

# Fine tuning LLMs,



Insop, 11/10/2023

# Take aways

- Why do we fine-tune
- How to fine-tune
- Things to consider

# Outlines

## LLM Overview

Why consider your own LLMs

Which LLMs to choose

Train your LLMs

Preparing Dataset for training

Evaluating LLMs

How to serve your own LLMs

# LLM

API based LLM access

- Proprietary and Open source LLMs

Approaches:

- Zero-shot, Few-shot prompt
- Crafted prompt
- Retrieval augmented generation

\* how to use llm talk



# ANTHROPIC



Meta Llama2, 



# Outlines

LLM Overview

? Why consider your own LLMs

Which LLMs to choose

Train your LLMs

Preparing Dataset for training

Evaluating LLMs

How to serve your own LLMs

# Why consider custom LLMs

Data residency and privacy

Intellectual property

Customization

Cost

Performance

Latency

Control over updates

Ownership

Transparency

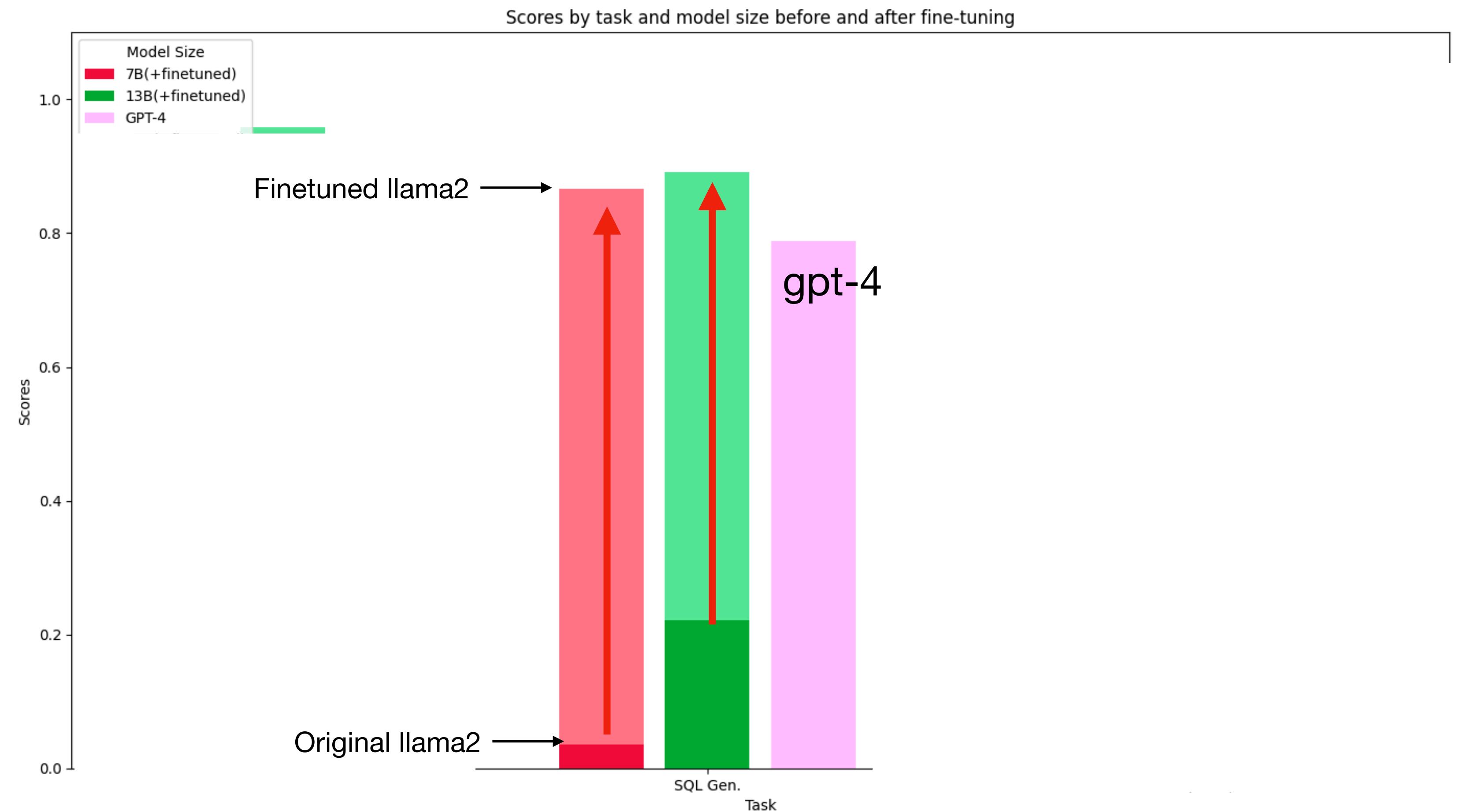
# Performance of Fine-tuned LLM

- How is the performance of fine-tuned LLM?
  - *It depends*
  - For a specific task, fine-tuning open source LLM can improve the performance. It could surpass closed models' capability
  - Depends on the complexity of task, degree of improvement can vary

# OSS LLM Fine-tuning



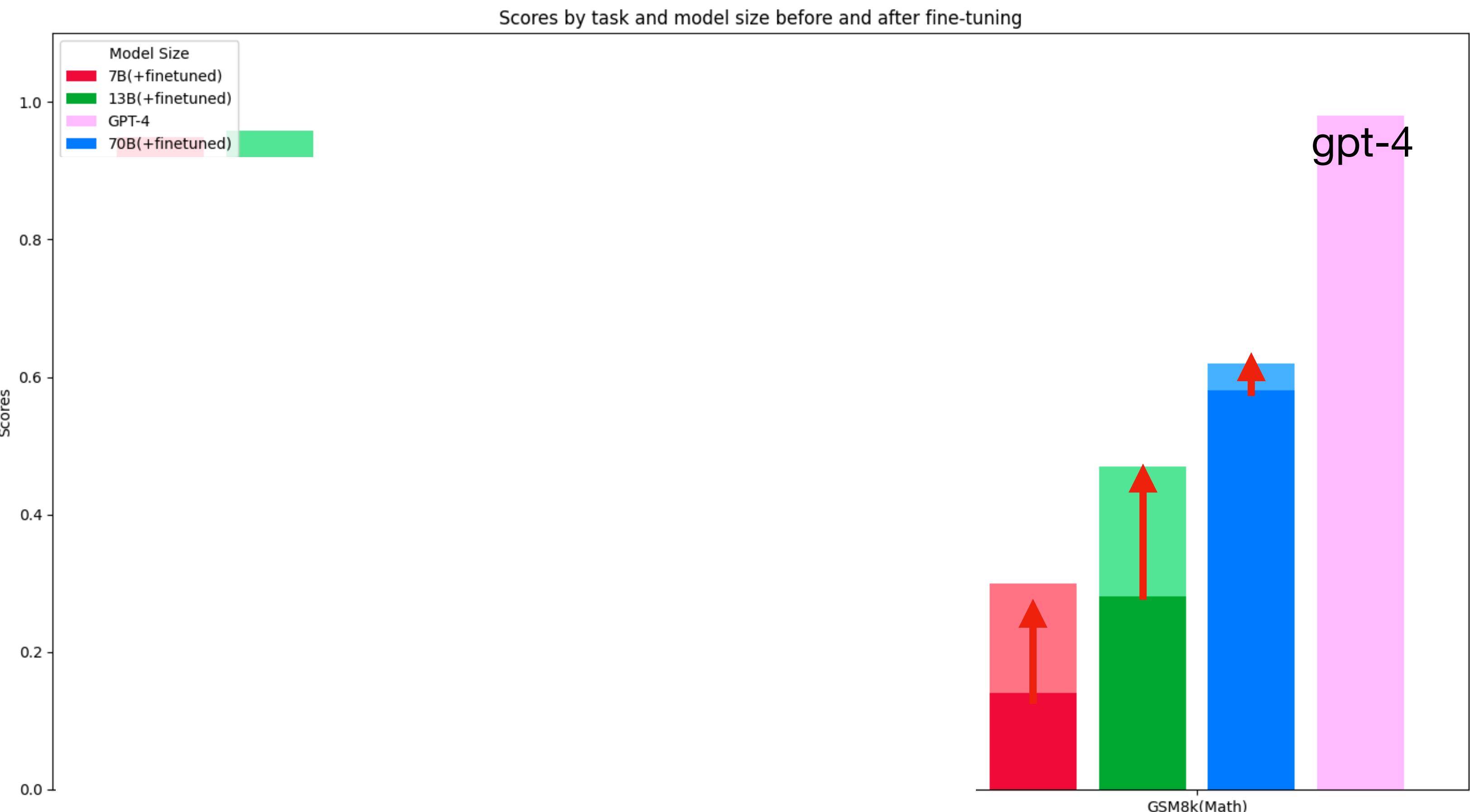
- Finetuned llama2 outperform gpt-4 on **SQL generation** task



# OSS LLM Fine-tuning



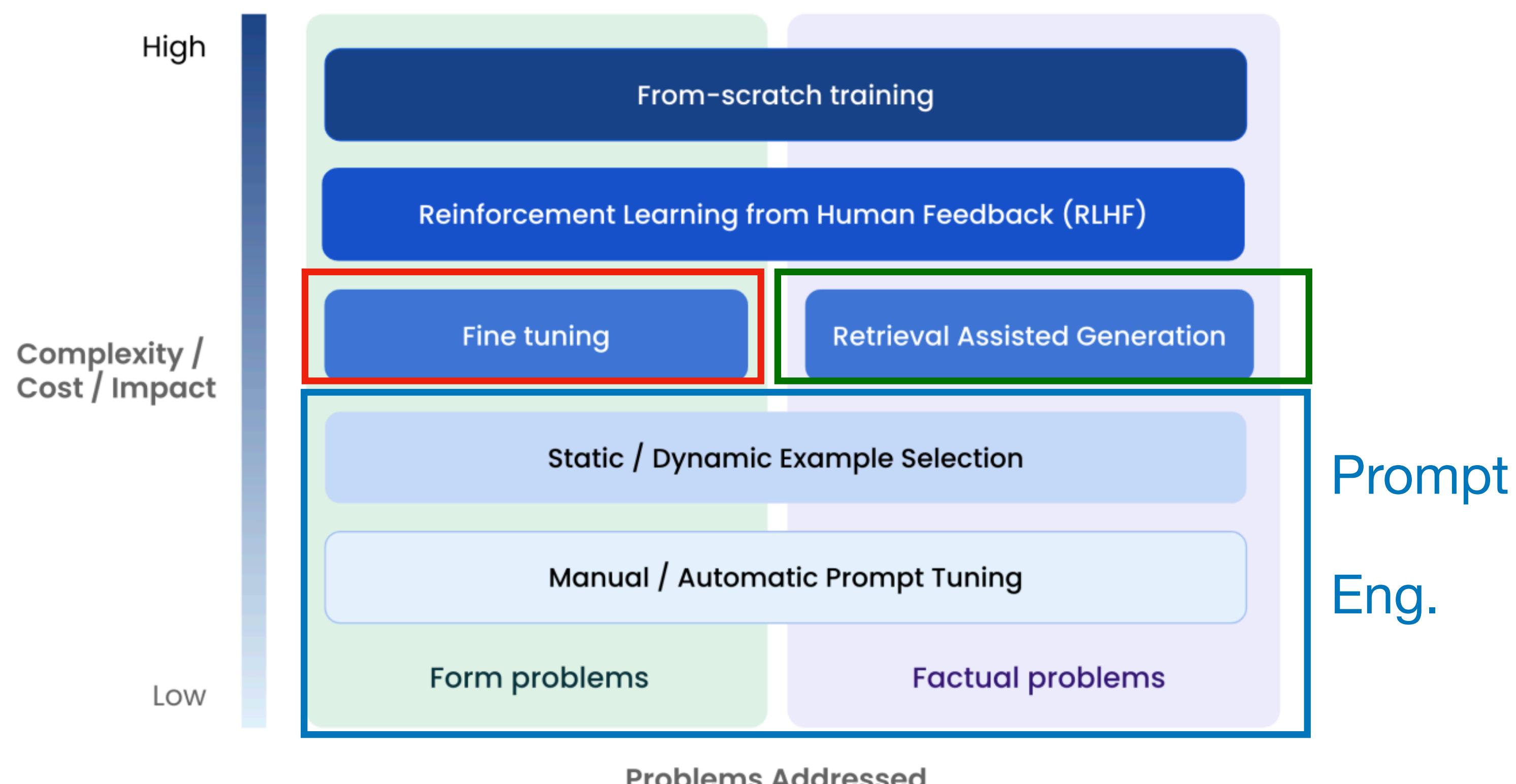
- Finetuned llama2 still behind gpt-4 on GSM8k (grade school math)



# What type of tasks?

- Start with simple method first and gradually move the more involved methods
  - In context learning
  - RAG:
    - When dealing with factual data
  - Fine tuning
    - Need to improve few shot behavior, domain adaptation
    - Reduce the known prompt patterns

## Space of Domain Specific Model Refinement (DSMR) techniques



# Customization options

## Fine-tuning commercial LLMs

## Fine-tuning OSS LLMs



### OpenAI fine-tuning service\*:

- Prepare and upload training data
- Train a new fine-tuned model
- Evaluate results and go back to step 1 if needed
- Use your fine-tuned model

#### OpenAI fine-tune dataset example

```
{"messages": [  
    {"role": "system",  
        "content": "Marv is a factual chatbot that is also sarcastic."},  
    {"role": "user",  
        "content": "What's the capital of France?"},  
    {"role": "assistant",  
        "content": "Paris, as if everyone doesn't know that already."}  
]
```

<https://platform.openai.com/docs/guides/fine-tuning>  
<https://docs.cohere.com/docs/training-custom-models>

<https://www.bloomberg.com/company/press/bloomberggpt-50-billion-parameter-llm-tuned-finance/>

<https://blog.replit.com/llm-training>

<https://openai.com/pricing>

<https://platform.openai.com/docs/guides/fine-tuning/preparing-your-dataset>

<https://openai.com/blog/openai-partners-with-scale-to-provide-support-for-enterprises-fine-tuning-models>

\*: GPT-3.5 Turbo

\* GPT-4-turbo fine-tuning service is just started

# Outlines

LLM Overview

Why consider your own LLMs

 Which LLMs to choose

Train your LLMs

Preparing Dataset for training

Evaluating LLMs

How to serve your own LLMs

# Which Open source LLMs

Target tasks

Requirements

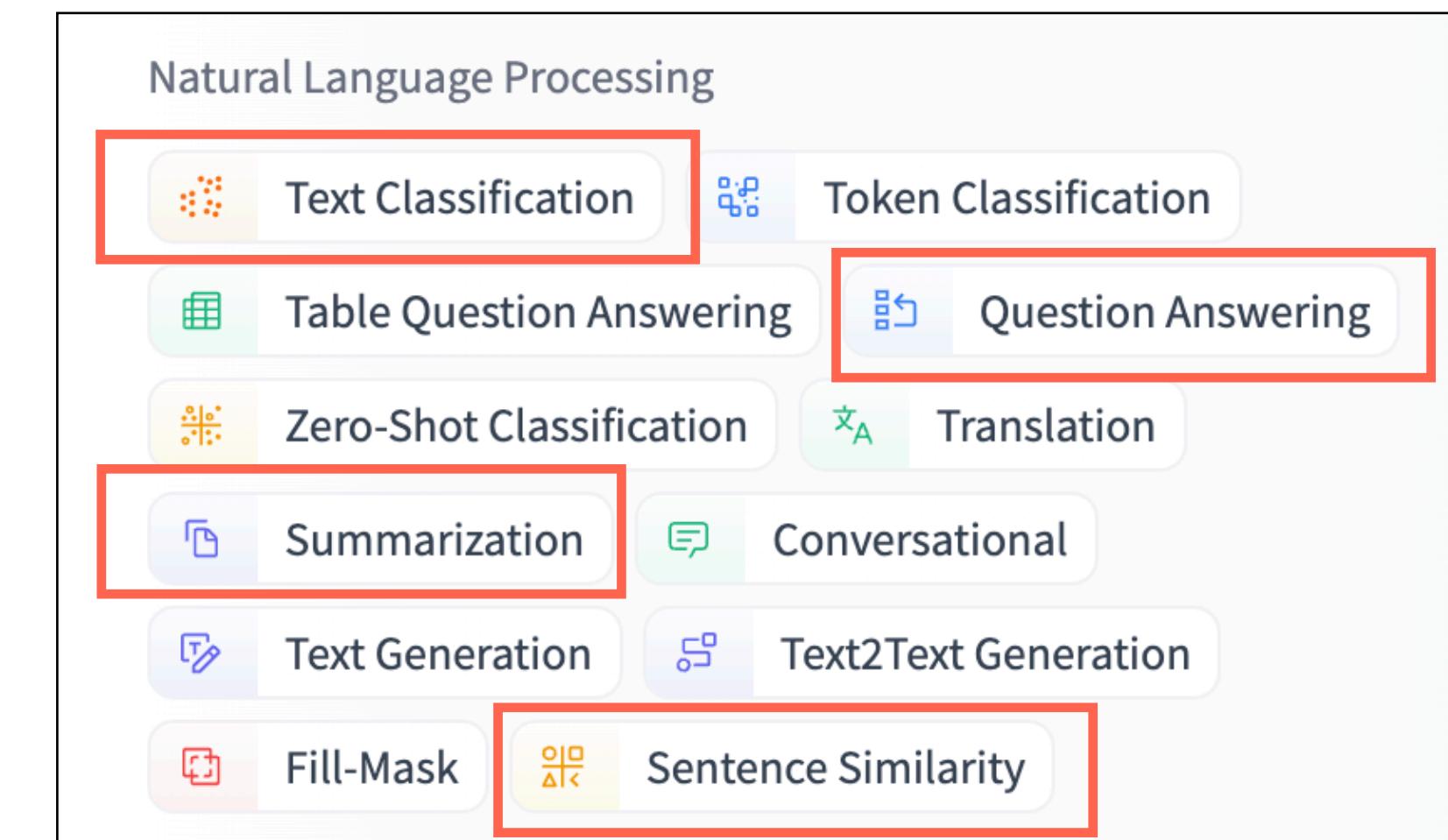
Model: size, structure, license

Dataset used

Training type: base, instruction-tuned

Evaluation: leaderboard

Iterative development



# Which LLMs

*Start with smaller model first*

Target tasks

Requirements

Model: size, structure, license

Dataset used

Training type: base, instruction-tuned

Evaluation: leaderboard

Iterative development

## Example model card

The screenshot shows a model card for the Llama-2-7b model on the Hugging Face platform. At the top, there's a logo for Hugging Face and a search bar. Below the search bar, the model name "meta-llama/Llama-2-7b" is displayed along with a "like" button and a count of 2.12k likes. There are also buttons for "Text Generation" and "PyTorch". A "Model card" tab is selected, showing the current page, and a "Files and versions" tab is also present. A "Gated model" notice states "You have been granted access to this model". The main content area is titled "Llama 2" and describes it as a collection of pretrained and fine-tuned generative text models ranging from 7 billion to 70 billion parameters. It mentions that this is the repository for the 7B pretrained model and links to other models can be found in the index at the bottom. A "Model Details" section includes a note about the Meta license and instructions to visit the website and accept the license before requesting access.

**Hugging Face** Search models, datasets, users...

meta-llama/Llama-2-7b like 2.12k

Text Generation PyTorch English facebook meta llama llama-2 arxiv:2307.09288

Model card Files and versions

Gated model You have been granted access to this model

**Llama 2**

Llama 2 is a collection of pretrained and fine-tuned generative text models ranging in scale from 7 billion to 70 billion parameters. This is the repository for the 7B pretrained model. Links to other models can be found in the index at the bottom.

**Model Details**

Note: Use of this model is governed by the Meta license. In order to download the model weights and tokenizer, please visit the [website](#) and accept our License before requesting access here.

# Which LLMs

Target tasks

Requirements

Model: size, structure, license

Dataset used

Training type: base, instruction-tuned

Evaluation:

Iterative development

- See if the model is used in similar tasks
- Check the evaluations score if the evaluation is similar to your task
- Start with small model and small size of dataset
- Run fraction of epoch to measure the performance
- Increase model size and/or increase dataset
- Iterate

# Outlines

LLM Overview

Why consider your own LLMs

Which LLMs to choose

 Train your LLMs

Preparing Dataset for training

Evaluating LLMs

How to serve your own LLMs

# Train your LLMs



## Huggingface:

- One stop shop for LLMs
- Model repo
- Dataset repo
- LLM libraries
- Model serving
- Model training

A screenshot of the Huggingface website. At the top, there is a search bar with the placeholder "Search models, datasets, users...". Below the search bar is a navigation menu with links for "Models", "Datasets", "Spaces", "Docs", "Solutions", "Pricing", and "Log In". The main content area features a large yellow emoji-like character with hands clasped together. Below the emoji, the text "The AI community building the future." is displayed in large, bold, white font. A smaller text below it reads "The platform where the machine learning community collaborates on models, datasets, and applications." To the right of the emoji, there is a list of available models categorized by task. Some categories include "Multimodal", "Computer Vision", "Natural Language Processing", "Audio", "Tabular", and "Reinforcement Learning". Each category lists several models with their names, descriptions, and statistics like "Updated 4 days ago", "25.2k", and "64".

Category	Model Name	Description	Last Updated	Statistics
Multimodal	meta-llama/Llama-2-70b	Text Generation • Updated 4 days ago	25.2k	64
Computer Vision	stabilityai/stable-diffusion-xl-base-0.9	Updated 6 days ago	2.01k	393
Natural Language Processing	openchat/openchat	Text Generation • Updated 2 days ago	1.3k	136
Audio	Illyasviel/ControlNet-v1-1	Updated Apr 26	1.87k	
Tabular	cerspense/zeroscope_v2_XL	Updated 3 days ago	2.66k	334
Reinforcement Learning	meta-llama/Llama-2-13b	Text Generation • Updated 4 days ago	328	64
Computer Vision	tiiuae/falcon-40b-instruct	Text Generation • Updated 27 days ago	288k	899
Natural Language Processing	WizardLM/WizardCoder-15B-V1.0	Text Generation • Updated 3 days ago	12.5k	332
Tabular	CompVis/stable-diffusion-v1-4	Text-to-Image • Updated about 17 hours ago	448k	5.72k
Reinforcement Learning	stabilityai/stable-diffusion-2-1	Text-to-Image • Updated about 17 hours ago	782k	2.81k
Computer Vision	Salesforce/xgen-7b-8k-inst	Text Generation • Updated 4 days ago	1.81k	57

# Train your LLMs

Dataset preparation

# Train your LLMs

## Model training

- Start with examples\*
- Simple training code
- Systematic experiments

\* see examples in reference section; due to rapid changes in libraries, you may need to change code/api depends on the library version you use

```
# load model
model = AutoModelForCausalLM.from_pretrained(
    model_path, device_map="auto",
)

# load tokenizer
tokenizer = AutoTokenizer.from_pretrained(
    model_path)

# load/set training arguments
training_arguments = TrainingArguments(...)

# load dataset
dataset = load_dataset(dataset_name)
train_data, val_data = dataset["train"], dataset["validation"]

# configure train
trainer = Trainer(
    model=model, args=training_args, train_dataset=train_data,
    eval_dataset=val_data
)

# train!!!
trainer.train()
```

# Compute resource

## GPU memory:

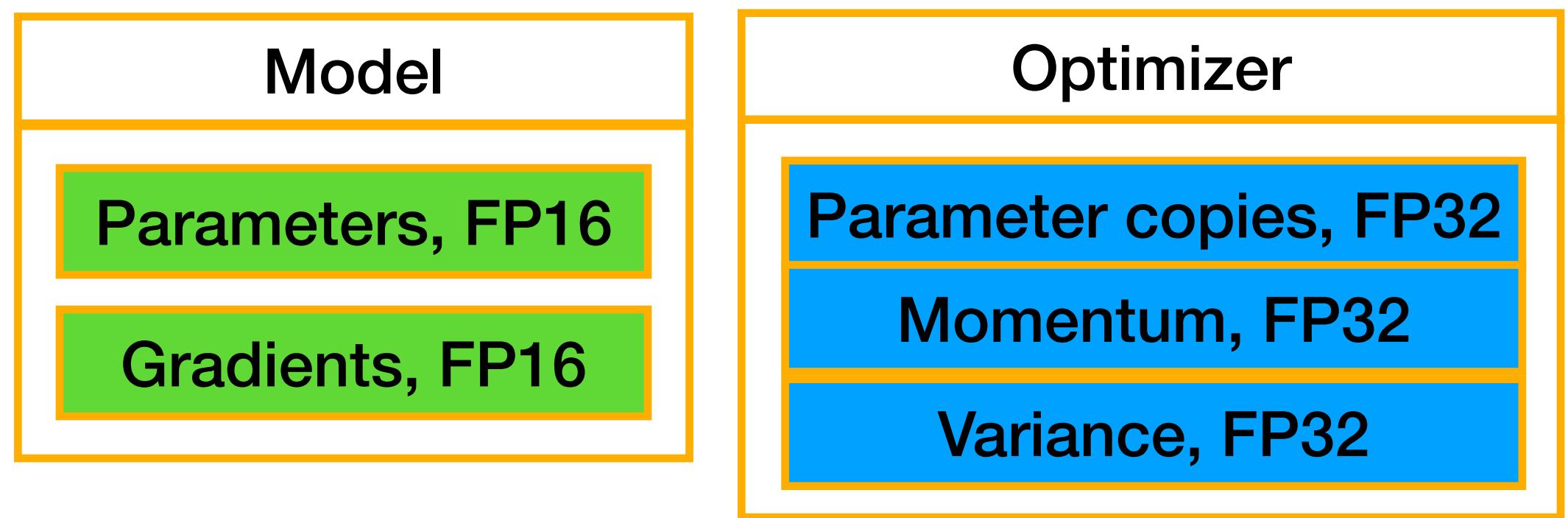
- RTX GPUs: ~24 GB
- A10G GPU: 24GB
- A100 GPU: 40/80 GB

Numbers <sup>100</sup> every LLM Developer  should know *					
 <b>Prompts</b>		 <b>Training and Fine Tuning</b>			
40-90%	Amount saved by appending "Be Concise" to your prompt	~\$1 million	Cost to train a 13 billion parameter model on 1.4 trillion tokens		
1.3	Average tokens per word	<0.001	Cost ratio of fine tuning vs training from scratch		
 <b>Price</b>		 <b>GPU Memory</b>			
~50	Cost Ratio of GPT-4 to GPT-3.5 Turbo	16GB	V100 GRAM capacity		
5	Cost Ratio of generation of text using GPT-3.5-Turbo vs OpenAI embedding	24GB	A10G GRAM capacity		
10	Cost Ratio of OpenAI embedding to Self-Hosted embedding	40/80GB	A100 GRAM capacity		
6	Cost Ratio of OpenAI base vs fine tuned model queries	<b>2x number of parameters</b>			
1	Cost Ratio of Self-Hosted base vs fine-tuned model queries	Typical GPU memory requirements of an LLM for serving			
<b>~1GB</b>		<b>&gt;10x</b>			
<b>1 MB</b>		Typical GPU memory requirements of an embedding model			
<b>Throughput improvement from batching LLM requests</b>		<b>GPU Memory required for 1 token of output with a 13B parameter model</b>			
* Check out <a href="https://bit.ly/llm-dev-numbers">bit.ly/llm-dev-numbers</a> for how we calculated the numbers					
Presented by  RAY &  anyscale with ❤️ Join the community <a href="https://ray.io">ray.io</a> or Request a Trial <a href="https://anyscale.com/signup">anyscale.com/signup</a> today					

# Param. & GPU memory

To train, need to store the followings on the GPU memory

- model parameters, 2 Bytes (floating point 16, FP16)
- model gradients, 2 Bytes
- optimizer states (Adam), 12 Bytes



*n* : number of parameters

Lower bound of static memory using Adam optimizer:

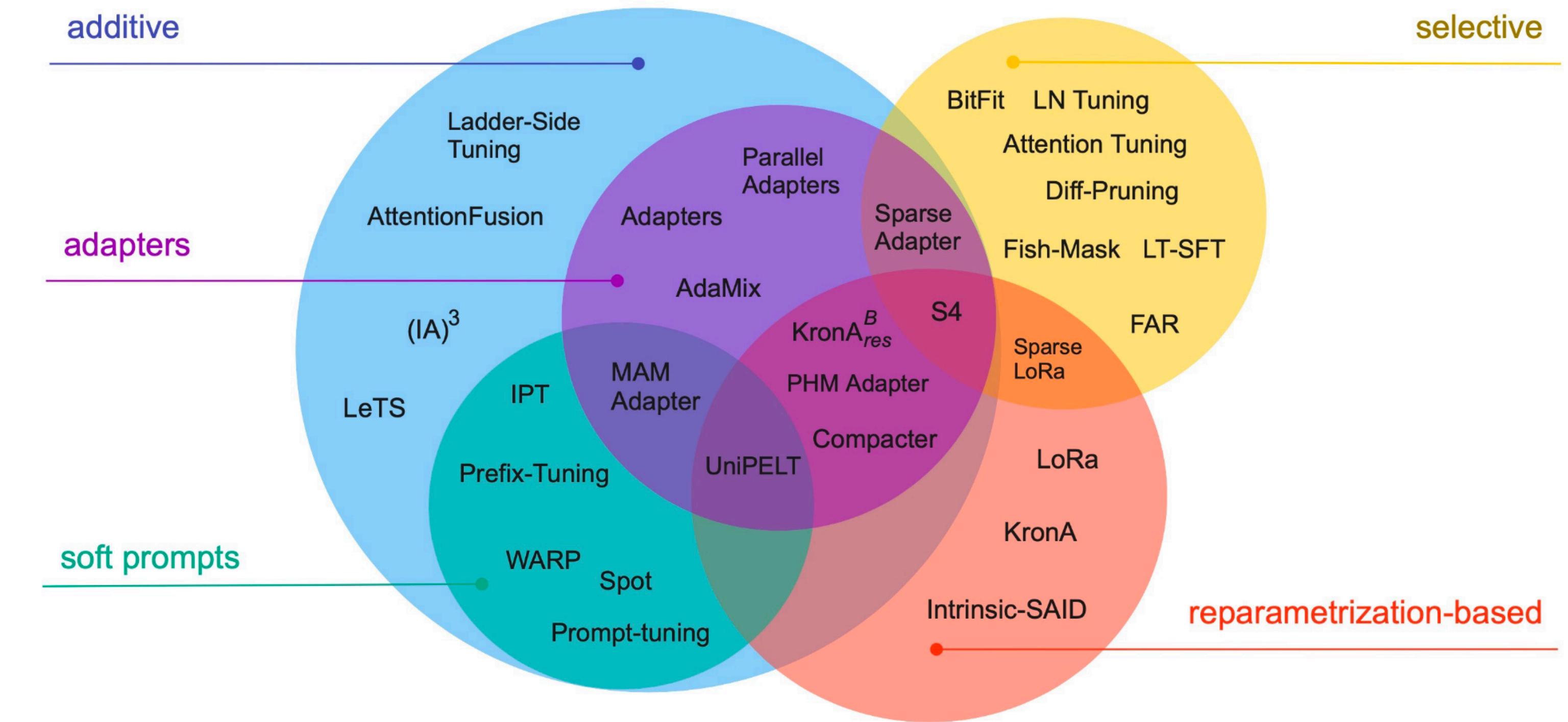
- 16 x n Bytes, (16=2+2+12)

ex> llama 2-7b, 7 b param. require ~ 112 GB GPU memory

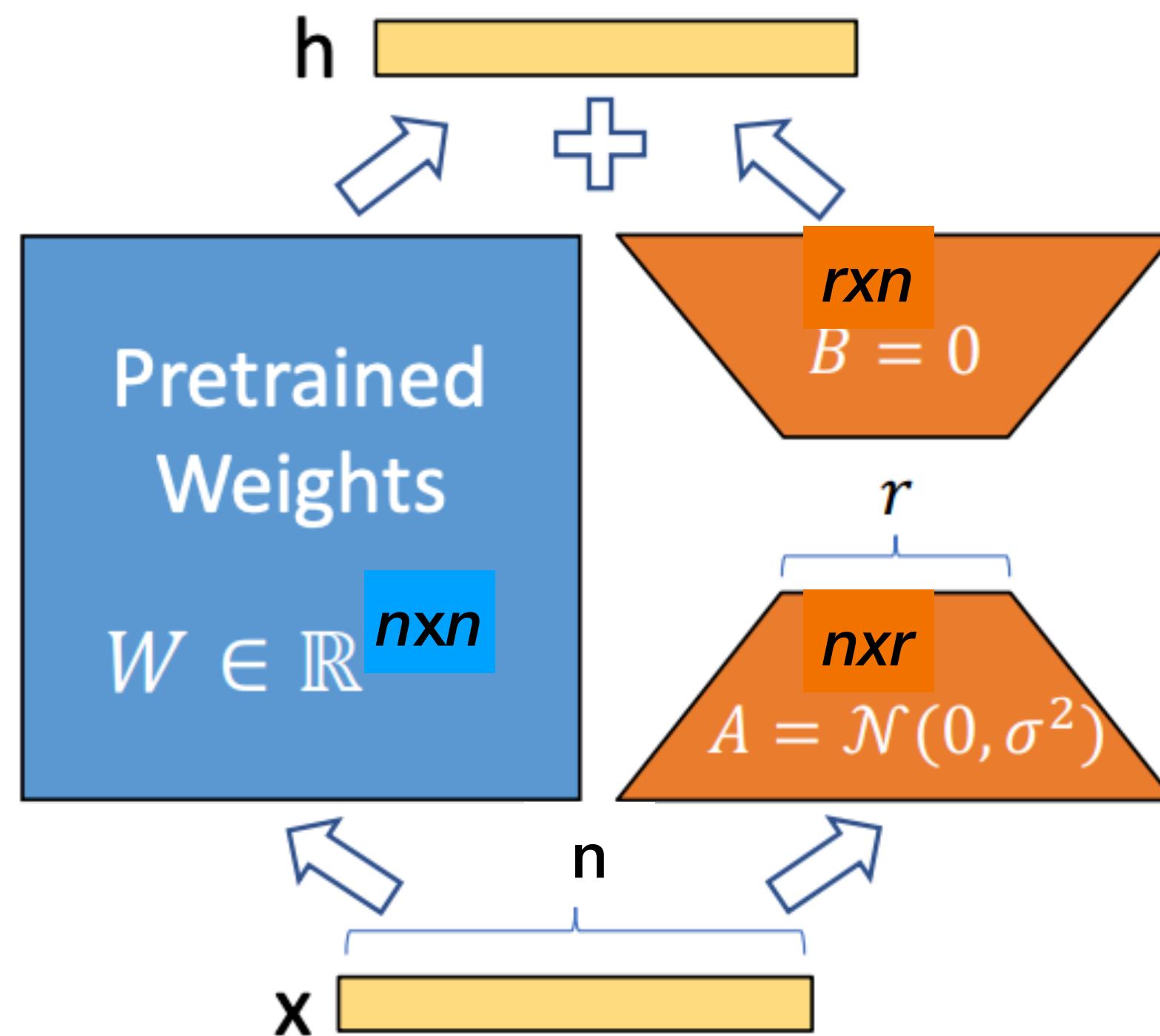
*This will not fit to 80GB memory on A100 GPU!*

# Parameter efficient fine tuning (PEFT)

- train additional small subset, additive
- train small set of model, re-parameterization
- *Low rank adaptation (LoRa)*



# Low rank adaptation (LoRa)



## Low rank adaptation (LoRa)

- Freeze pretrained weights
- Only train lower rank matrices

```
def lora_linear(x):  
    """LoRA  
    W: nxn, frozen param  
    W_A: nxr, W_B: rxn, where r < n, learnable param  
    ....  
    h = x @ W # regular linear  
    h += x @ W_A @ W_B # low-rank update  
    return scale * h
```

# Low rank adaptation (LoRa)

- Rank configuration
- target\_module: which modules to use

Trainable parameter with lora rank 8 w.r.t original number of trainable parameters: 0.3 %

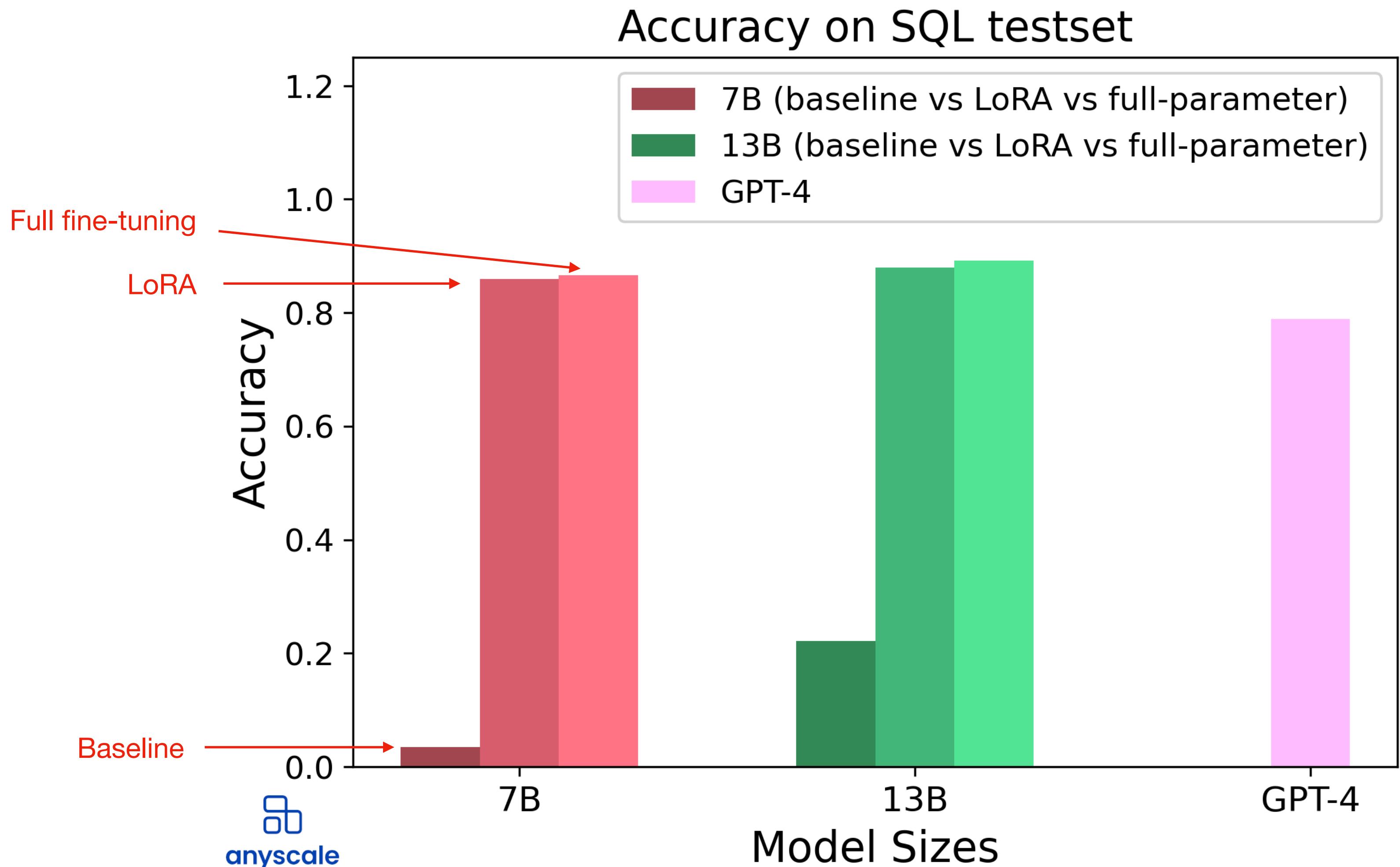
Memory required llama 2 7b model

- 112GB = 16x7
  - *14GB is for keep the frozen weight*
- =>  $(112 * 0.003 + 14)$  GB ~ 14 GB

```
peft_config = LoraConfig(  
    r=args.lora_r,  
    lora_alpha=args.lora_alpha,  
    lora_dropout=args.lora_dropout,  
    target_modules=modules,  
    bias="none",  
    task_type=TaskType.CAUSAL_LM,  
)  
  
model = get_peft_model(model, peft_config)
```

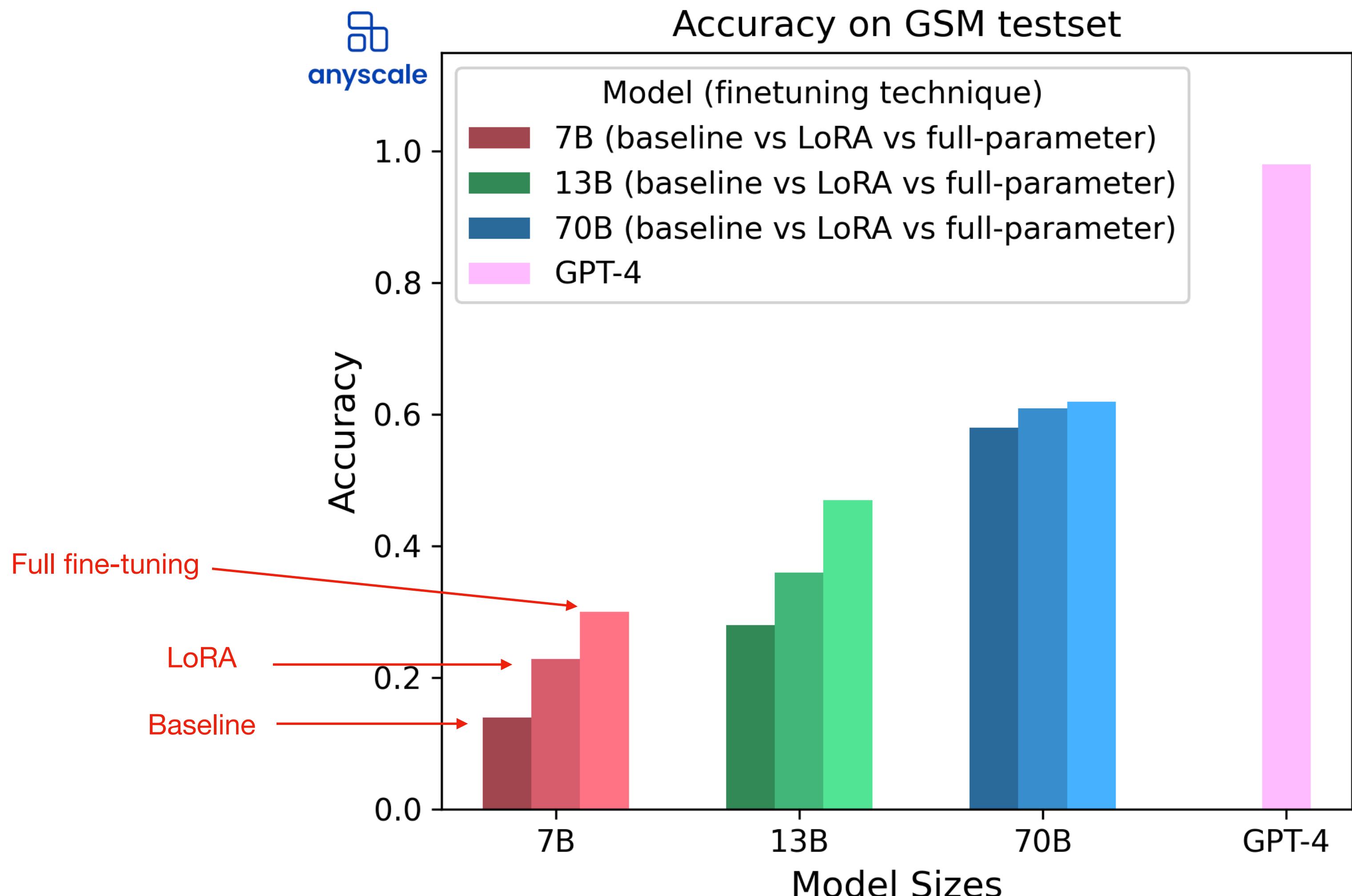
# Full fine-tuning vs LoRA

- How is the performance of LoRA compared to full fine-tuning?
  - It *depends*, but small
  - Simple task (**SQL** generation)
    - Small performance difference



# Full fine-tuning vs LoRA

- Complex task (**GSM8K**, grade school math)
  - Large performance difference in smaller model
  - small performance difference in larger model



# Outlines

LLM Overview

Why consider your own LLMs

Which LLMs to choose

Train your LLMs

 Preparing Dataset for training

Evaluating LLMs

How to serve your own LLMs

# Dataset preparation

Dataset for domain adaptation

Dataset for instruction tuning

- Prepare text dataset
- Verify dataset manually
- Apply filtering & dedup
- Continuous pre-training
- Next token prediction objective

# Dataset preparation

Dataset for domain adaptation

Dataset for instruction tuning

- Prepare Q&A pair dataset
- Allows chatgpt-like behavior after this training
- High quality dataset is important
- Suggested number of dataset varies from 100 to 1,000 or more. So test incrementally based on the your own results
- Self-instruct method: use LLM to generate/filter/improve Q&A pairs for dataset. There are many dataset follows this idea.

<https://platform.openai.com/docs/guides/fine-tuning>

LIMA, less is more, <https://arxiv.org/pdf/2305.11206.pdf>

Self-instruct: <https://arxiv.org/abs/2212.10560>

dolly-dataset:<https://www.databricks.com/blog/2023/03/24/hello-dolly-democratizing-magic-chatgpt-open-models.html>

Codellama2: <https://ai.meta.com/research/publications/code-llama-open-foundation-models-for-code/>

\*: frozen model only used to generate text

\*\*: model under training

# Outlines

LLM Overview

Why consider your own LLMs

Which LLMs to choose

Train your LLMs

Preparing Dataset for training

 Evaluating LLMs

How to serve your own LLMs

# Evaluation

- Very important to have evaluations metric early
- Find out metric that can measure the performance of the task
- LLM based evaluation can be effective

# Outlines

LLM Overview

Why consider your own LLMs

Which LLMs to choose

Train your LLMs

Preparing Dataset for training

Evaluating LLMs



How to serve your own LLMs

# How to serve LLMs

Post train model conversion

End point serving

- GPTQ: post train quantization (8, 4, 3, 2-bit precision) using the GPTQ algorithm
- AutoGPTQ: library that applies GPTQ for LLMs
  - Huggingface integrates AutoGPTQ library, which makes it easier to use

# How to serve LLMs

## Post train model conversion

### End point serving

- 😊 tgi (text generation inference)
  - LLM serving tool support
  - Support most OSS models
  - Feature rich tool
- vllm
  - Easy, Fast, and Cheap LLM Serving with PagedAttention
  - Performance number is very promising

# Summary

OSS LLMs for custom LLM use cases

Based on requirements, find OSS LLMs using  model hub

Use huggingface or similar framework to train model

Use PEFT method to reduce compute requirements

Prepare raw dataset and/or instruction following dataset, based on the application requirements.

Evaluate early and improve the evaluation metrics as needed

Serve LLM as API end point

# Questions?

# References

- Insop, [how to use llm talk](#)
- Anyscale, [Fine-Tuning LLMs: LoRA or Full-Parameter? An in-depth Analysis with Llama 2](#)
- Jeremy Howard, [A Hackers' Guide to Language Models](#)
- Andrej Karpathy, [Let's build GPT: from scratch, in code, spelled out.](#)
- Ludwig, [Efficient Fine-Tuning for Llama-v2-7b on a Single GPU](#)
- Huggingface, [Making LLMs even more accessible with bitsandbytes, 4-bit quantization and QLoRA](#)
- Huggingface, [Training a Llama in your backyard](#)
- QLoRA example, [Fine-tune LLaMA 2 \(7-70B\) on Amazon SageMaker](#)

# References

- AWS w/ *deeplearning.ai*, Generative AI with Large Language Models
- *Databricks*, *EdX*, Large Language Models: Foundation Models from the Ground Up
- *Databricks*, *EdX*, Large Language Models: Application through Production
- *deeplearning.ai*, Finetuning Large Language Models
- *Databricks*, Data+AI Summit 2023
- AWS, Generative AI Foundations on AWS Technical Deep Dive Series
- Anyscale, Fine-Tuning Llama-2: A Comprehensive Case Study for Tailoring Models to Unique Applications
- Anyscale, Fine tuning is for form, not facts

# Customization options

Fine-tuning commercial LLMs

Fine-tuning OSS LLMs

Train your LLMs from scratch

- Requires significant engineering and budget
- Design model structure and scales
- Dataset preparation
- There are commercial services\* to help pre-training process, but it is required to build in-house expertise
- Examples:
  - bloomberggpt
  - replit LLM

<https://www.bloomberg.com/company/press/bloomberggpt-50-billion-parameter-lm-tuned-finance/>

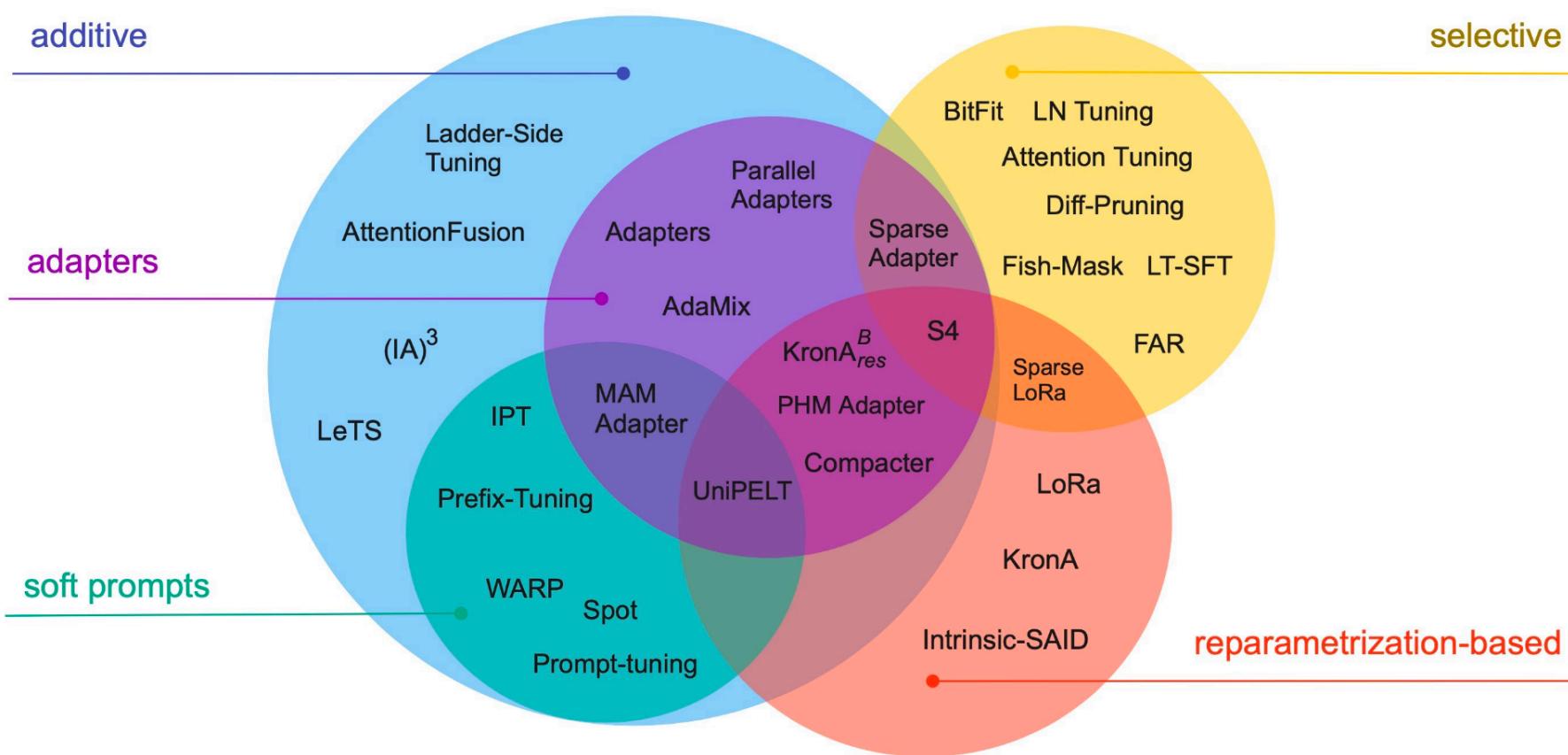
<https://blog.replit.com/llm-training>

\*: <https://www.mosaicml.com/>

\*: <https://huggingface.co/training-cluster>

\*: openAI will work with selected customers to build custom model, additional domain specific pre-training, to running a custom RL post-training process

# Train your LLMs



## Soft prompt tuning

- Concatenate trainable parameters to the input embeddings
- Consist with task description

```
def soft_prompted_model(input_ids):  
    """Soft prompt tuning  
    soft_prompt is a learnable parameter  
    """  
    x = Embed(input_ids)  
    x = concat([soft_prompt, x], dim=seq)  
    return model(x)
```

## Parameter efficient fine tuning (PEFT)

- train additional small subset, additive
- train small set of model, re-parameterization

## Low rank adaptation (LoRa)

- Freeze pretrain weight
- Only train lower rank matrices

```
def lora_linear(x):  
    """LoRA  
    W: nxn, frozen param  
    W_A: nxr, W_B: rxn, where r < n, learnable param  
    """  
    h = x @ W # regular linear  
    h += x @ W_A @ W_B # low-rank update  
    return scale * h
```