

Module 3

Deployment Optimizations

Improving model size and speed



Learning Objectives

By the end of this module you will:

- Be able to make the design choices for your LLM development
- Create and design your own pseudo Mixture-of-Experts LLM system
- Develop an understanding of the utility of quantization in LLMs for both training and inferencing





Extra-Large Language Models

What if our models are too big?



The issue with high performance LLMs

Paying the price for quality

As models grow in size, they get “better” and “worse”.

- **Better** – accuracy, alignment, abilities
- **Worse** – speed, memory footprint, updatability

small fast low quality model

Large...slow...high...quality...model.



What if we could improve the speed and footprint while preserving quality?





Improving Learning Efficiency

How can we train and fine-tune better?






How we interact with LLMs

The importance of context length

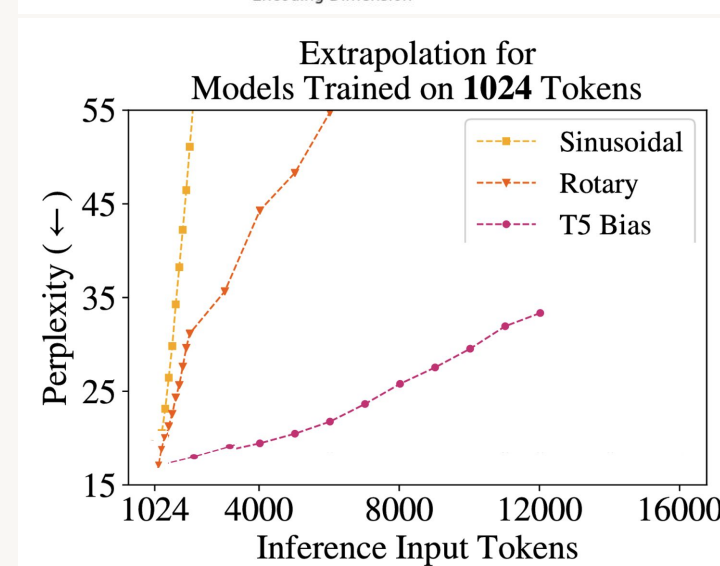
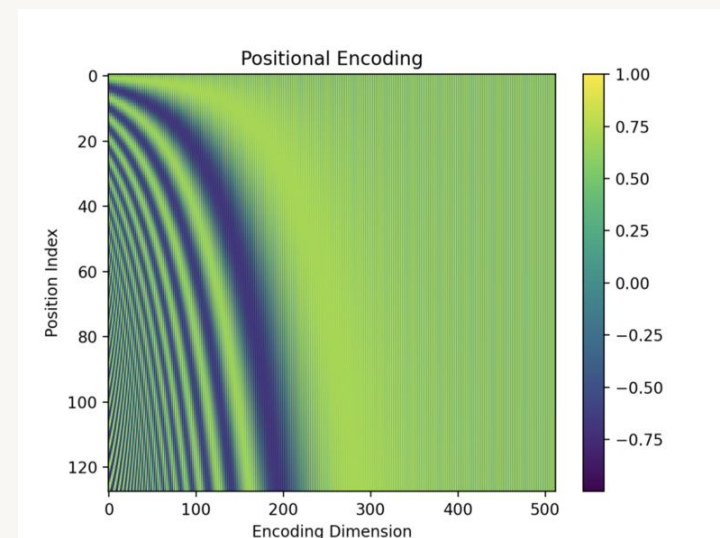
LLMs, like us, do better at tasks with more context.

This means longer input/context length.

- Computing input embeddings  linearly
- Performing FFNN calculations  linearly
- Calculating attention scores  **quadratically**

Even worse:

Attention cannot perform as well on longer context than it was trained on.



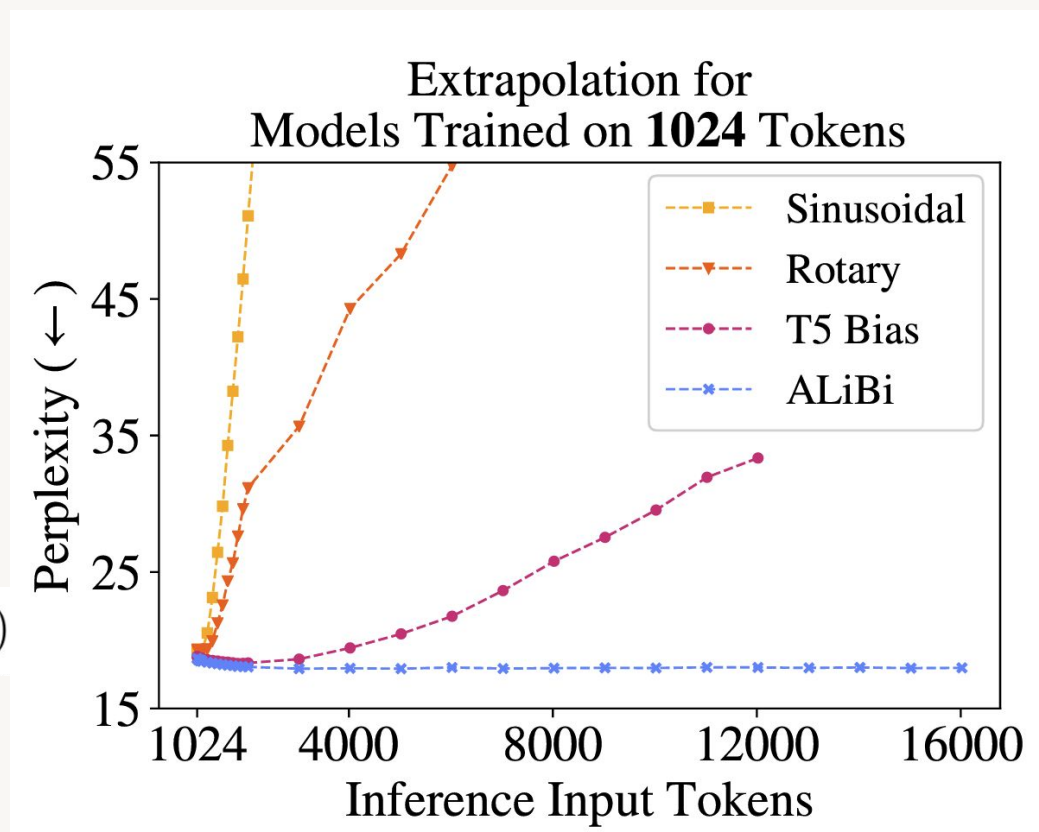
Training short but inference long

You'll need a good Alibi for this one

Attention is all you need... to fix! And just with a linear bias.

$$\begin{bmatrix} q_1 \cdot k_1 & & & & \\ q_2 \cdot k_1 & q_2 \cdot k_2 & & & \\ q_3 \cdot k_1 & q_3 \cdot k_2 & q_3 \cdot k_3 & & \\ q_4 \cdot k_1 & q_4 \cdot k_2 & q_4 \cdot k_3 & q_4 \cdot k_4 & \\ q_5 \cdot k_1 & q_5 \cdot k_2 & q_5 \cdot k_3 & q_5 \cdot k_4 & q_5 \cdot k_5 \end{bmatrix} + \begin{bmatrix} 0 & & & & \\ -1 & 0 & & & \\ -2 & -1 & 0 & & \\ -3 & -2 & -1 & 0 & \\ -4 & -3 & -2 & -1 & 0 \end{bmatrix} \cdot m$$

$$\text{softmax}(\mathbf{q}_i \mathbf{K}^\top) \rightarrow \text{softmax}(\mathbf{q}_i \mathbf{K}^\top + m \cdot [-(i-1), \dots, -2, -1, 0])$$



Faster calculations

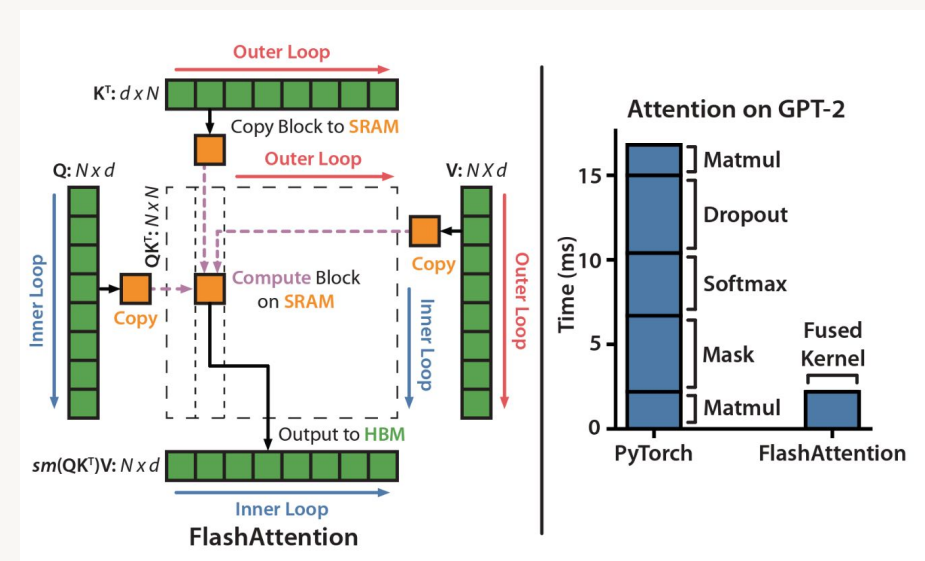
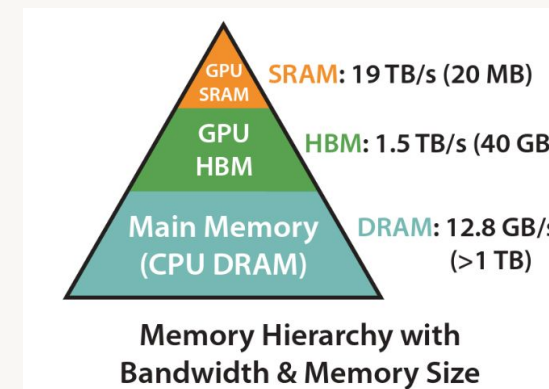
Calculating attention in a flash.

Observation:

- Fastest memory in the GPU is SRAM
- Longer context = larger attention matrices

Problem:

- SRAM is small relative to the attention matrix needed in calculations
- Solution: Flash Attention!
- Attention compute operations are redone, no matrix created!
- More time spent in SRAM, massive performance boost



Source: [Flash Attention](#)



Many queries, fewer keys

Multi-query and Grouped-query Attention

- Multi-Headed Attention

#Queries = #Keys = #Values

Each head can focus on different parts of language.

Inference – slow, accurate.

- Multi-Query Attention

Many Queries, 1 Key, 1 Value

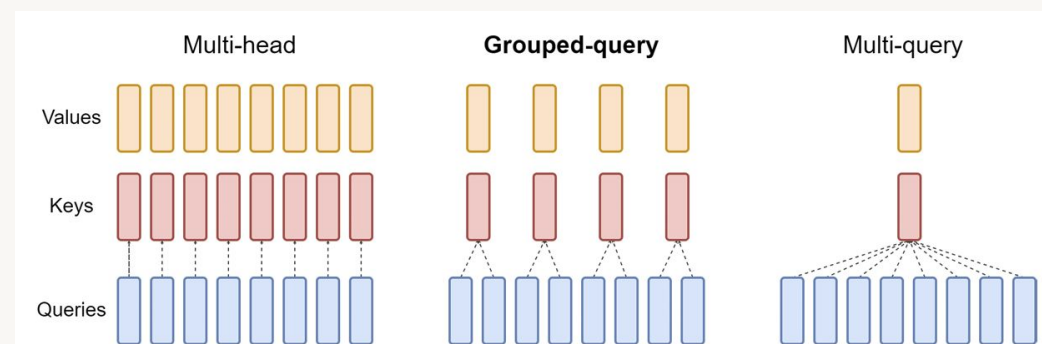
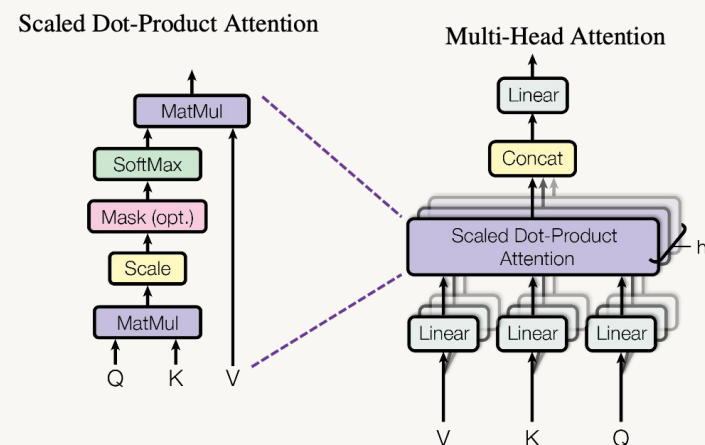
Forcing the model to use different queries

Inference – fast, inaccurate.

- Grouped-Query Attention

#Queries = #/n Keys = #/n Values

Inference – fast, accurate.



Source: [Grouped Query Attention](#)





Improving Model Footprint

Doing more with less



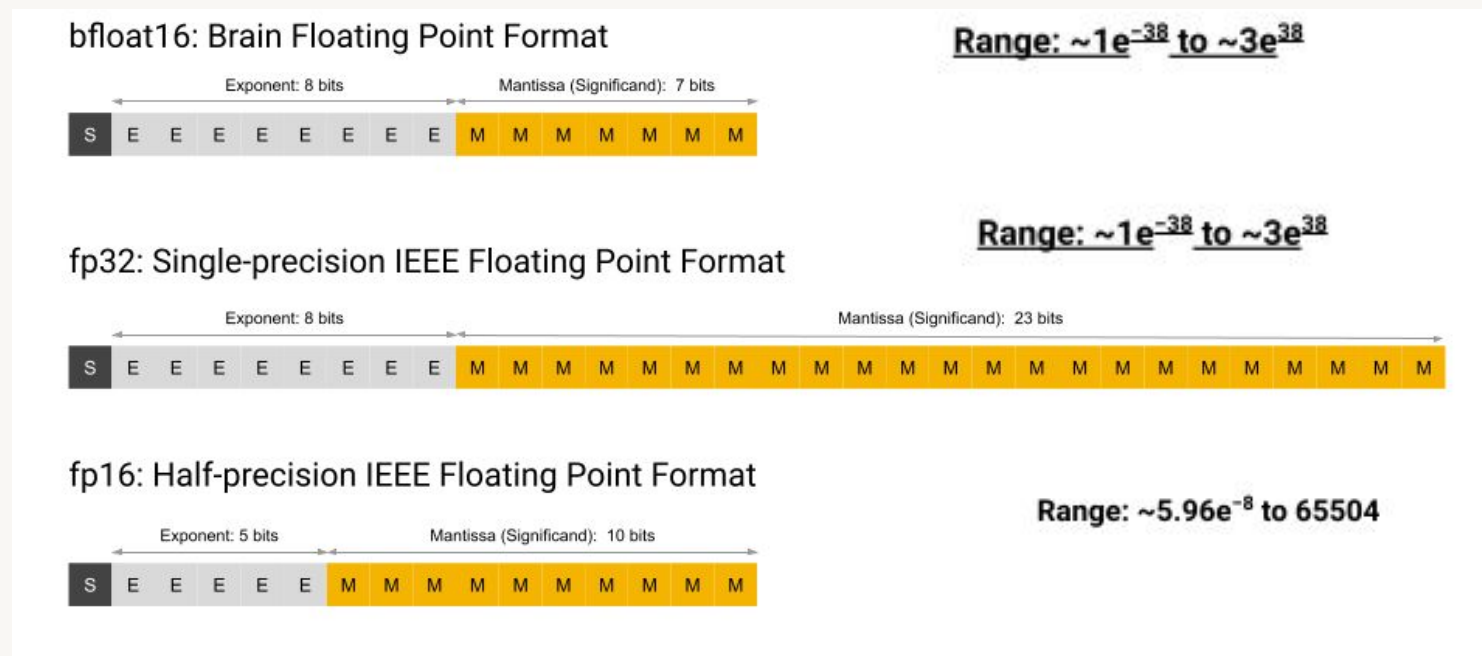
Storing numbers

Billions of parameters. Each a floating point number.

FP16 FP32 – IEEE standards.

Google Brain saw this and created the BF16

- Same range as FP32
- Same size as FP16



Source: [Google bfloat16](#)

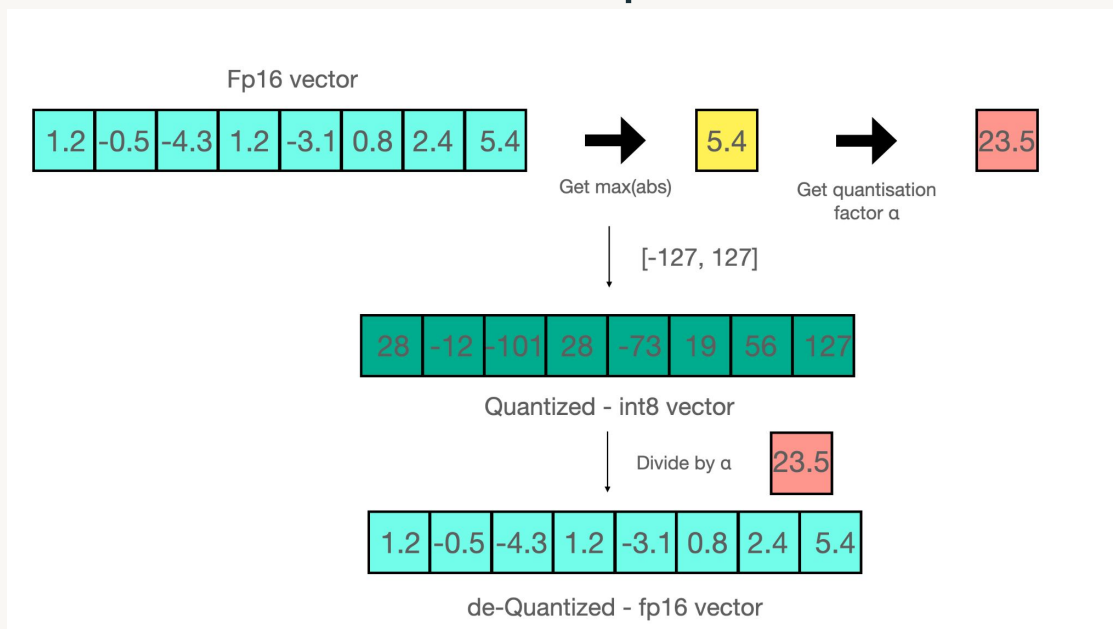
What if we need to go even further?



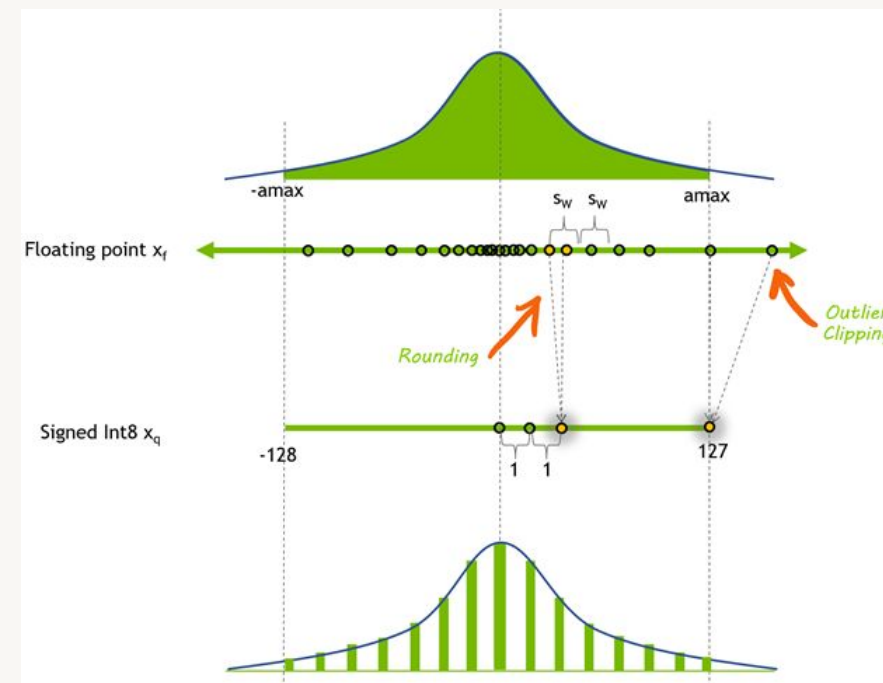
Quantization

Do we need so much precision?

Approximate the values in quantized forms



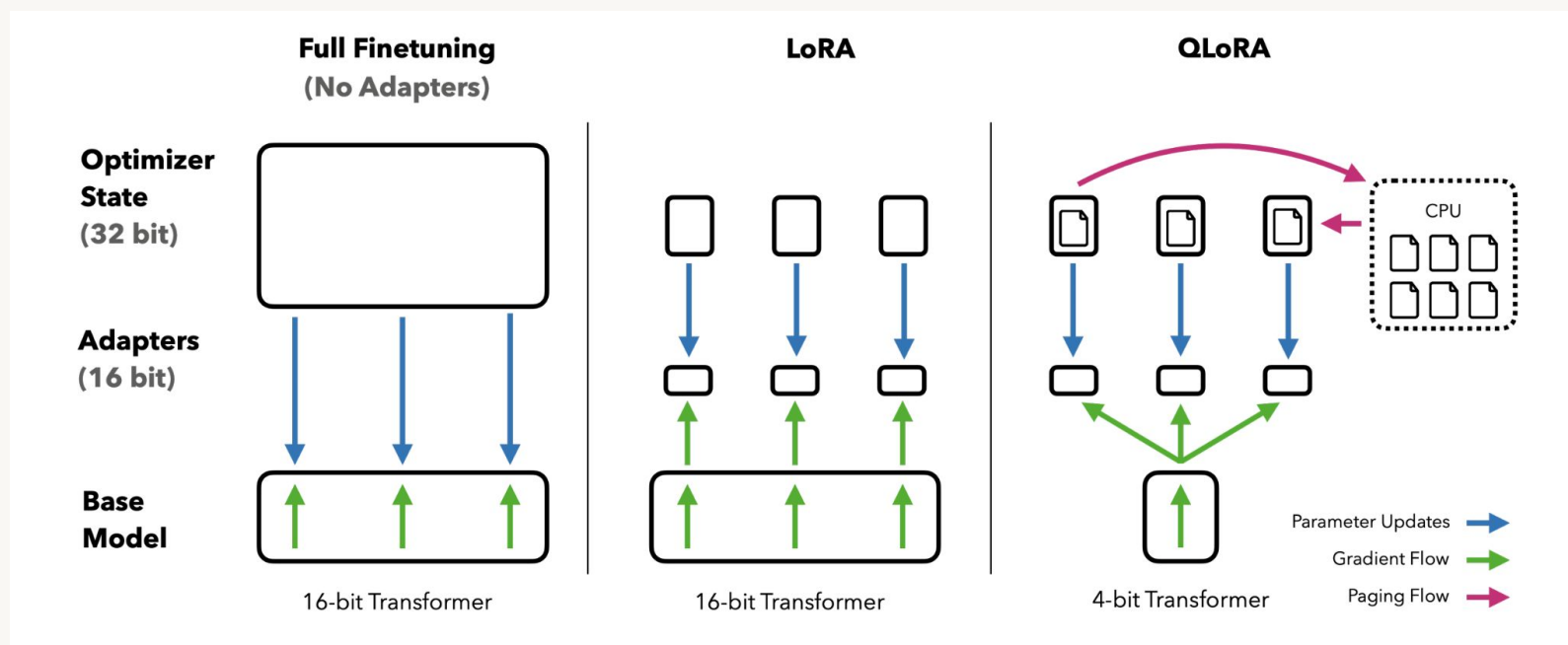
Create quantized functions



QLoRA

Applying quantization to fine tuning

- LoRA was already great!
- QLoRA adds even more:
 - 4-bit quantization
 - Paged optimization



Source: [QLoRA](#)





Multi-LLM Inferencing

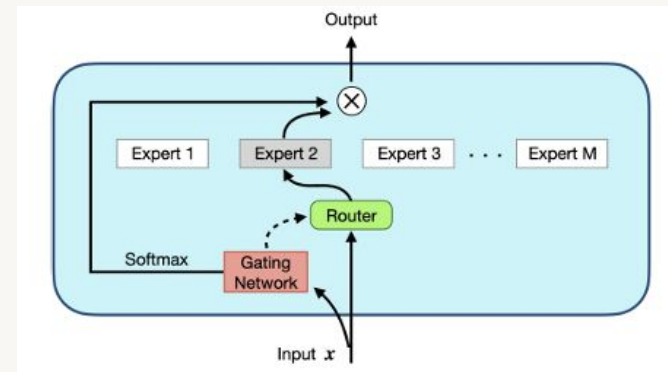
Hybrid and Ensemble-based systems

Mixture-of-Experts

A trillion parameters, for a fraction of the training

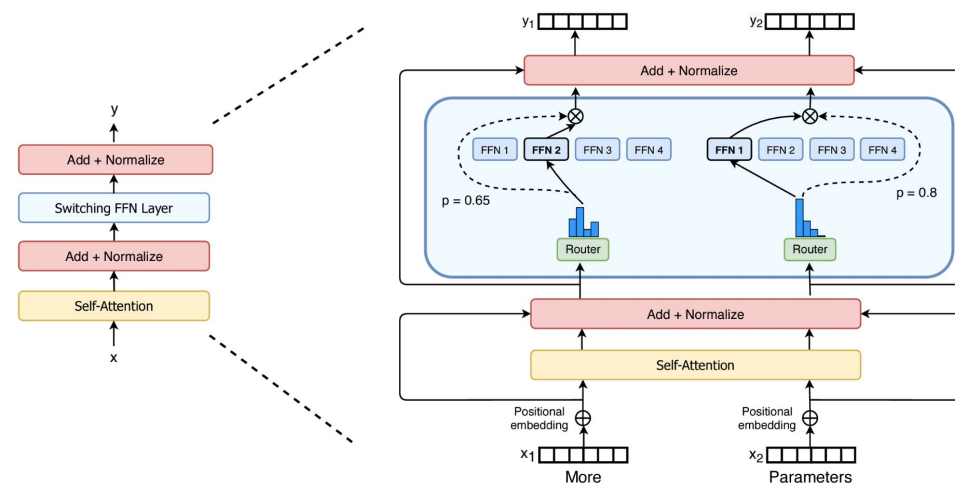
Mixture-of-Experts (MoE):

- Input is sent to a router
- Multiple NNs are trained



Switch Transformer:

- Application of MoE
- Position-wise FFNN are multiplied
- Single attention network

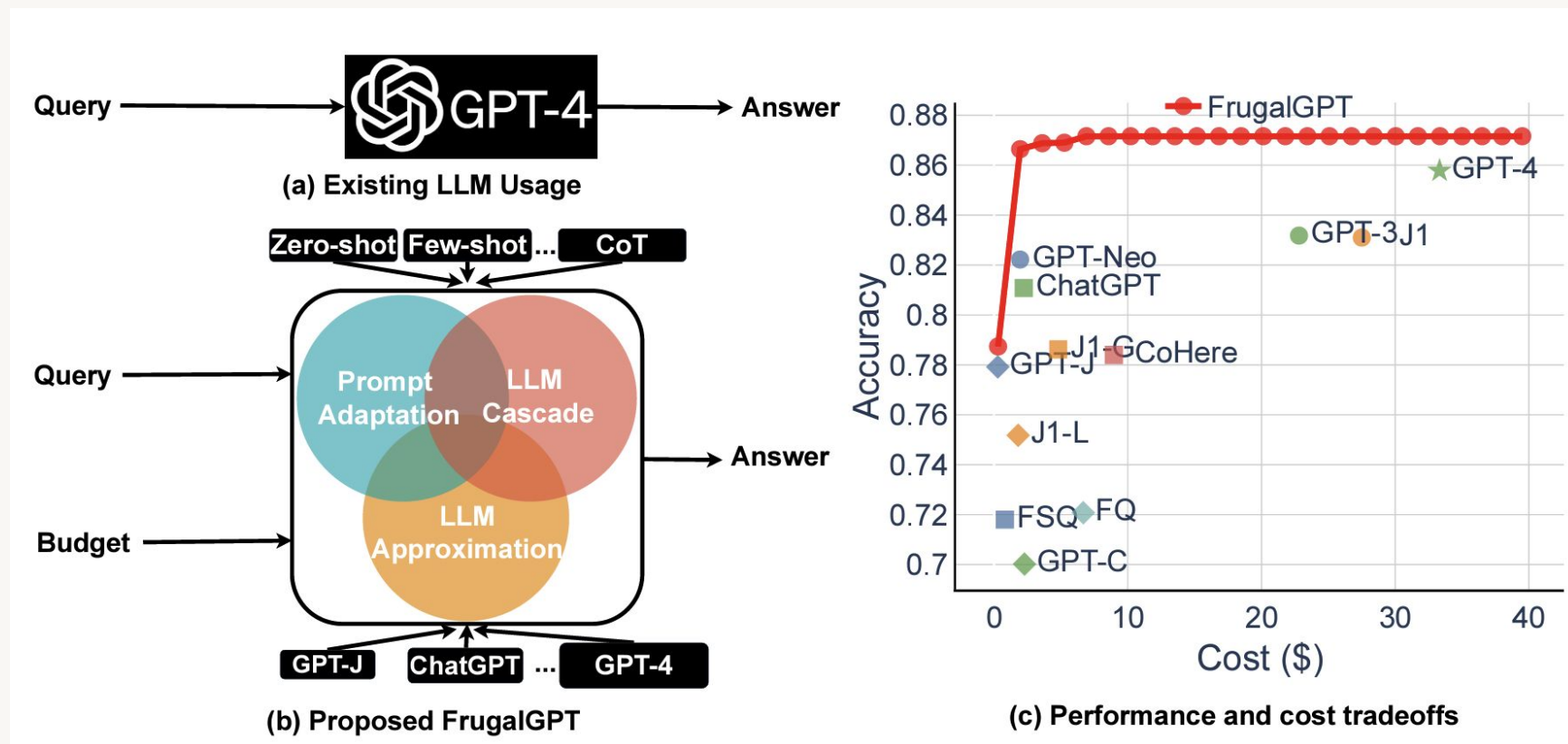


LLM Cascades and FrugalGPT

Improving our resource allocation in LLM inferencing

LLM cascade:

- Send prompts to smallest models
- ↓
- Gather confidence of response
- ↓
- If too small, move to a larger model



Source: [FrugalGPT](#)





Current Best Practices

If you want to build now, do it right



Best Practices

Training from scratch:

- ALiBi
- Flash Attention
- Grouped-Query Attention
- Mixture-of-Experts

Fine-tuning/Inferencing:

- LoRA/QLoRA
- FrugalGPT

Numbers ~~100~~ every LLM Developer should know *

Prompts

40-90% Amount saved by appending "Be Concise" to your prompt

1.3 Average tokens per word

Training and Fine Tuning

~\$1 million Cost to train a 13 billion parameter model on 1.4 trillion tokens

<0.001 Cost ratio of fine tuning vs training from scratch

Price

~50 Cost Ratio of GPT-4 to GPT-3.5 Turbo

5 Cost Ratio of generation of text using GPT-3.5-Turbo vs OpenAI embedding

10 Cost Ratio of OpenAI embedding to Self-Hosted embedding

6 Cost Ratio of OpenAI base vs fine tuned model queries

1 Cost Ratio of Self-Hosted base vs fine-tuned model queries

GPU Memory

16GB V100 GRAM capacity
24GB A10G GRAM capacity
40/80GB A100 GRAM capacity

2x number of parameters Typical GPU memory requirements of an LLM for serving

~1GB Typical GPU memory requirements of an embedding model

>10x Throughput improvement from batching LLM requests

1 MB GPU Memory required for 1 token of output with a 13B parameter model

* Check out bit.ly/llm-dev-numbers for how we calculated the numbers

Presented by  RAY &  anyscale with  Join the community ray.io or Request a Trial anyscale.com/signup today



Module Summary

Deployment Optimizations – What have we learned?

- LLMs are currently outpacing modern compute capacity, necessitating the development of work around solutions and approaches
- Modifying the original approach to attention has allowed for longer contexts, better use of hardware, more efficient calculations
- Quantization helps to store and use massive LLMs on smaller hardware
- Combining LLM inferences of different models allows an effective scale up of parameters with minimal cost changes



Time for some code!

