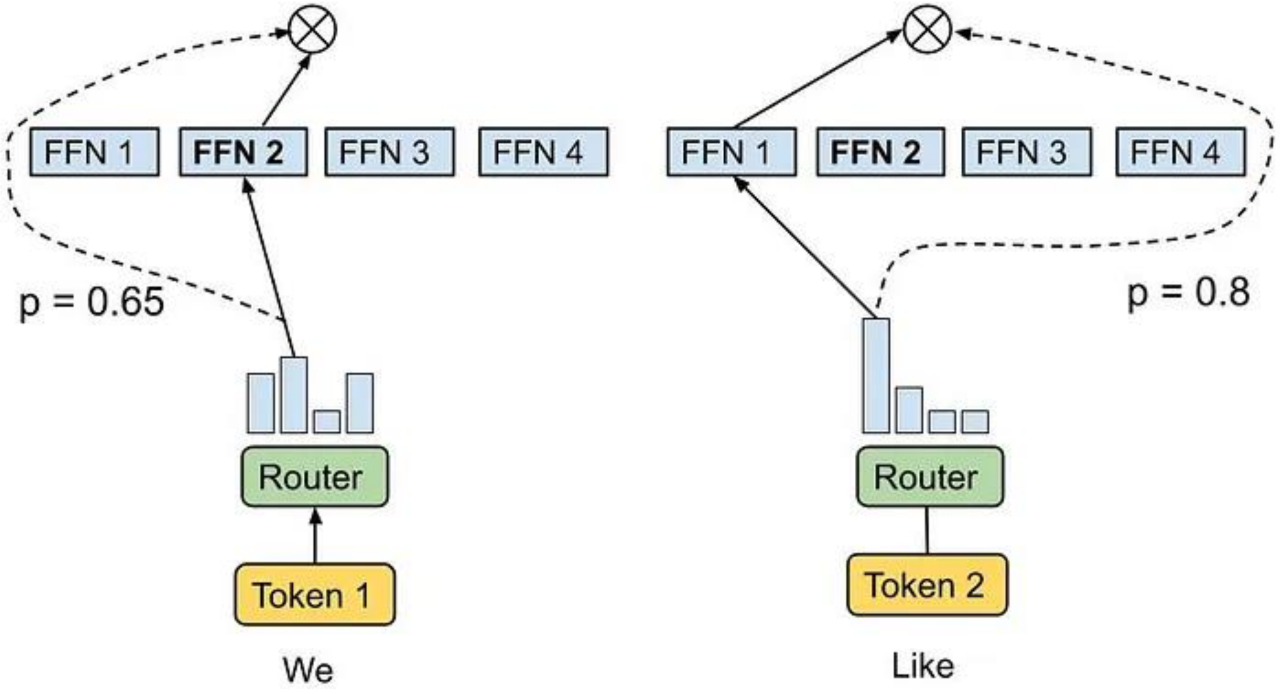
Example: LLaMA-2-7B, LLaMA-3-8B



Mixture of Experts (MoE)

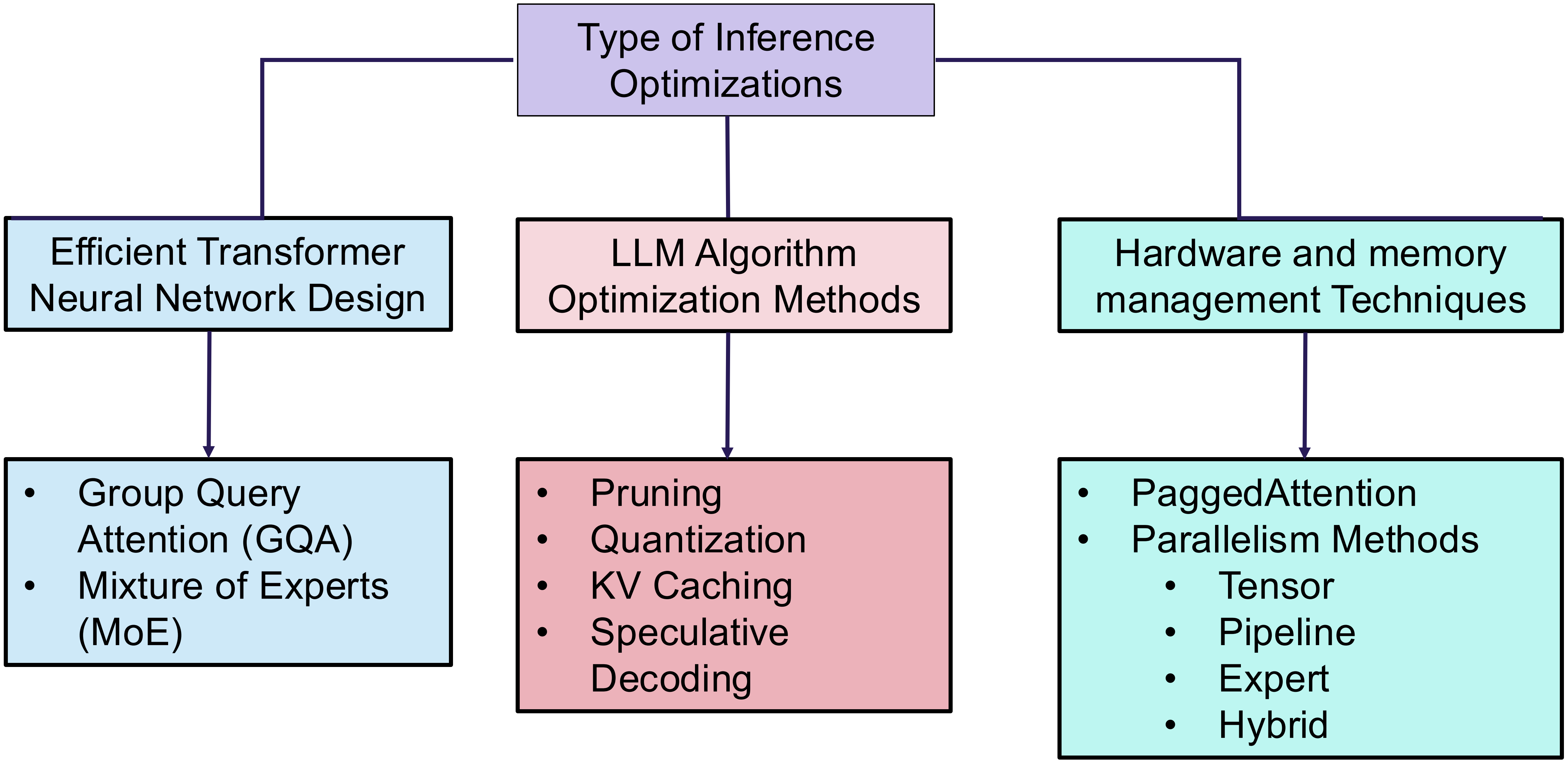
- MoE models employ a combination of specialized sub-networks called experts and a gating mechanism to selectively activate only a subset of parameters for each input
- In a regular dense LLM, all parameters are active while in MoE only a few model weights are utilized, decreasing the compute time of the layer
- Example: Mixtral-8x7B, GPT-MoE 1.8T



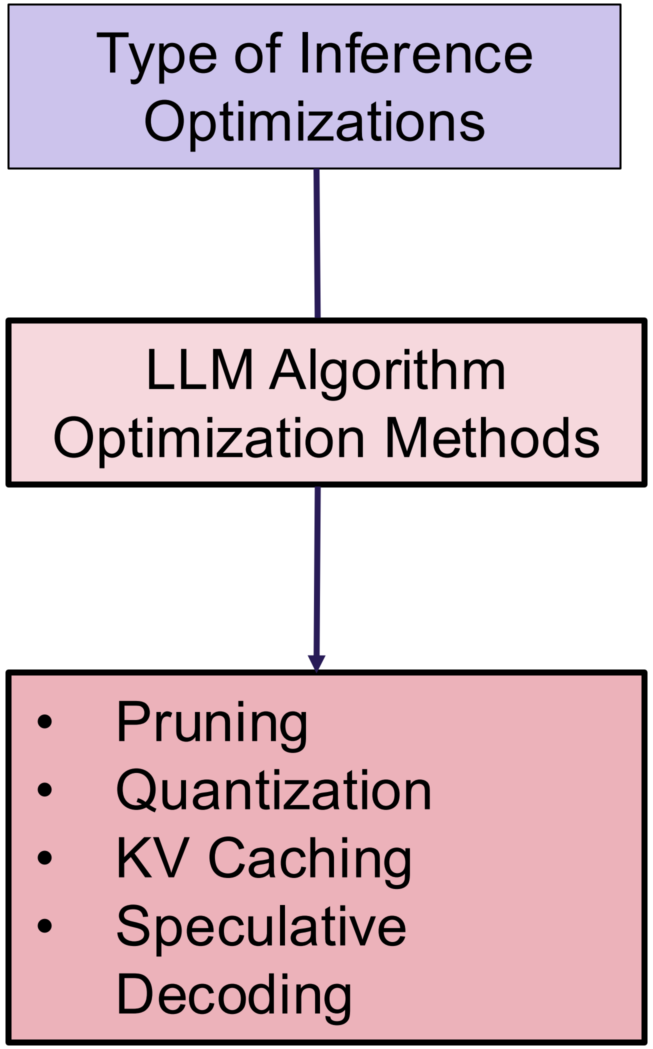


https://deepgram.com/learn/mixture-of-experts-ml-model-guide

Classification of LLM Inference Methods

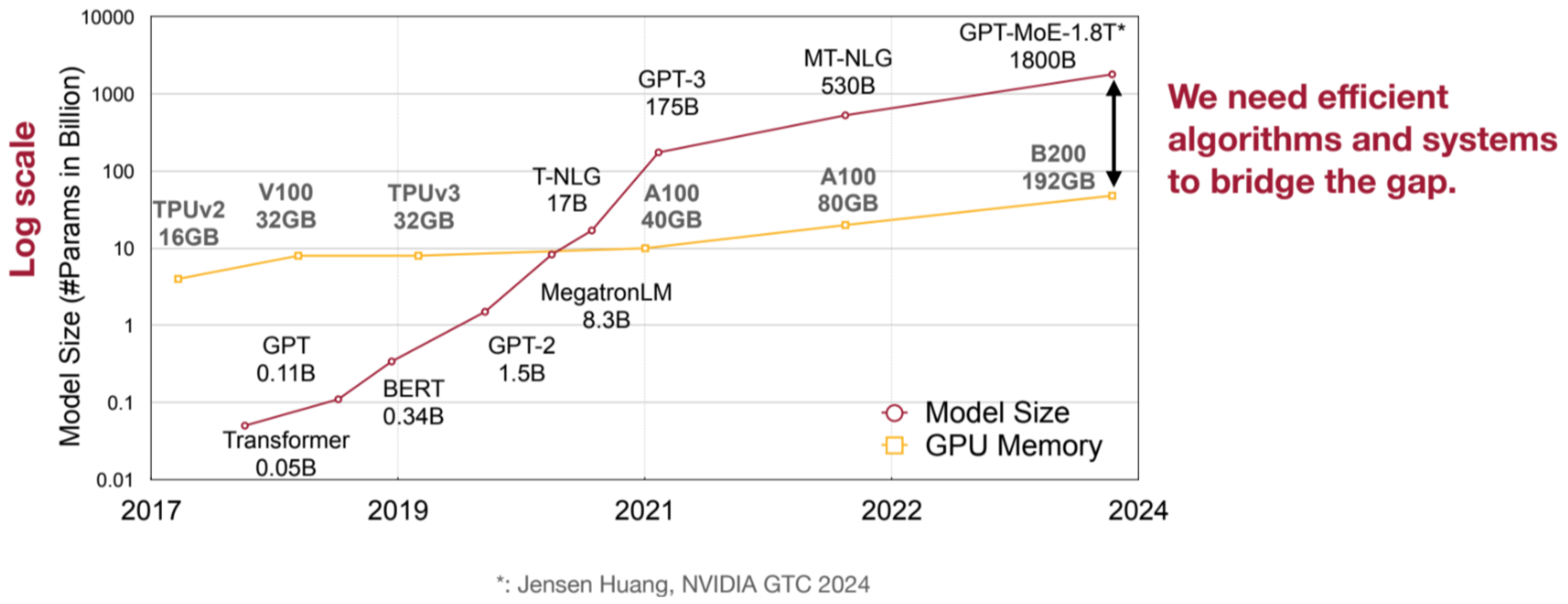

Type of Inference Optimizations
Efficient Transformer Neural Network Design
LLM Algorithm Optimization Methods
Hardware and memory management Techniques
• Group Query Attention (GQA)
• Mixture of Experts (MoE)
• Pruning
• Quantization
• KV Caching
• Speculative Decoding
• PaggedAttention
• Parallelism Methods
• Tensor
• Pipeline
• Expert
• Hybrid

Classification of LLM Inference Methods


Type of Inference Optimizations
LLM Algorithm Optimization Methods
• Pruning
• Quantization
• KV Caching
• Speculative Decoding

Challenge for LLM deployment: colossal sizes

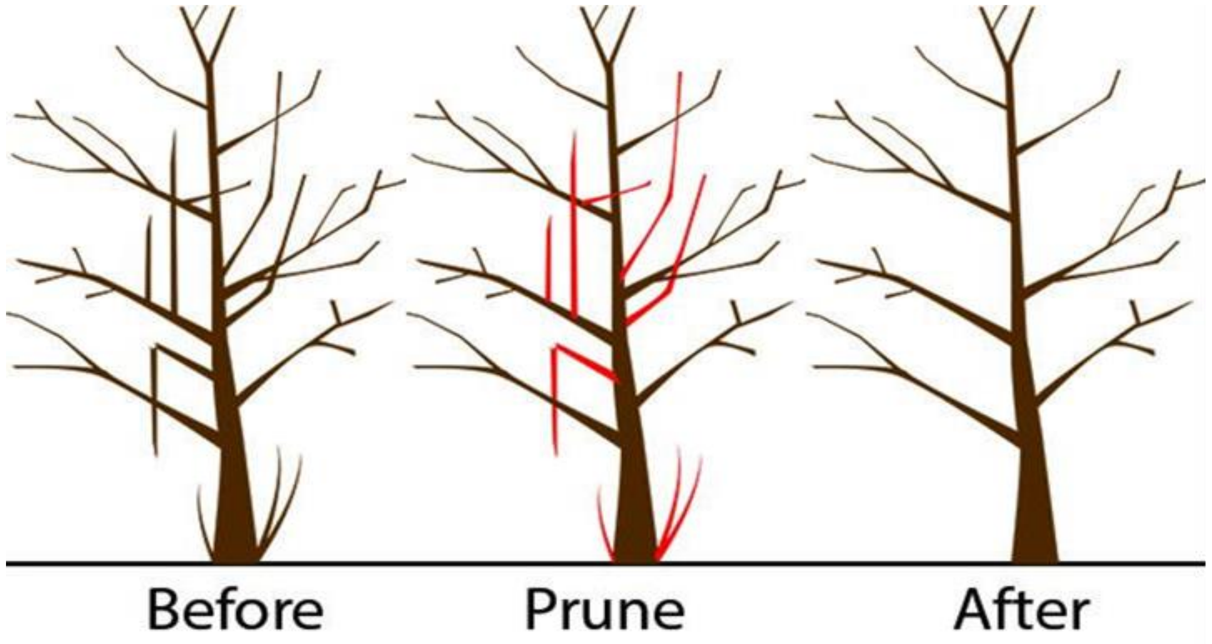
- Despite being powerful, LLMs are hard to serve
- LLM sizes and computation are increasing exponentially
- We need model compression techniques and system support to bridge the gap

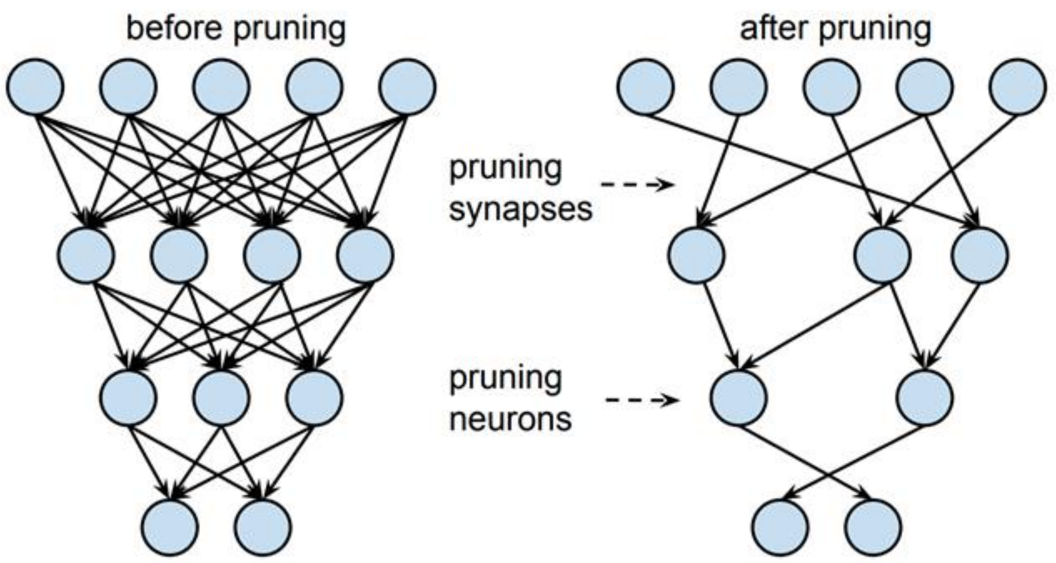


Lin et al. AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration

Pruning

- Pruning in general refers to cutting down branches/leaves
- Neural Network Pruning refers to removing weights/connection without compromising the accuracy
- The parameters which do not contribute to the final accuracy are removed to save memory
- Example: Magnitude-based

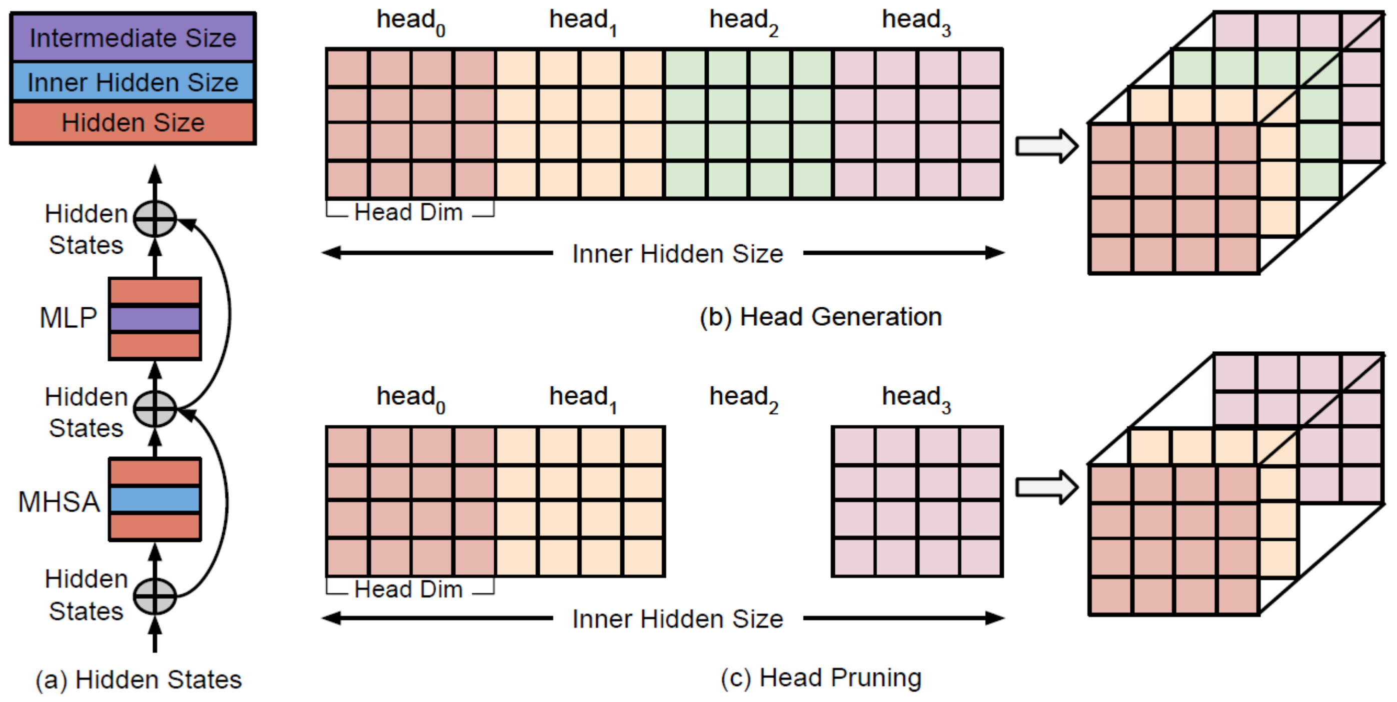




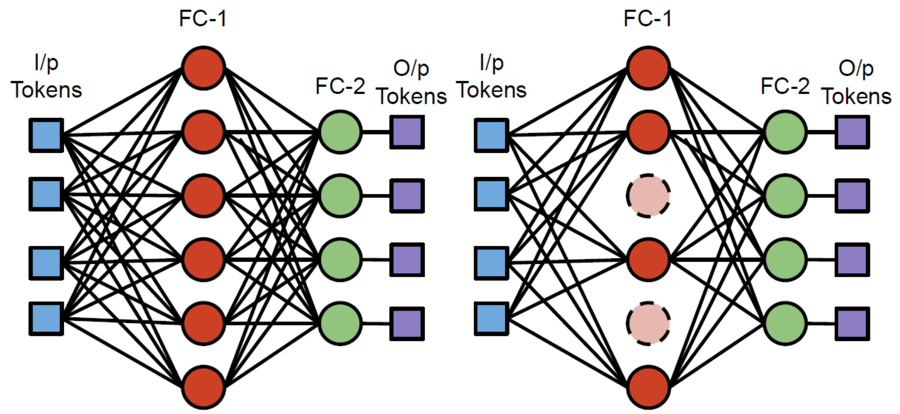
Han et al. "Learning both Weights and Connections for Efficient Neural Networks"

LLM Pruning Methods

- LLM pruning refers to pruning to pruning Heads or FFN layer in the transformer model
- LLM Pruning Examples: LLM-Pruner, WActiGrad



Head pruning

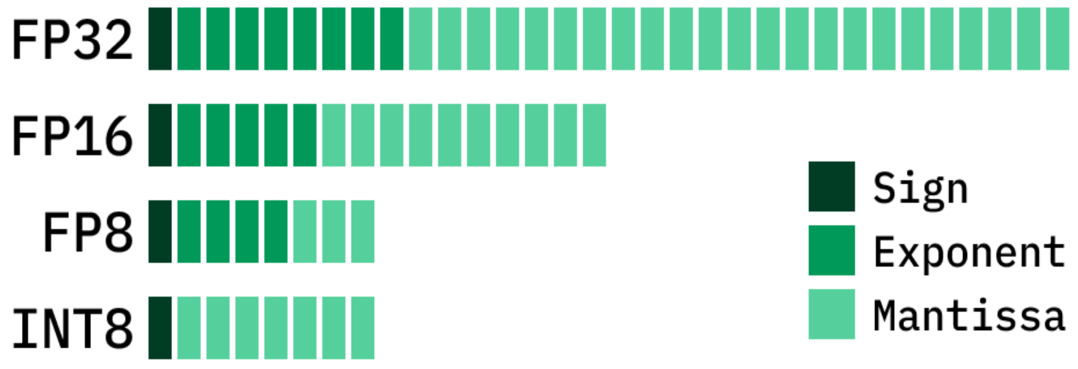


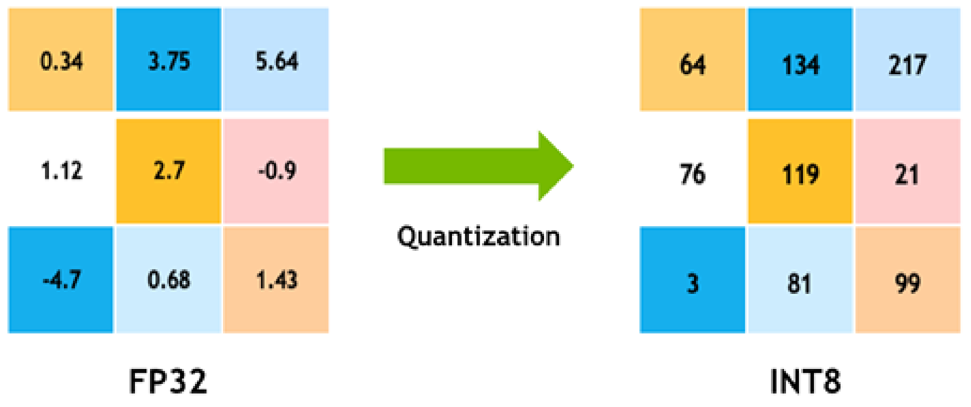
FFN pruning

KT Chitty-Venkata et al. "WActiGrad: Structured Pruning for Efficient Finetuning and Inference of Large Language Models on AI Accelerators"

Quantization

- Quantization converts the weights and activations of a model from a high-precision data representation to a lower-precision data representation
- Quantization helps in reducing the model size by using less number of bits to represent the parameters
- For example, quantizing FP32 to Int8 reduces the model size by 4x
- Quantization is extremely important for LLM inference, given the size of complexity of the architecture



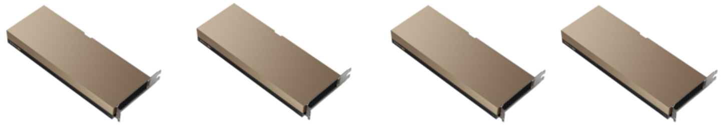


https://developer.nvidia.com/blog/achieving-fp32-accuracy-for-int8-inference-using-quantization-aware-training-with-tensorrt/

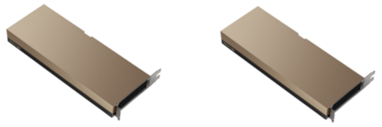
Benefits of LLM Quantization

LLaMa-3-70B model requires at least :

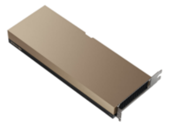
- **FP16:** 140GB memory➞4 x 40GB A100 GPUs



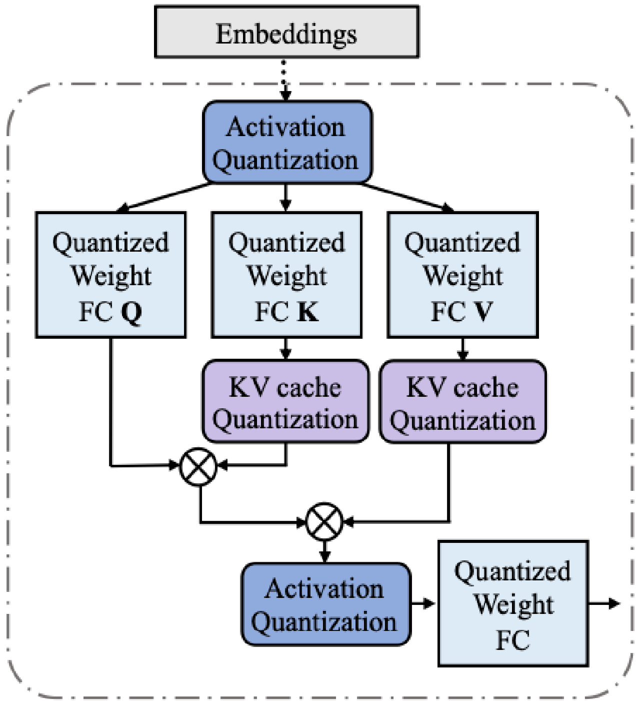
- **INT8:** 70GB memory➞2 x 40GB A100 GPU



- **INT4:** 35GB memory➞1 x 40GB A100 GPU

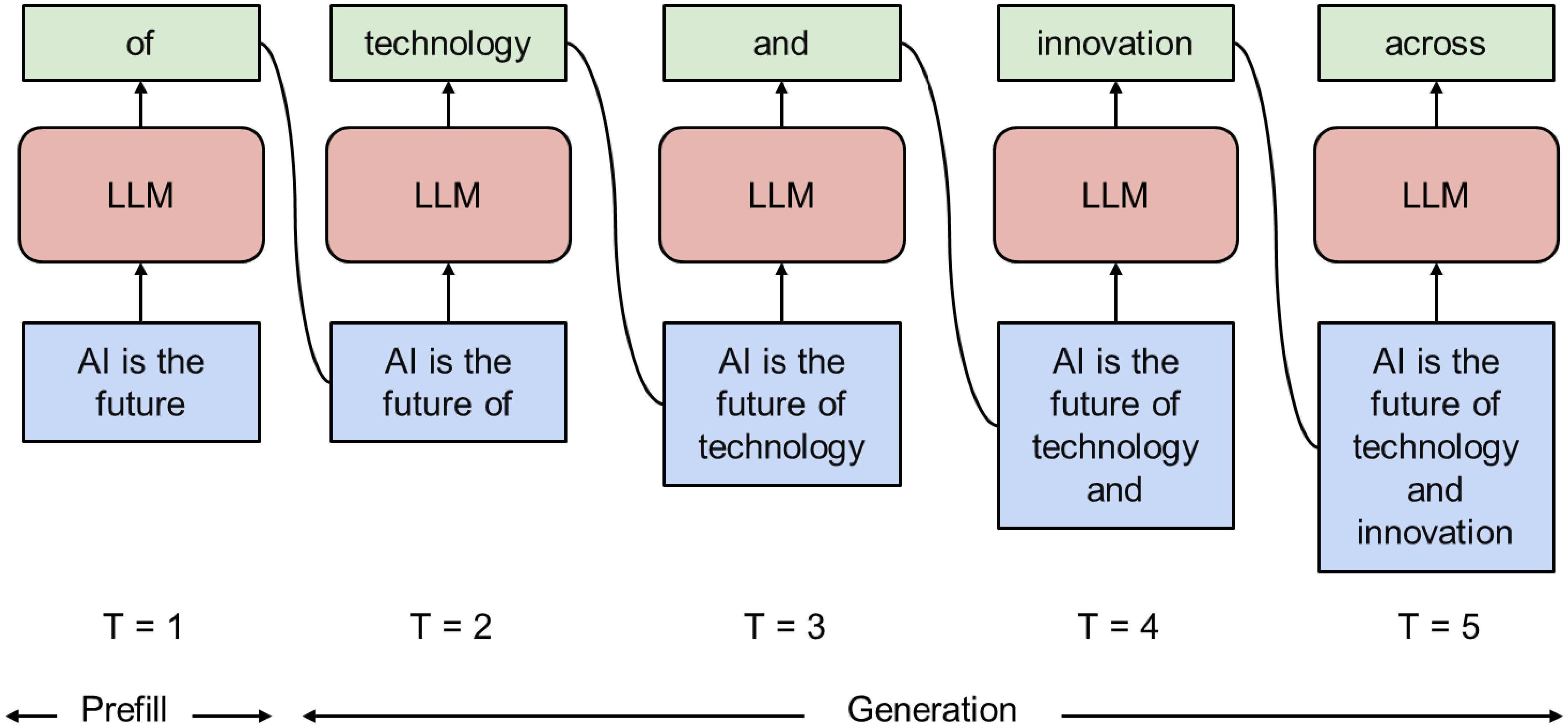


- LLM Quantization Methods: GPTQ, AWQ

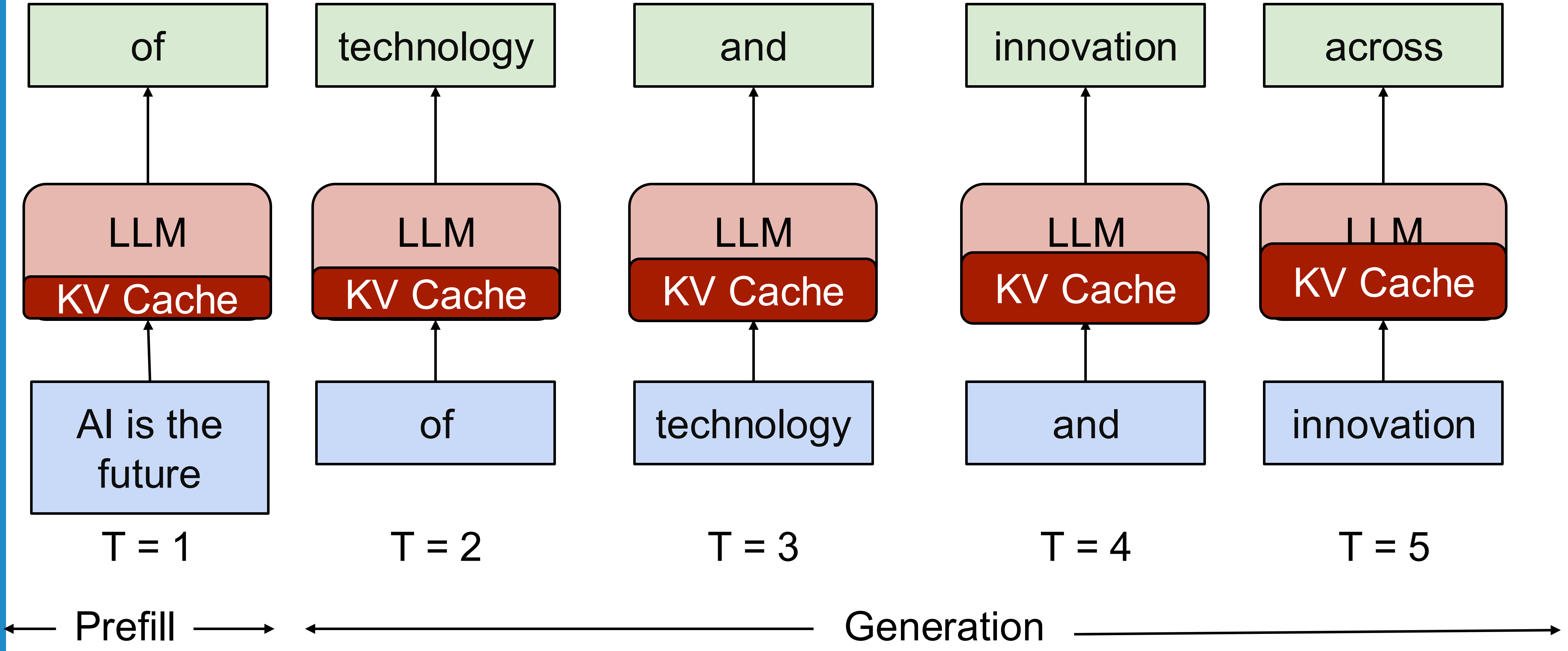


Liu et al. LLM-QAT: Data-Free Quantization Aware Training for Large Language Models

LLM Inference 101

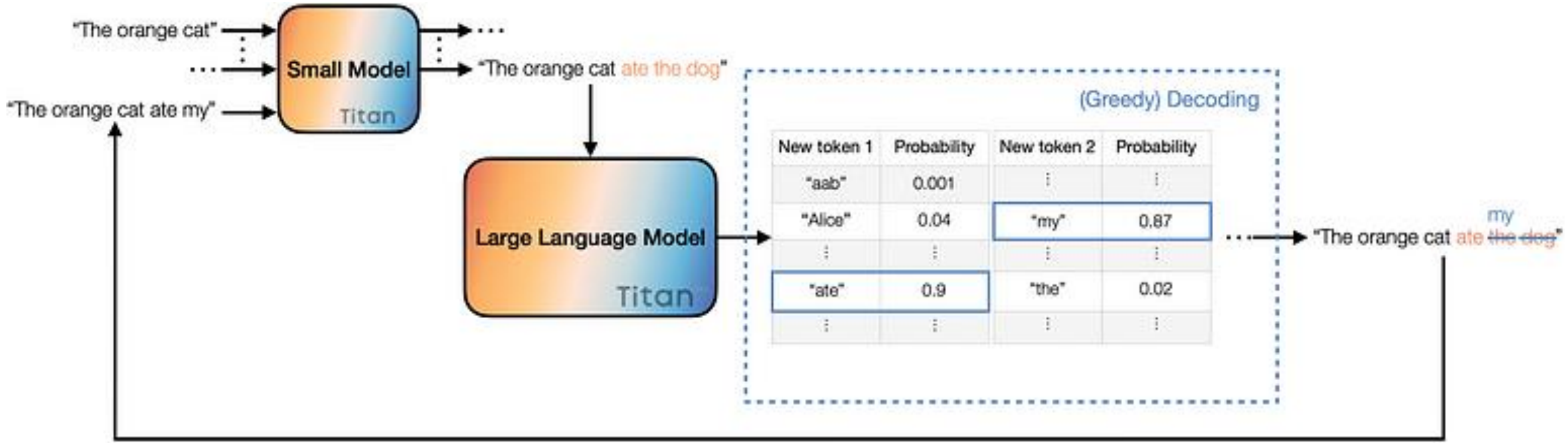


LLM Inference with KV Cache



Speculative Decoding

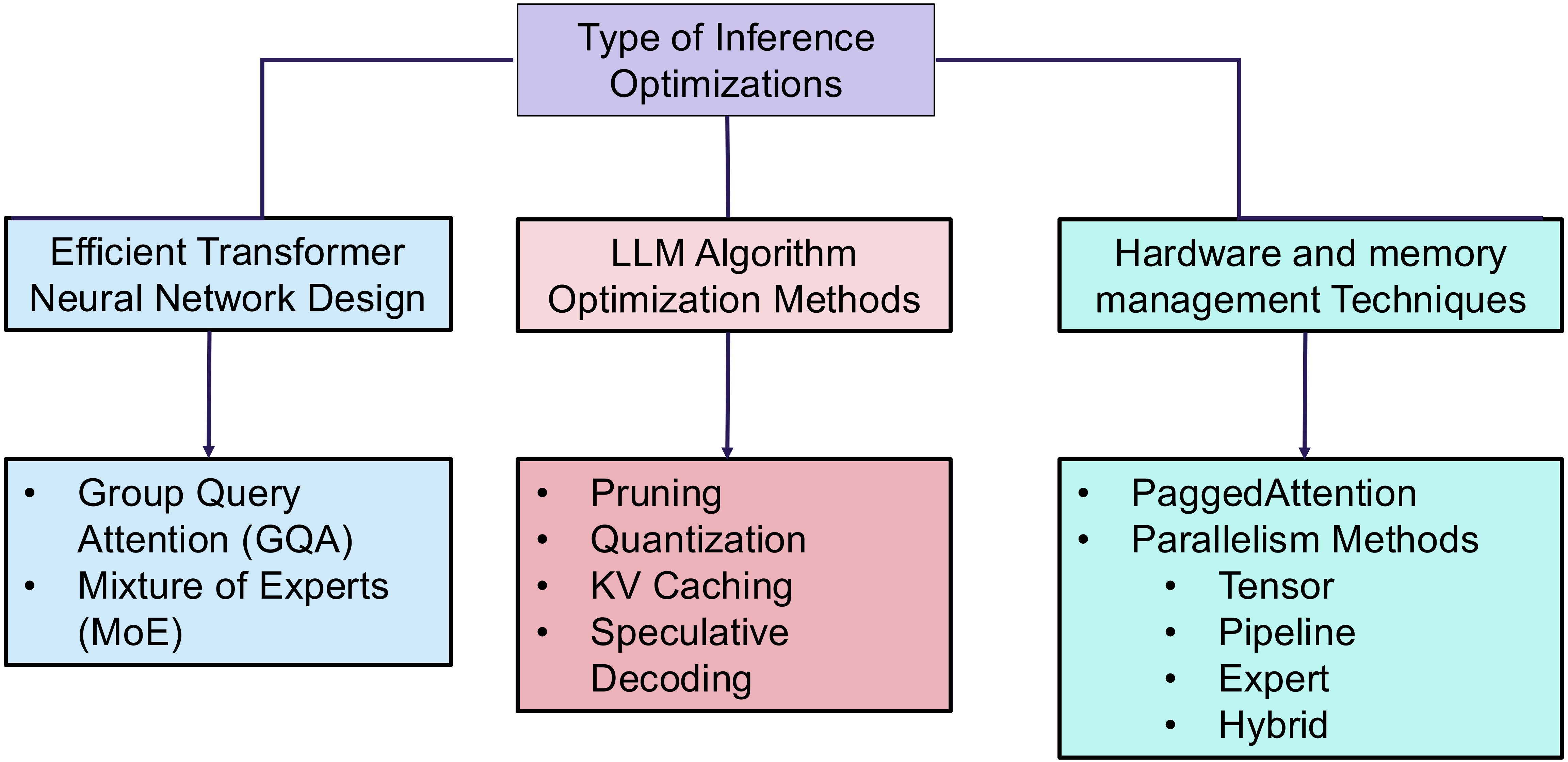
- Speculative Decoding is a widely used technique to speed up inference for LLMs without greatly compromising the output quality
- During inference, the speculative decoding method utilizes a smaller draft model (Eg: OPT-125M) to generate speculative tokens and then uses the larger LLM (LLaMA-2-7B) to verify those draft tokens.
- Both draft model and the main model should have the same vocab size



New token 1	Probability	New token 2	Probability
"aab"	0.001	⋮	⋮
"Alice"	0.04	"my"	0.87
⋮	⋮	⋮	⋮
"ate"	0.9	"the"	0.02
⋮	⋮	⋮	⋮

https://www.titanml.co/glossary/speculative-decoding

Classification of LLM Inference Methods

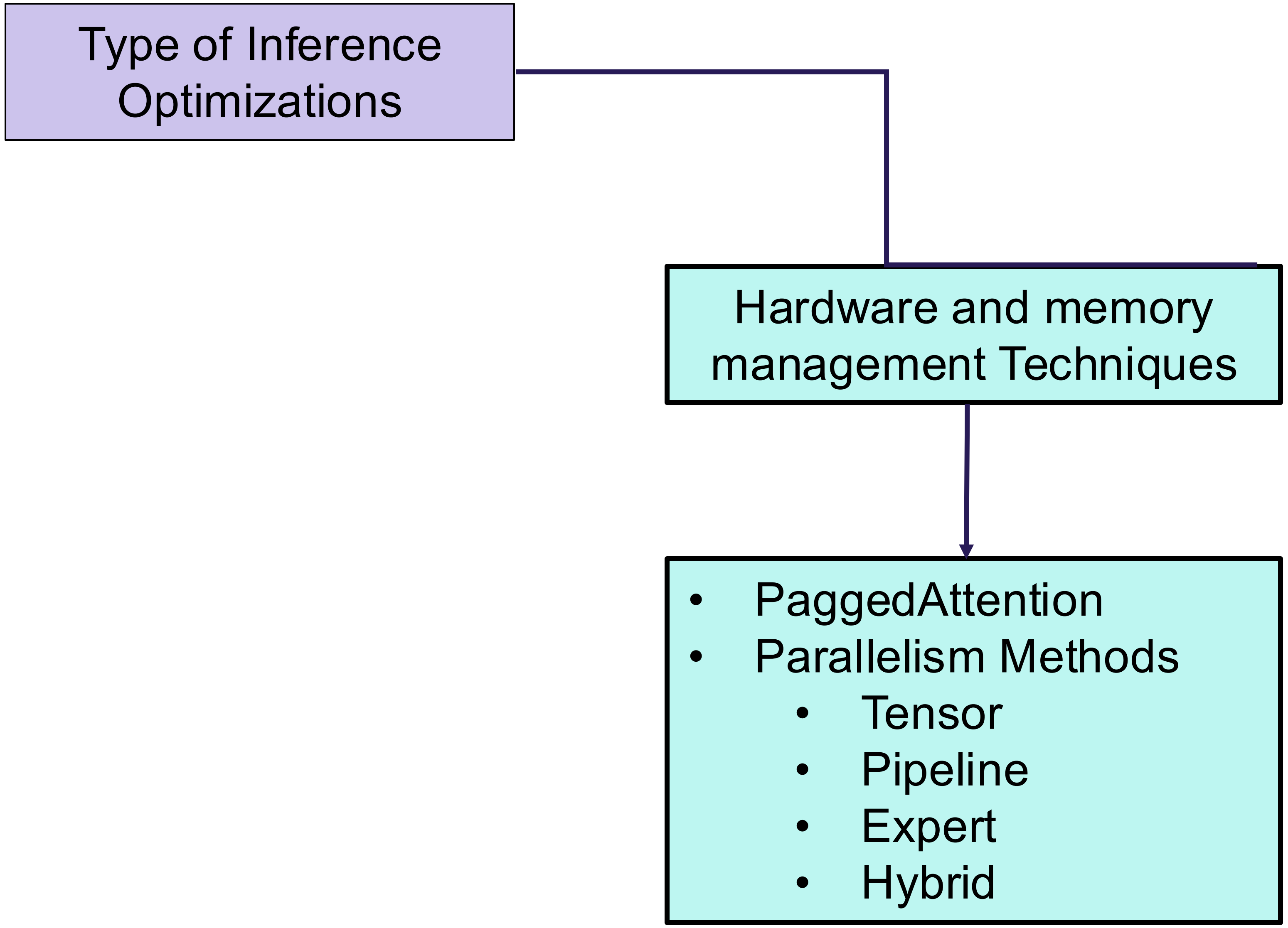

Type of Inference Optimizations
Efficient Transformer Neural Network Design
LLM Algorithm Optimization Methods
Hardware and memory management Techniques

- Group Query Attention (GQA)
- Mixture of Experts (MoE)

- Pruning
- Quantization
- KV Caching
- Speculative Decoding

- PaggedAttention
- Parallelism Methods
  - Tensor
  - Pipeline
  - Expert
  - Hybrid

Classification of LLM Inference Methods



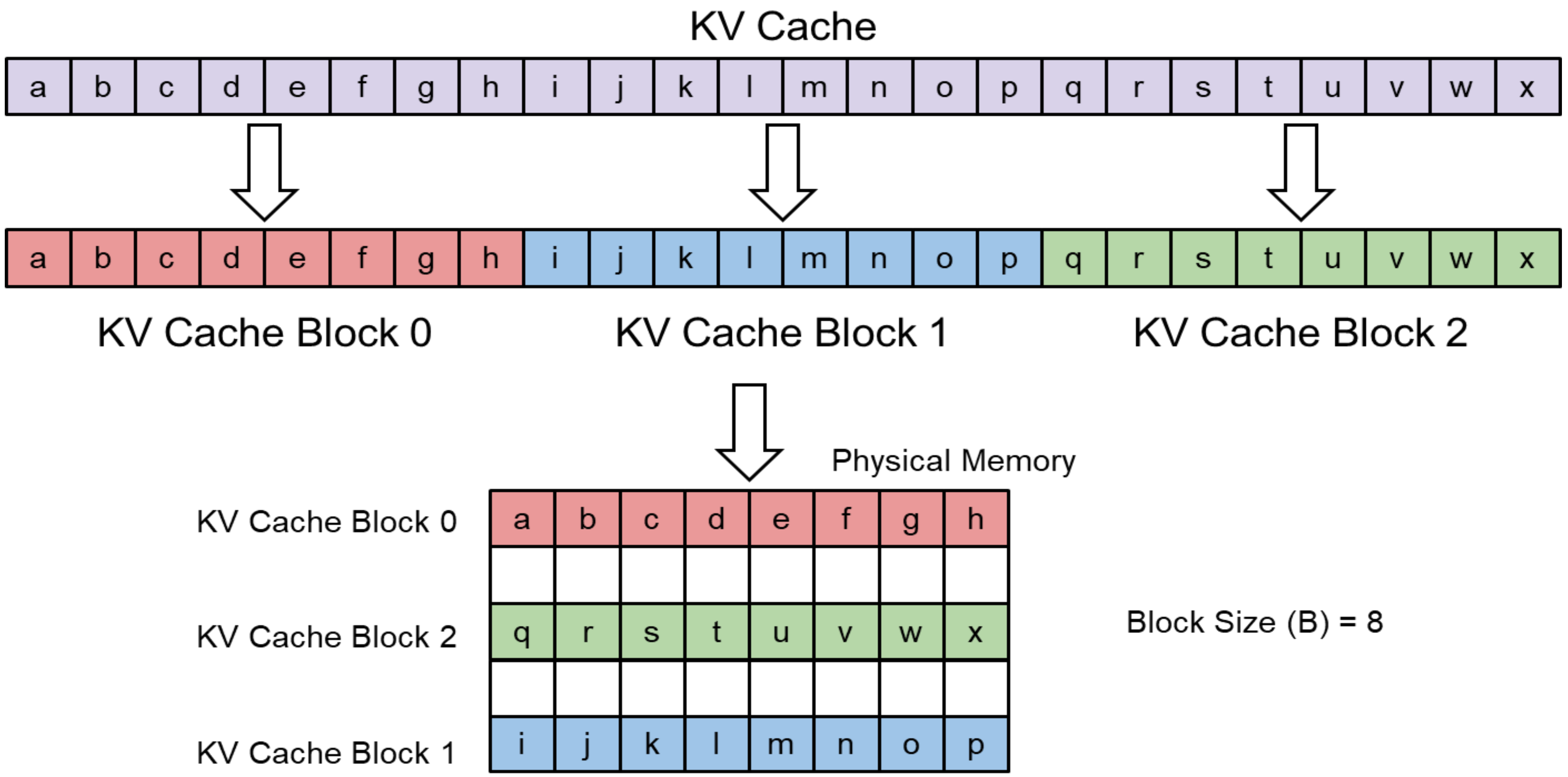
Type of Inference Optimizations

Hardware and memory management Techniques

- PaggedAttention
- Parallelism Methods
  - Tensor
  - Pipeline
  - Expert
  - Hybrid

PagedAttention

- Paged Attention partitions the KV cache of each sequence into smaller, more manageable "**pages**" or "**blocks**". Each block contains key-value vectors for a fixed number of tokens.

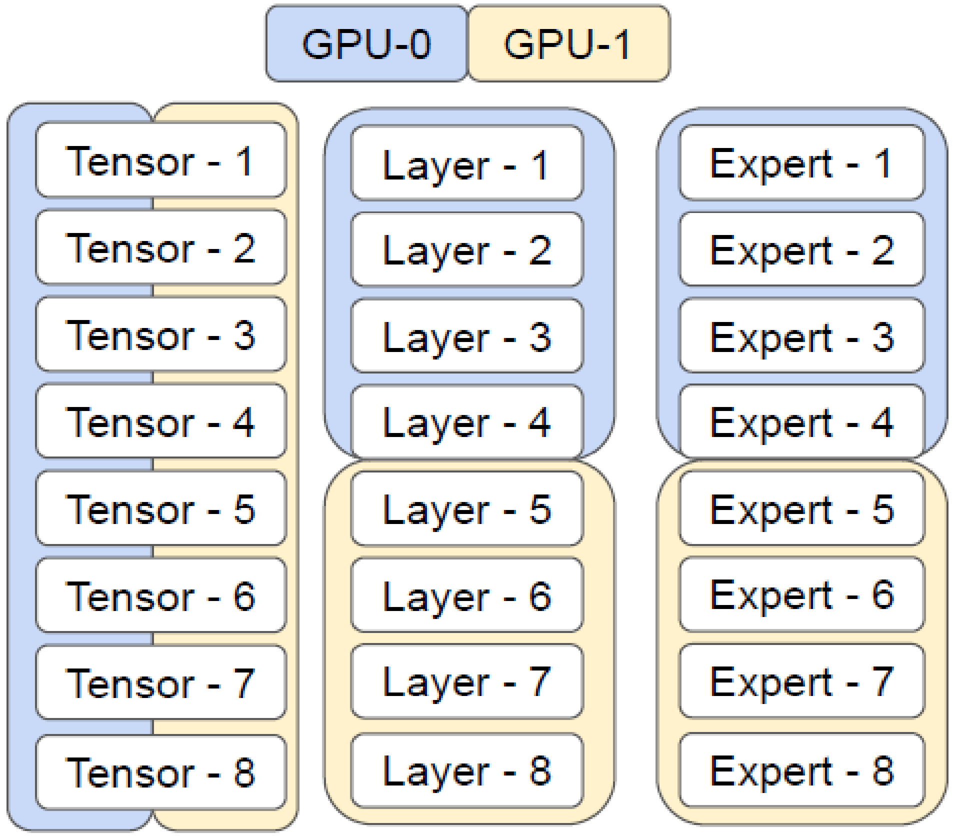


Parallelism Techniques

- **Tensor Parallelism**
  - Distributes the weight tensor of a layer across multiple devices.
  - The devices communicate with each other to share the input and output activations.

- **Pipeline Parallelism**
  - Divides the model into different layers, and each device computes its assigned layers and passes the output to the next device in the pipeline.

- **Expert Parallelism**
  - Distributes the experts of the MoE model across multiple devices.



(a) Tensor Parallelism (b) Pipeline Parallelism (c) Expert Parallelism

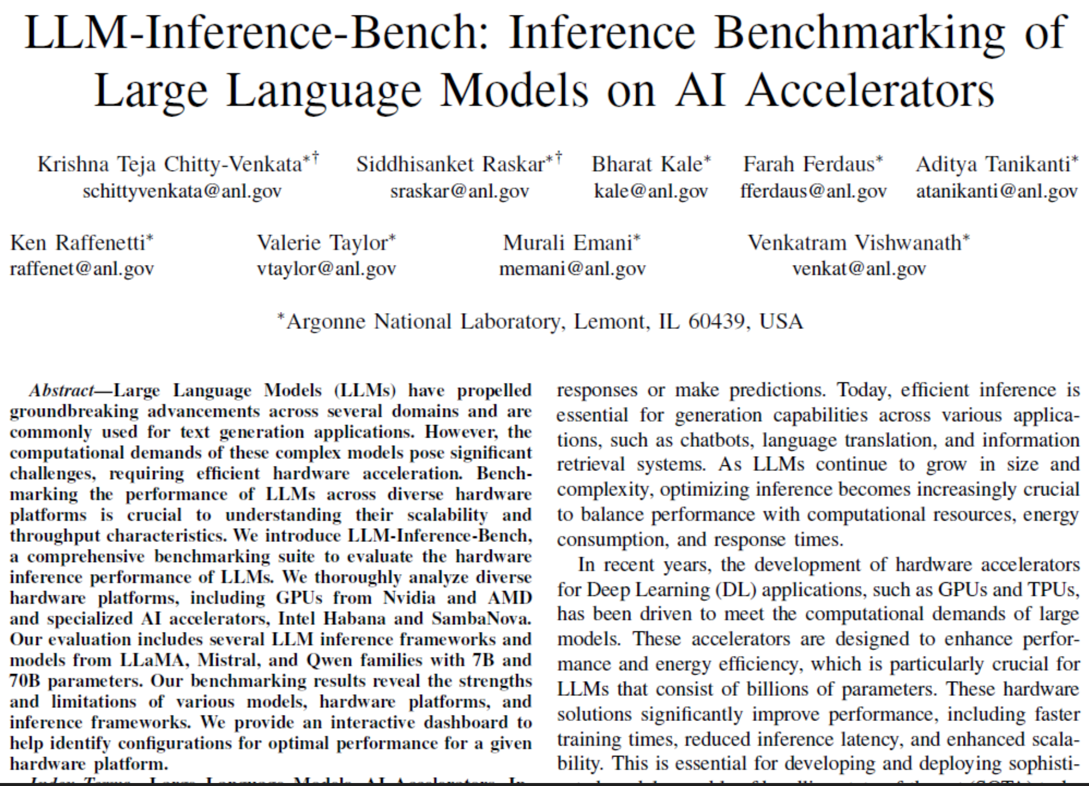
Inference Frameworks

- There has been a rise in Inference frameworks for LLM over the last few years

vLLM	TensorRT-LLM	Deepspeed-MII	LLaMA.cpp
Can run on diverse hardware platforms including Intel, Nvidia, AMD GPUs and AI accelerators such as Graphcore and Habana	Limited to Nvidia GPUs, such as A100, H100, GH200 series	Limited to Nvidia GPUs (such as A100, H100, GH200)	Can run on diverse hardware platforms including Intel, Nvidia, AMD GPUs
Supports wide range of Inference Optimizations	Supports wide range of Inference Optimizations	Lacks many key LLM optimizations instead relies on GPU kernel optimizations	Lacks many optimizations and does not scale with increase in number of GPUs
Has wide Community Support	Developed within Nvidia	Developed within Microsoft	Has wide Community support

vLLM DeepSpeed MII NVIDIA TensorRT-LLM LLaMA C++

LLM Inference Publication



LLM-Inference-Bench: Inference Benchmarking of Large Language Models on AI Accelerators

Krishna Teja Chitty-Venkata*† schittyvenkata@anl.gov
Siddhisanket Raskar*† sraskar@anl.gov
Bharat Kale* kale@anl.gov
Farah Ferdaus* fferdaus@anl.gov
Aditya Tanikanti* atanikanti@anl.gov
Ken Raffenetti* raffenet@anl.gov
Valerie Taylor* vtaylor@anl.gov
Murali Emani* memani@anl.gov
Venkatram Vishwanath* venkat@anl.gov

*Argonne National Laboratory, Lemont, IL 60439, USA

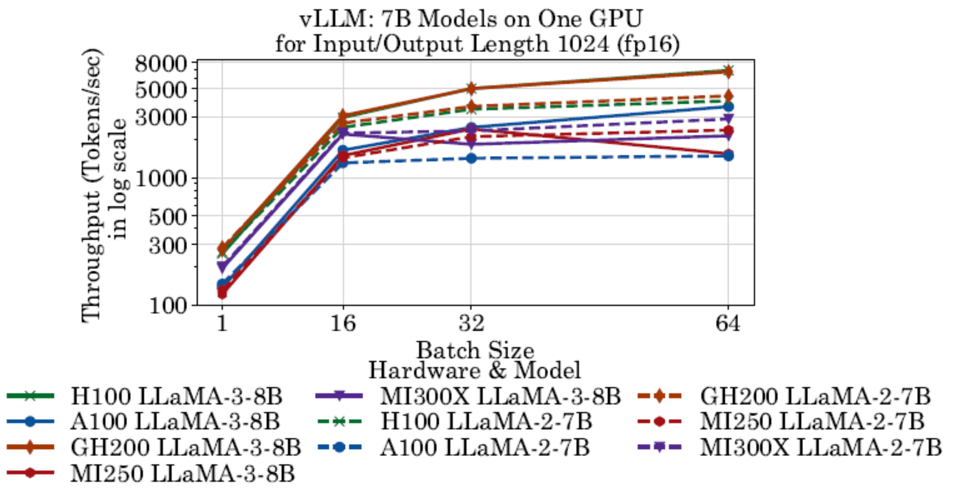
Abstract—**Large Language Models (LLMs) have propelled groundbreaking advancements across several domains and are commonly used for text generation applications. However, the computational demands of these complex models pose significant challenges, requiring efficient hardware acceleration. Benchmarking the performance of LLMs across diverse hardware platforms is crucial to understanding their scalability and throughput characteristics. We introduce LLM-Inference-Bench, a comprehensive benchmarking suite to evaluate the hardware inference performance of LLMs. We thoroughly analyze diverse hardware platforms, including GPUs from Nvidia and AMD and specialized AI accelerators, Intel Habana and SambaNova. Our evaluation includes several LLM inference frameworks and models from LLaMA, Mistral, and Qwen families with 7B and 70B parameters. Our benchmarking results reveal the strengths and limitations of various models, hardware platforms, and inference frameworks. We provide an interactive dashboard to help identify configurations for optimal performance for a given hardware platform.**

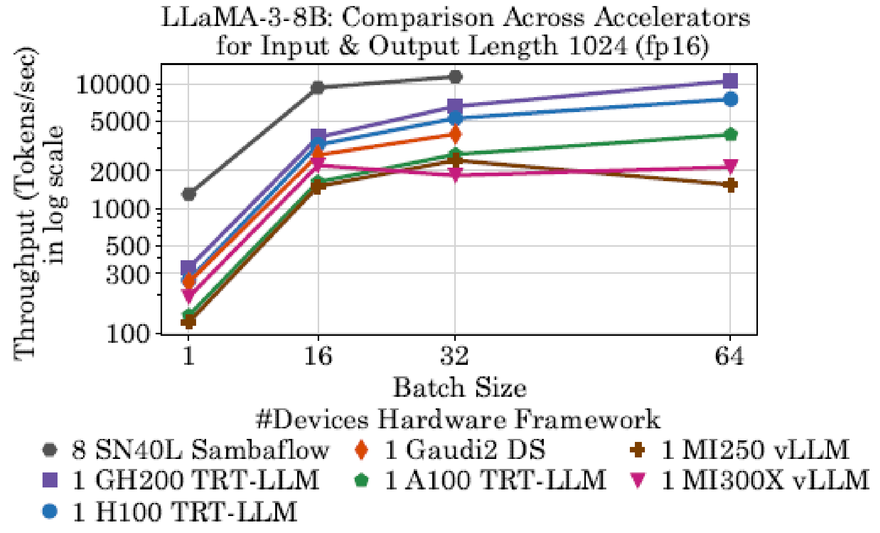
responses or make predictions. Today, efficient inference is essential for generation capabilities across various applications, such as chatbots, language translation, and information retrieval systems. As LLMs continue to grow in size and complexity, optimizing inference becomes increasingly crucial to balance performance with computational resources, energy consumption, and response times.

In recent years, the development of hardware accelerators for Deep Learning (DL) applications, such as GPUs and TPUs, has been driven to meet the computational demands of large models. These accelerators are designed to enhance performance and energy efficiency, which is particularly crucial for LLMs that consist of billions of parameters. These hardware solutions significantly improve performance, including faster training times, reduced inference latency, and enhanced scalability. This is essential for developing and deploying sophisti-

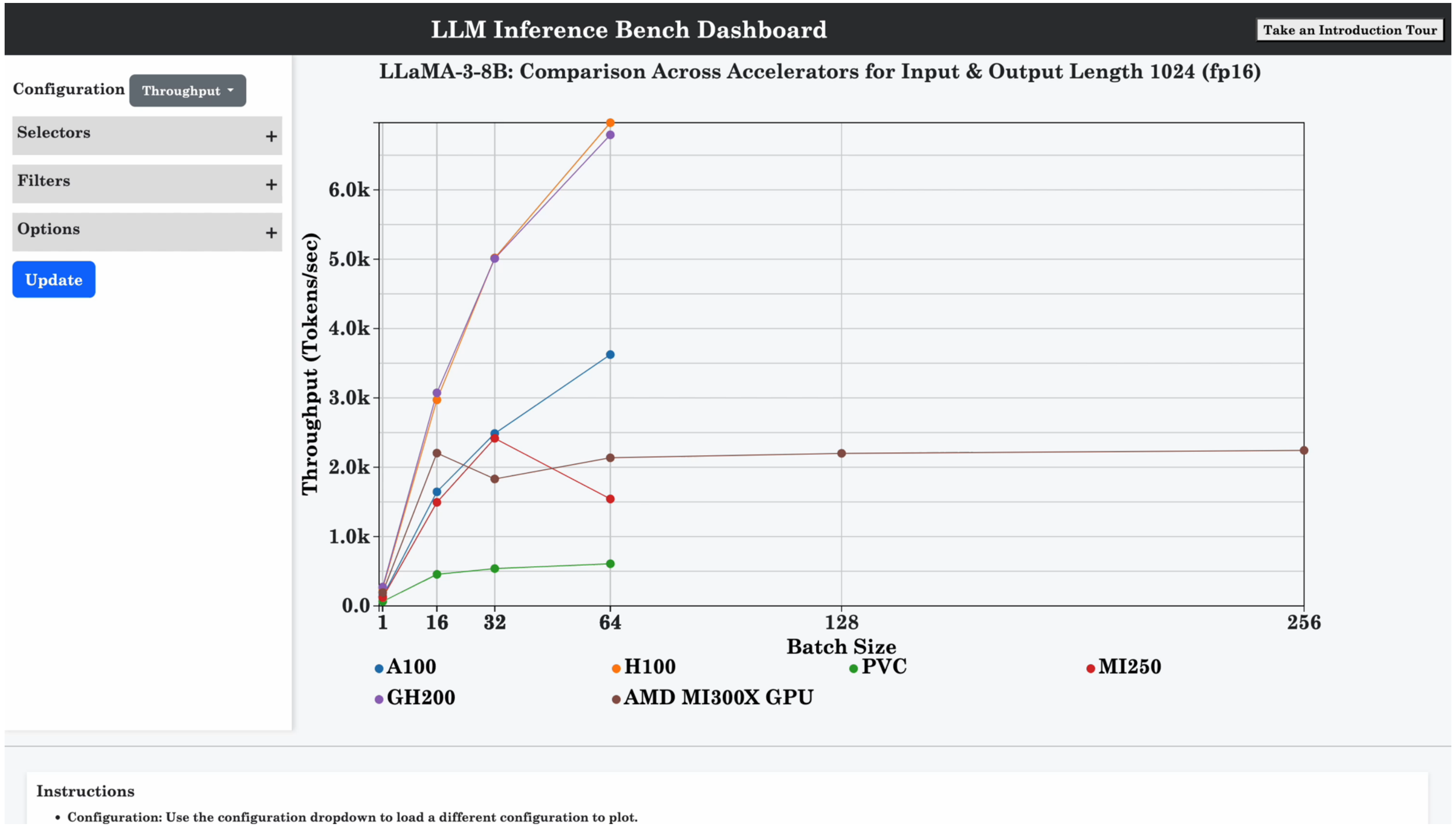
15th IEEE International Workshop on
Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems
held in conjunction with SC24: The International Conference for High Performance Computing, Networking, Storage and Analysis

SC24 Atlanta, GA | hpc creates.





Performance Dashboard

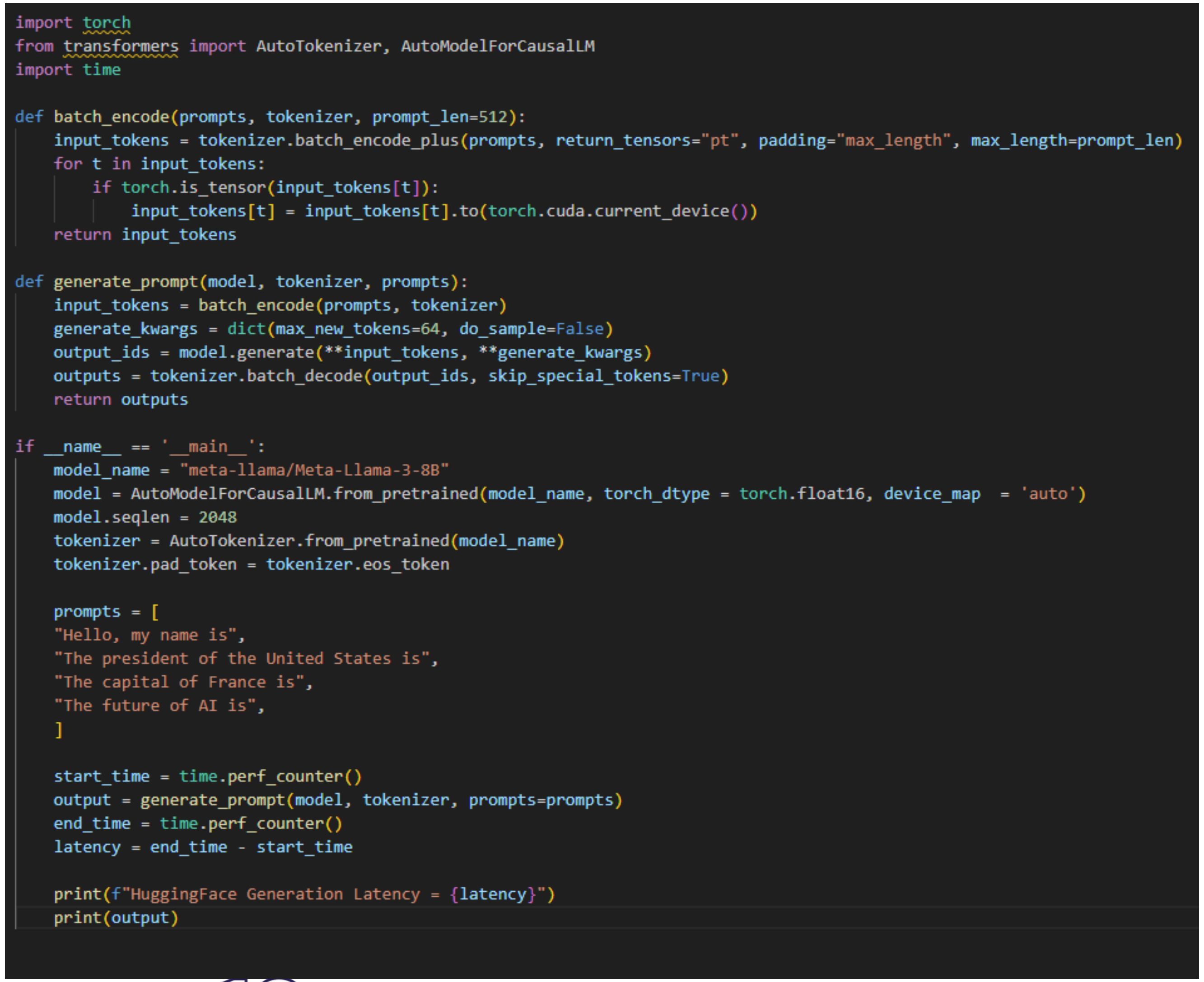

LLM Inference Bench Dashboard

Instructions

- Configuration: Use the configuration dropdown to load a different configuration to plot.

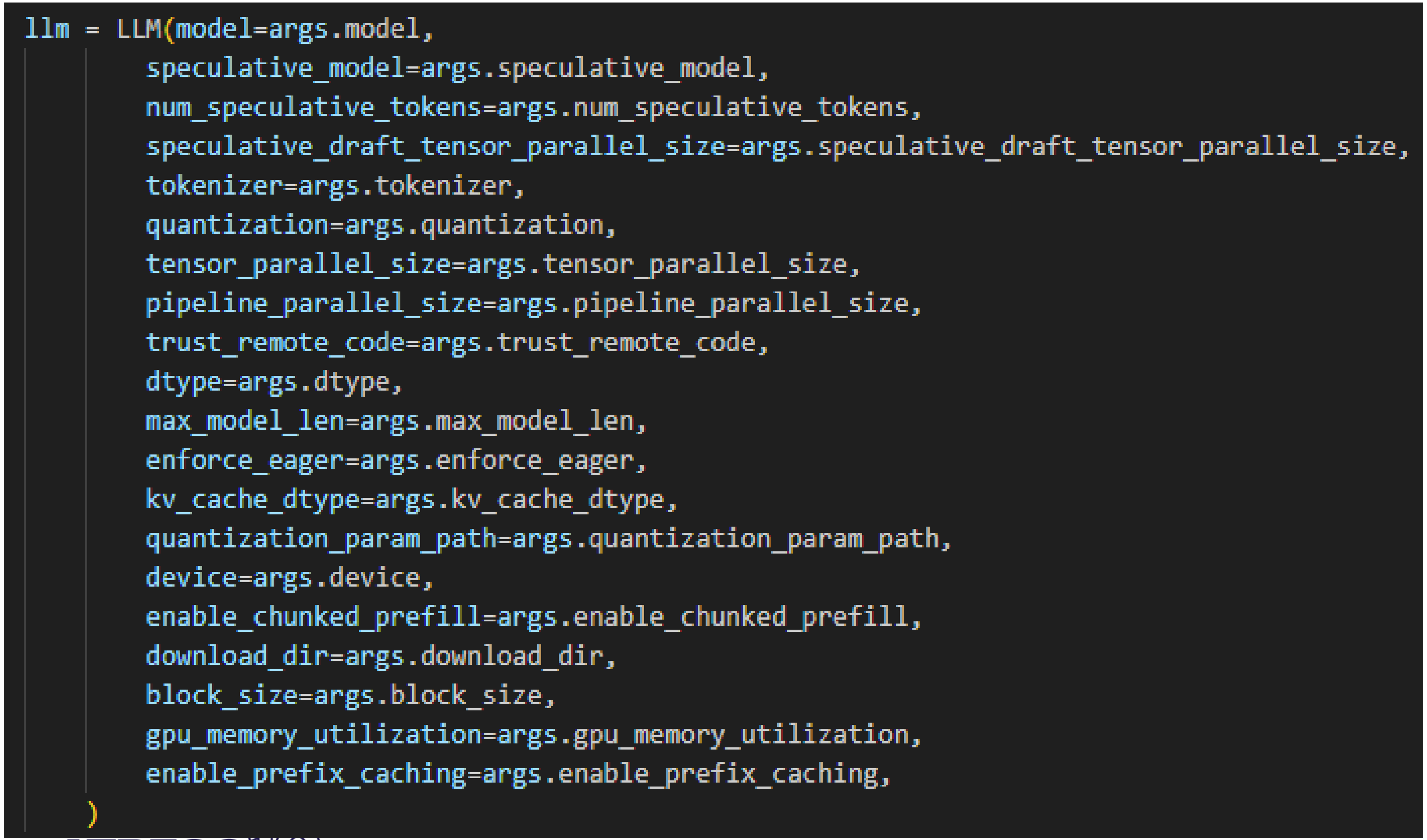
https://argonne-lcf.github.io/LLM-Inference-Bench/

Hugging Face Inference

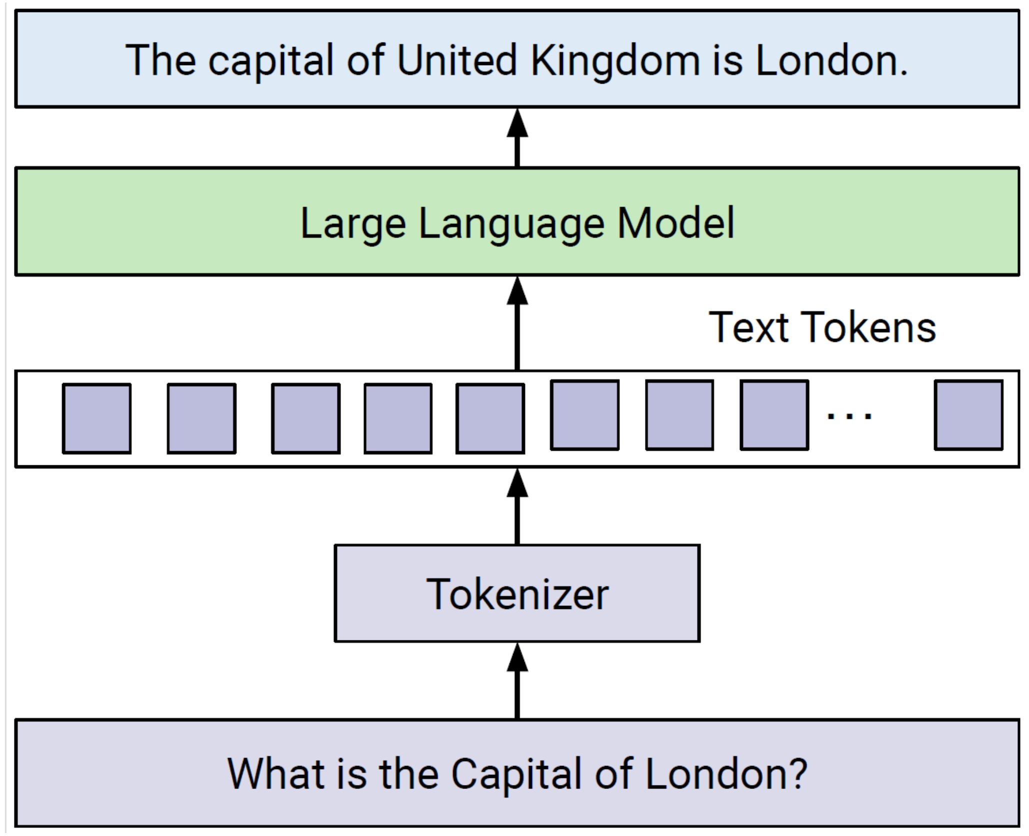


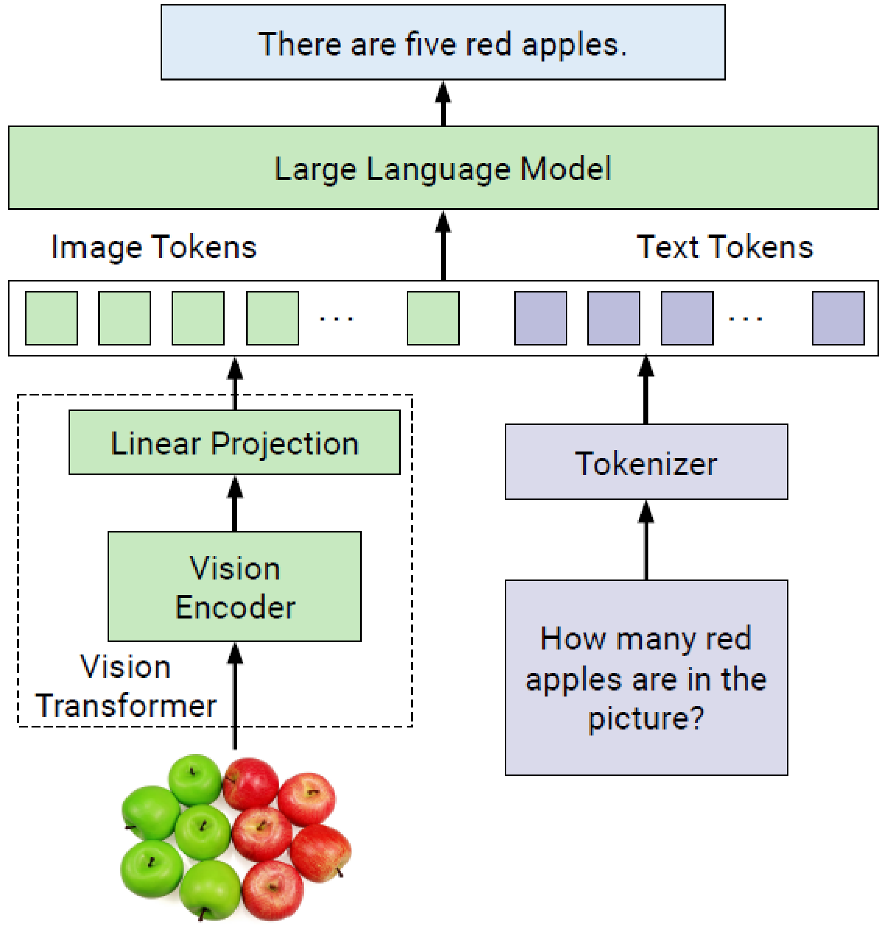
```python
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM
import time

def batch_encode(prompts, tokenizer, prompt_len=512):
    input_tokens = tokenizer.batch_encode_plus(prompts, return_tensors="pt", padding="max_length", max_length=prompt_len)
    for t in input_tokens:
        if torch.is_tensor(input_tokens[t]):
            input_tokens[t] = input_tokens[t].to(torch.cuda.current_device())
    return input_tokens

def generate_prompt(model, tokenizer, prompts):
    input_tokens = batch_encode(prompts, tokenizer)
    generate_kwargs = dict(max_new_tokens=64, do_sample=False)
    output_ids = model.generate(**input_tokens, **generate_kwargs)
    outputs = tokenizer.batch_decode(output_ids, skip_special_tokens=True)
    return outputs

if __name__ == '__main__':
    model_name = "meta-llama/Meta-Llama-3-8B"
    model = AutoModelForCausalLM.from_pretrained(model_name, torch_dtype = torch.float16, device_map  = 'auto')
    model.seqlen = 2048
    tokenizer = AutoTokenizer.from_pretrained(model_name)
    tokenizer.pad_token = tokenizer.eos_token

    prompts = [
    "Hello, my name is",
    "The president of the United States is",
    "The capital of France is",
    "The future of AI is",
    ]

    start_time = time.perf_counter()
    output = generate_prompt(model, tokenizer, prompts=prompts)
    end_time = time.perf_counter()
    latency = end_time - start_time

    print(f"HuggingFace Generation Latency = {latency}")
    print(output)
```

vLLM Inference


```
llm = LLM(model=args.model,
          speculative_model=args.speculative_model,
          num_speculative_tokens=args.num_speculative_tokens,
          speculative_draft_tensor_parallel_size=args.speculative_draft_tensor_parallel_size,
          tokenizer=args.tokenizer,
          quantization=args.quantization,
          tensor_parallel_size=args.tensor_parallel_size,
          pipeline_parallel_size=args.pipeline_parallel_size,
          trust_remote_code=args.trust_remote_code,
          dtype=args.dtype,
          max_model_len=args.max_model_len,
          enforce_eager=args.enforce_eager,
          kv_cache_dtype=args.kv_cache_dtype,
          quantization_param_path=args.quantization_param_path,
          device=args.device,
          enable_chunked_prefill=args.enable_chunked_prefill,
          download_dir=args.download_dir,
          block_size=args.block_size,
          gpu_memory_utilization=args.gpu_memory_utilization,
          enable_prefix_caching=args.enable_prefix_caching,
      )
```

Future of LLMs – Multimodal Models


The capital of United Kingdom is London.
Large Language Model
Text Tokens
. . .
Tokenizer
What is the Capital of London?



Any Questions?

Hands-On Session

Instructions on Polaris:

https://github.com/argonne-lcf/ATPESC_MachineLearning/tree/master/05_LLM_inference

Instructions on Aurora:

https://docs.alcf.anl.gov/aurora/data-science/inference/vllm/

ARGONNE
ATPESC2025
EXTREME-SCALE COMPUTING

ARGONNE TRAINING PROGRAM ON EXTREME-SCALE COMPUTING

Produced by Argonne National Laboratory, a U.S. Department of Energy Laboratory managed by UChicagoArgonne, LLC under contract DE-AC02-06CH11357.

Special thanks to the National Energy Research Scientific Computing Center (NERSC) and Oak Ridge Leadership Computing Facility (OLCF) for the use of their resources during the training event.