# 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

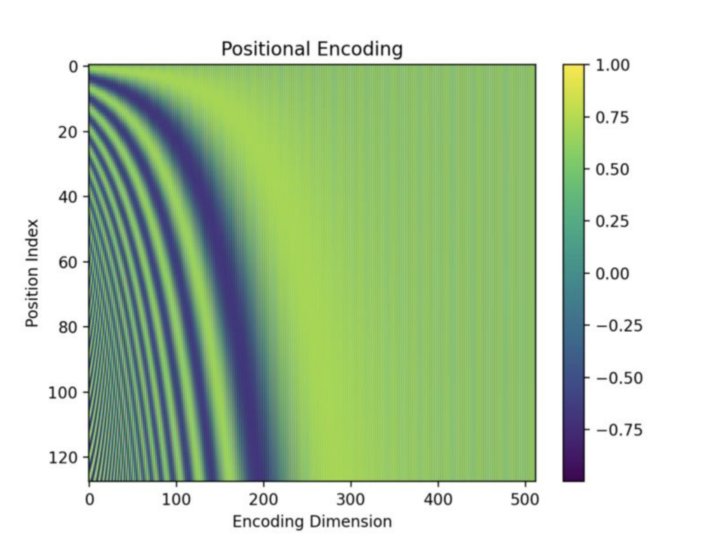
LLMs, like us, do better at tasks with more context.

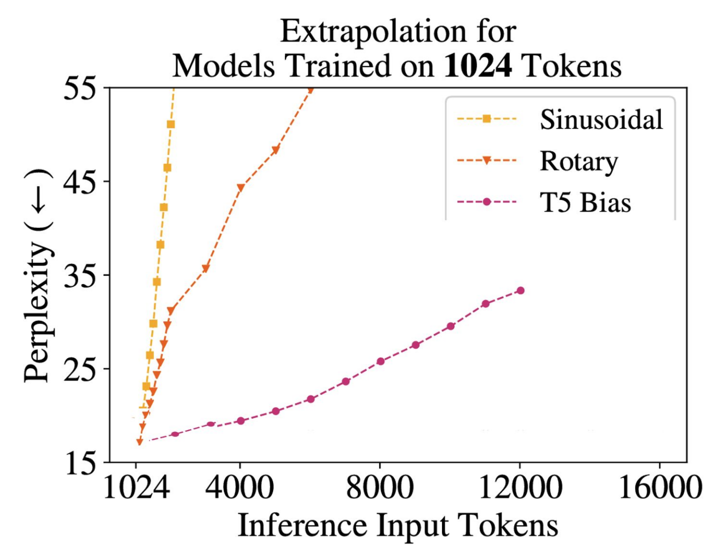
This means longer input/context length.

- Computing input embeddings ⬆ linearly
- Performing FFNN calculations ⬆ linearly
- Calculating attention scores ⬆⬆ **<u>quadratically</u>**

Even worse:

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



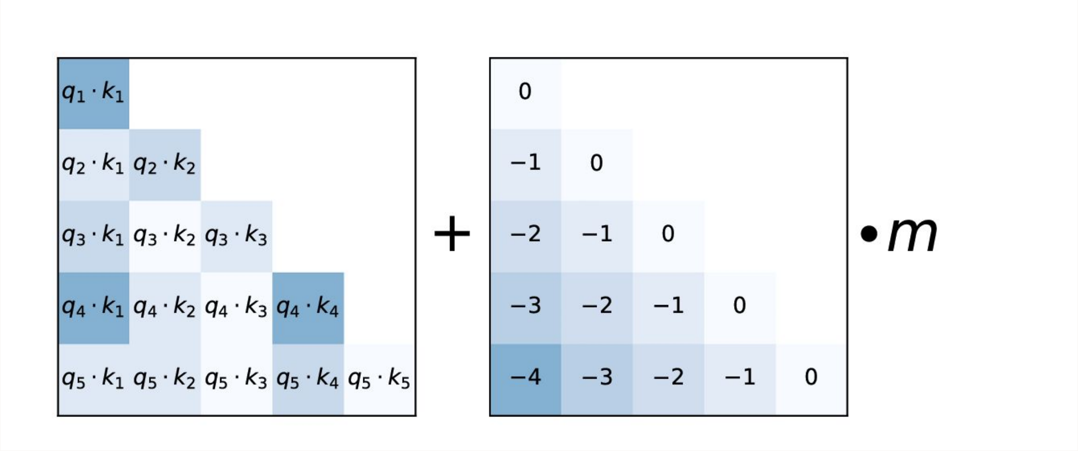




Source: ALiBi

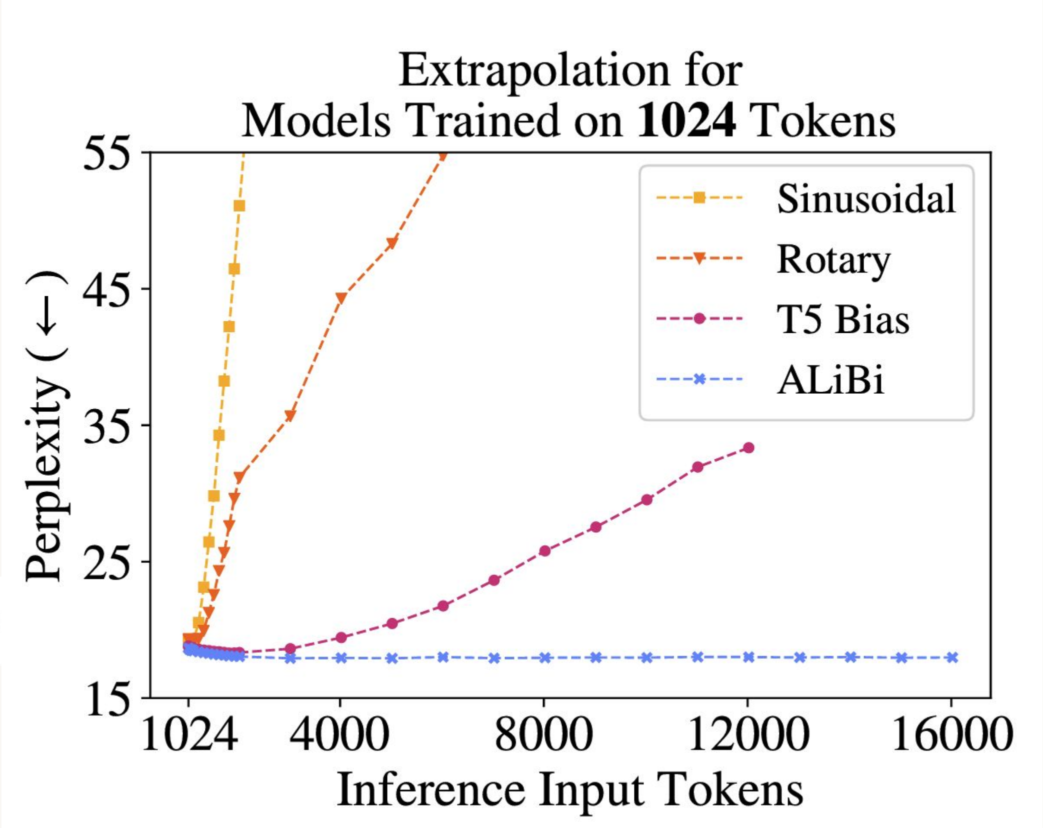
# Training short but inference long

## You'll need a good Alibi for this one

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




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






Source: ALiBi

# Faster calculations

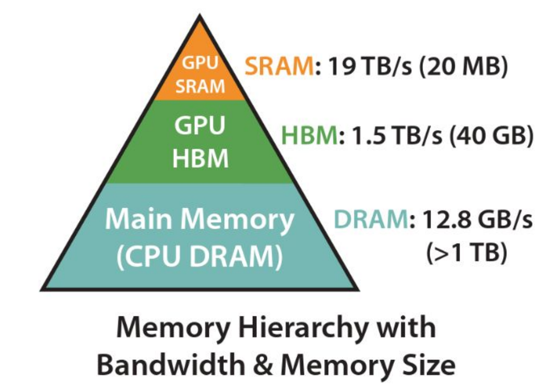
## Calculating attention in a flash.

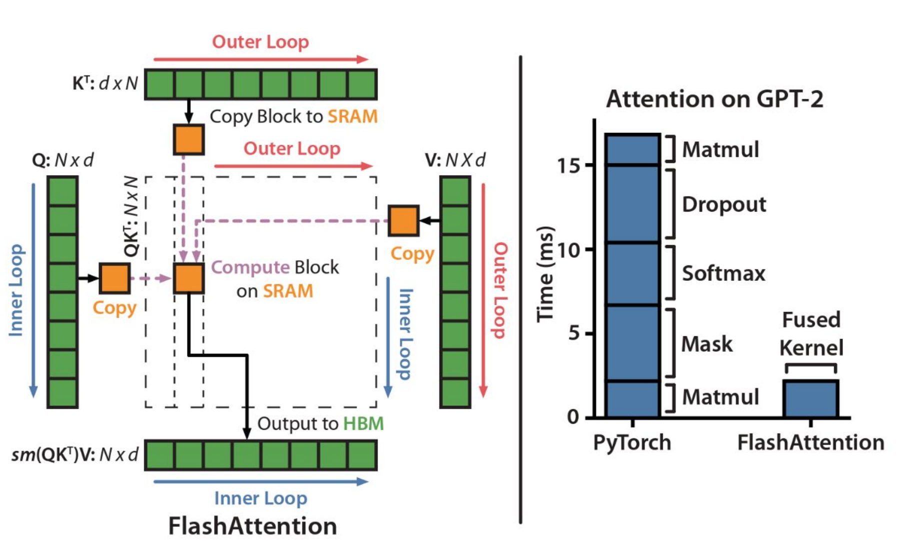
Observation:

- 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

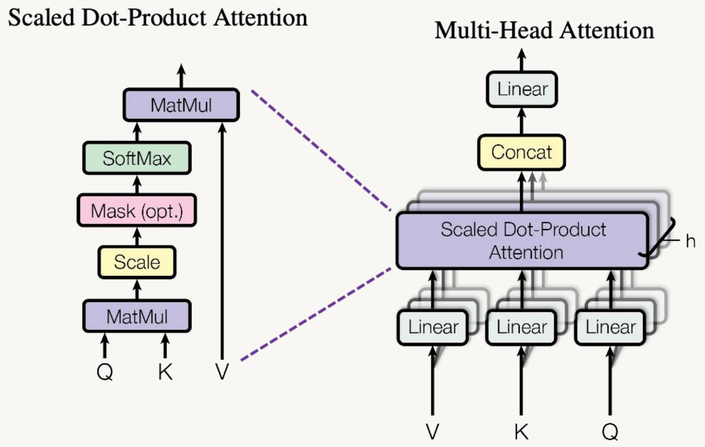
  Many Queries, 1 Key, 1 Value
  Forcing the model to use different queries

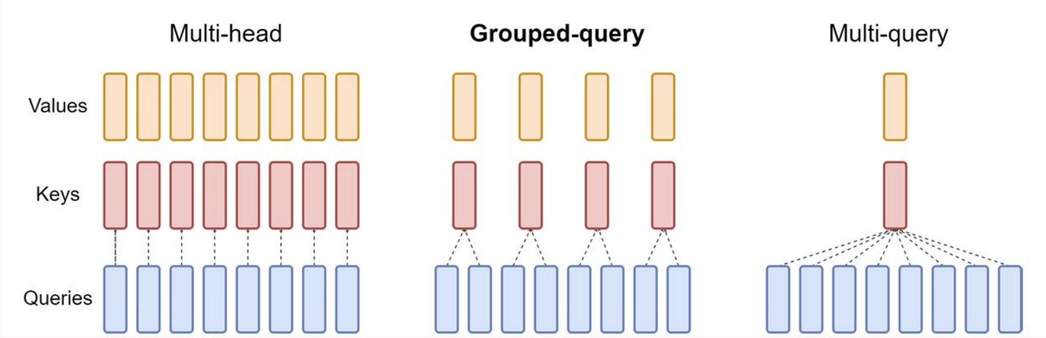
  **Inference – fast, inaccurate.**

- Grouped-Query Attention

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

  **Inference – fast, accurate.**




Source: Grouped Query Attention

# Improving Model Footprint

## Doing more with less

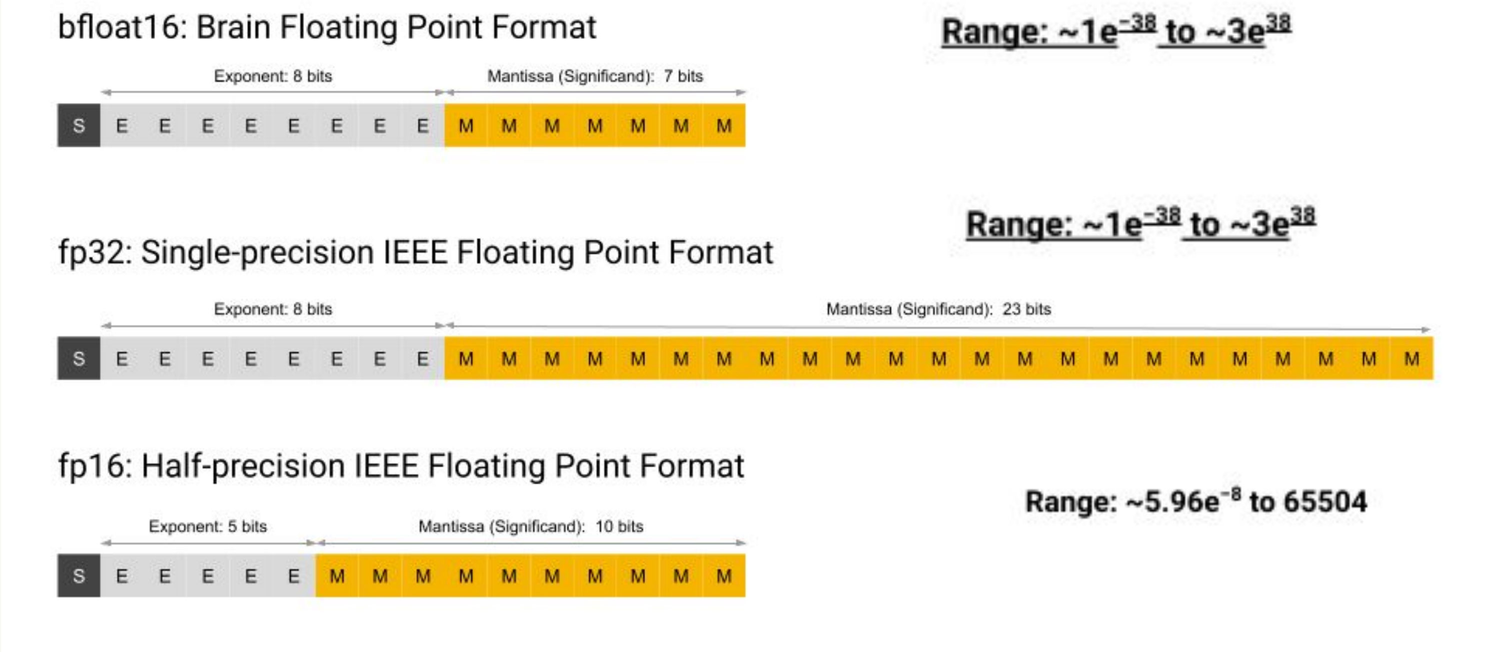
# Storing numbers

## 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



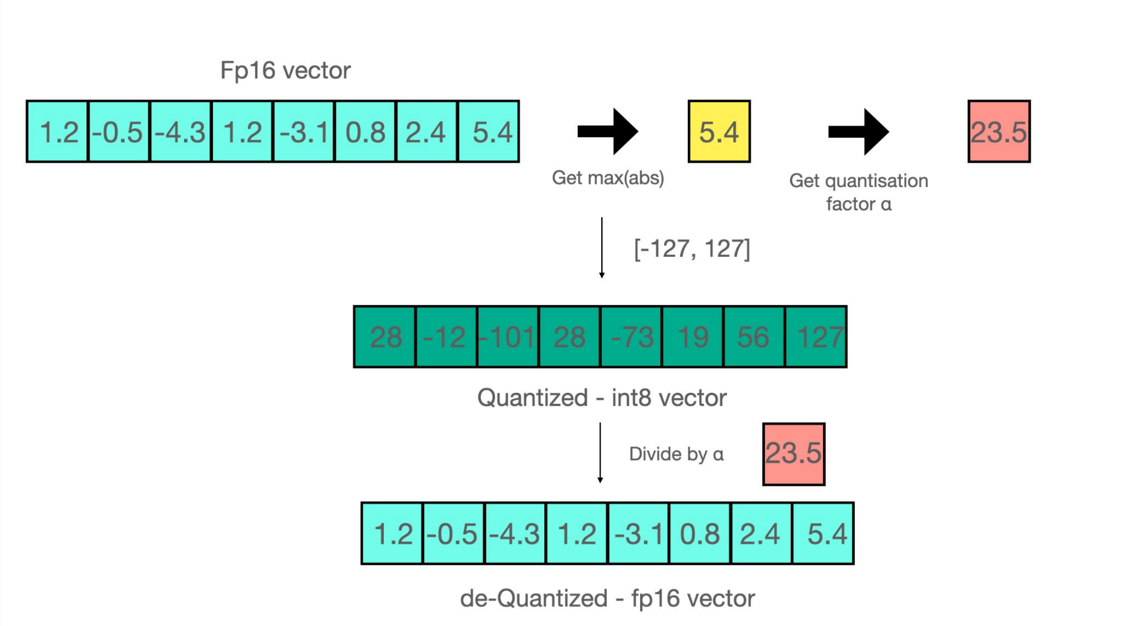

Source: Google bfloat16

What if we need to go even further?

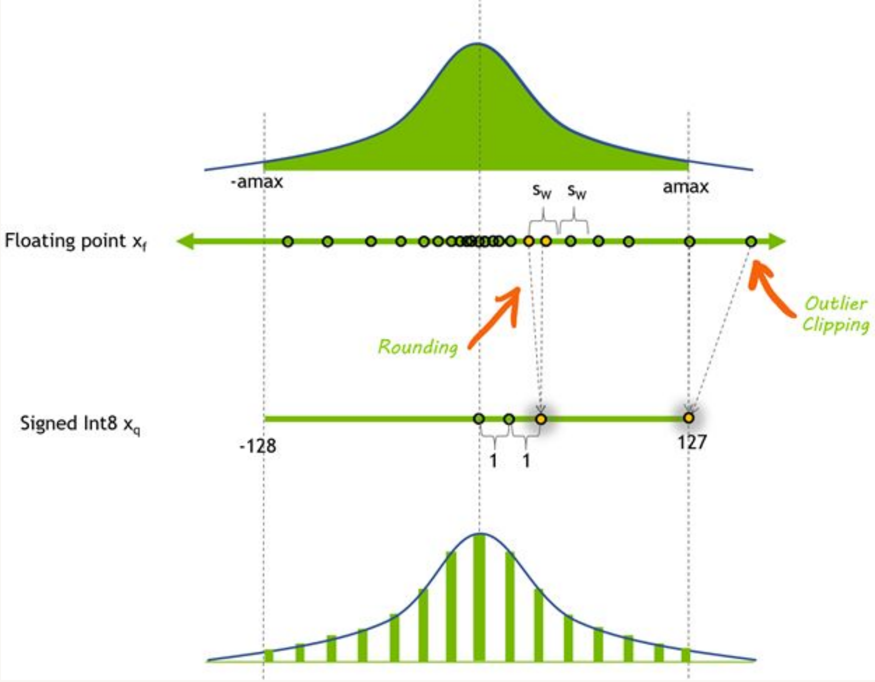
# Quantization

## Do we need so much precision?

## Approximate the values in quantized forms




## Create quantized functions

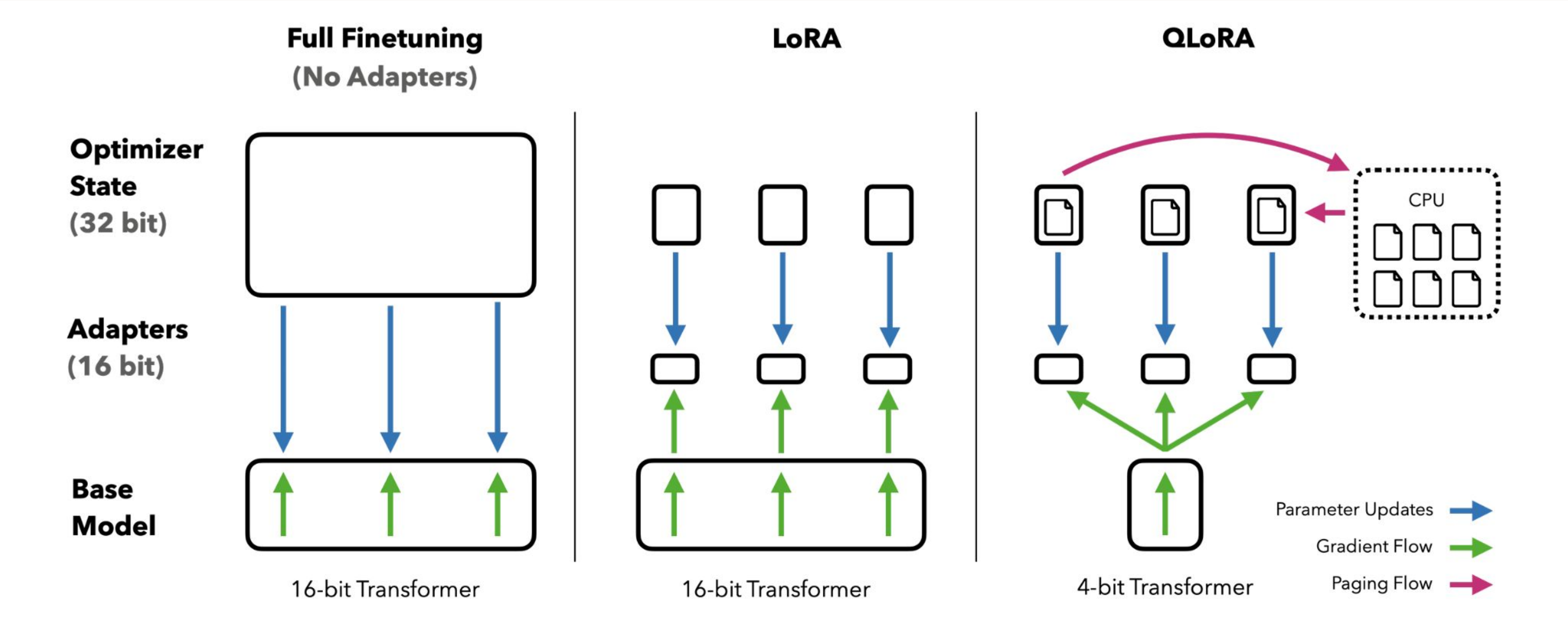



Source: LLM.int8()

# QLoRA

## Applying quantization to fine tuning

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




Source: QLoRA

# Multi-LLM Inferencing

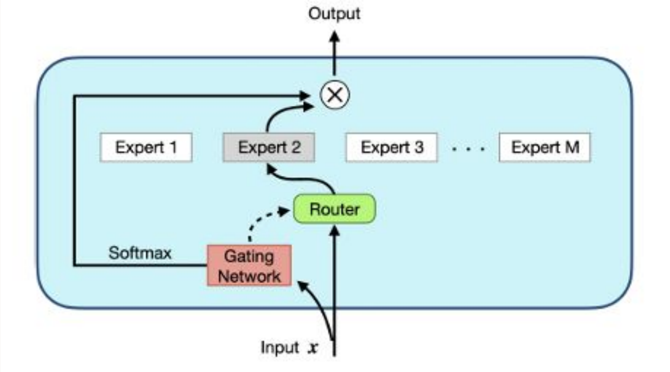
## Hybrid and Ensemble-based systems

# Mixture-of-Experts

## A trillion parameters, for a fraction of the training

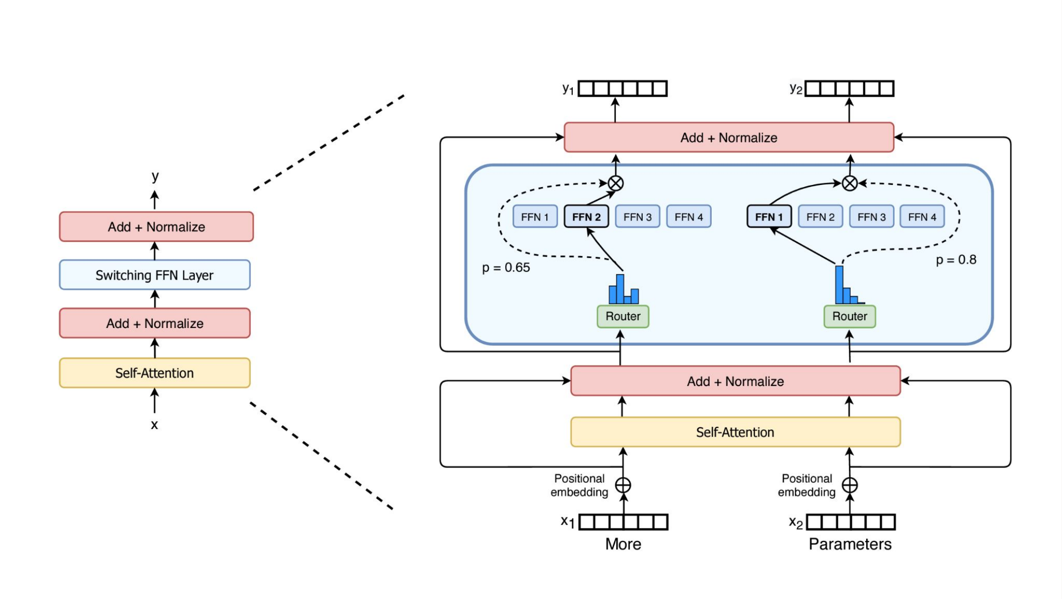
Mixture-of-Experts (MoE):

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




Switch Transformer:

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




Source: Switch Transformer

# LLM Cascades and FrugalGPT

## Improving our resource allocation in LLM inferencing

LLM cascade:

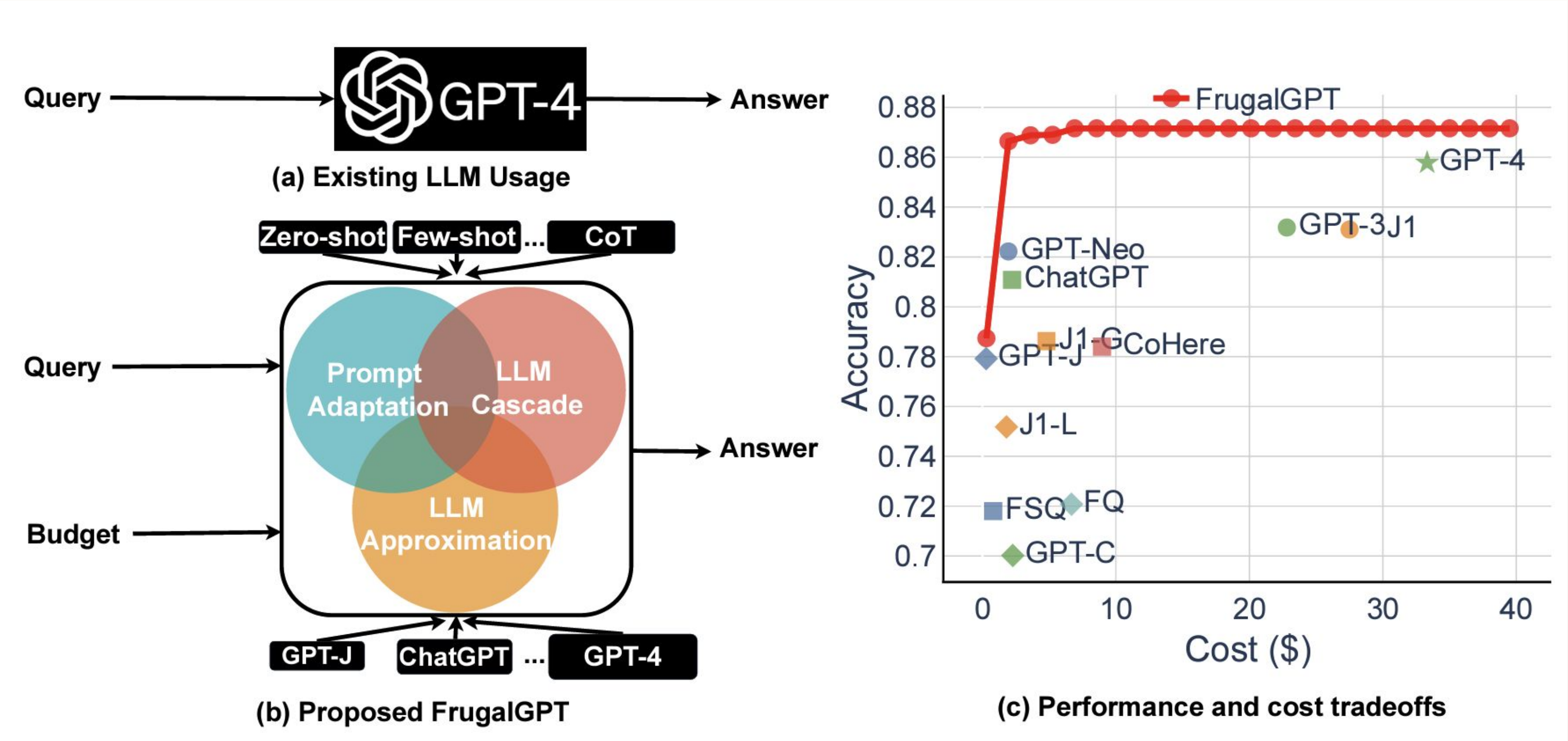
- Send prompts to smallest models

⬇

- 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

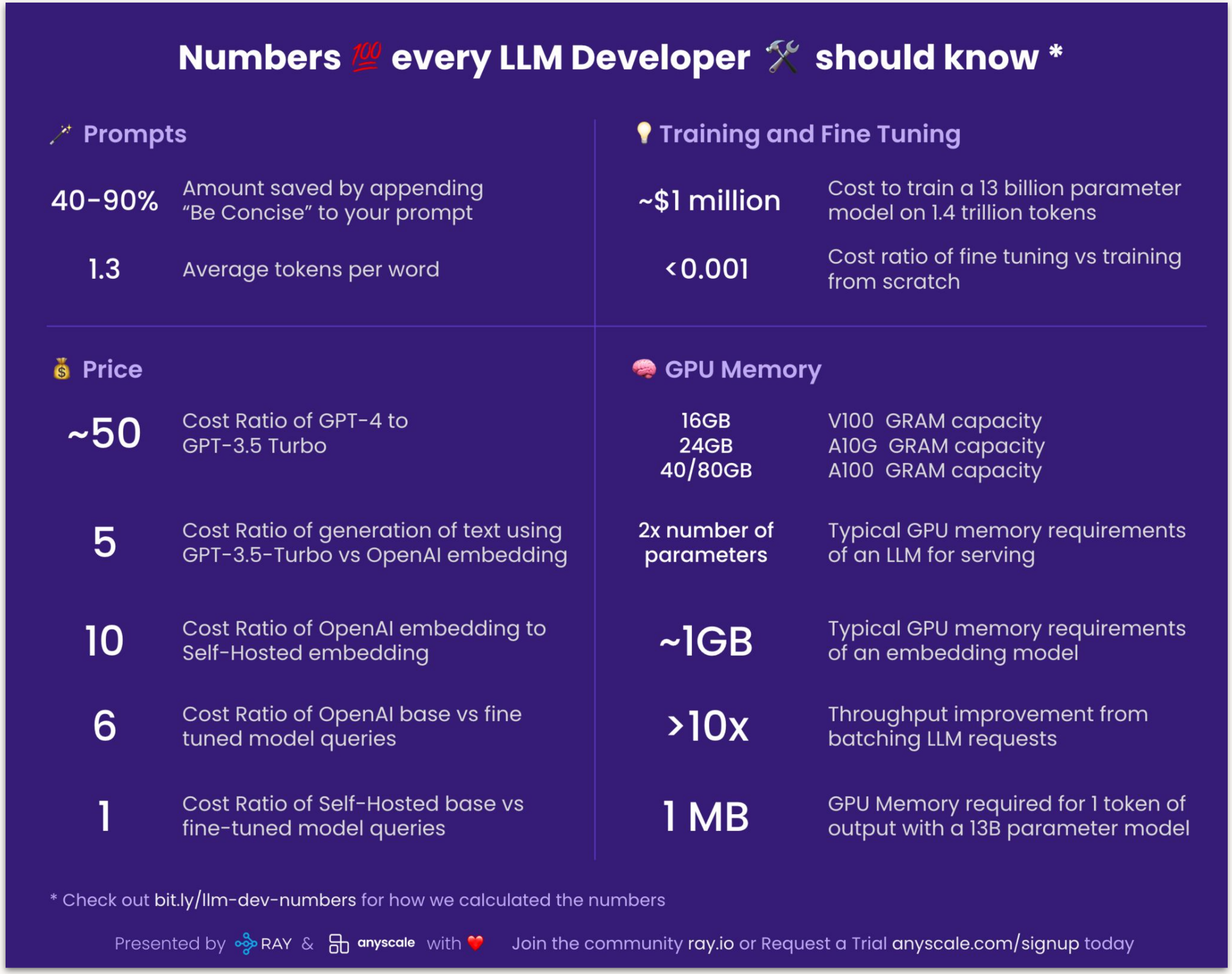
# Best Practices

Training from scratch:

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

Fine-tuning/Inferencing:

- LoRA/QLoRA
- FrugalGPT




Source: LLM-Numbers

# 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
- Combing LLM inferences of different models allows an effective scale up of parameters with minimal cost changes

# Time for some code!