LLM Pre-training & Fine-tuning

September 8th, 2023

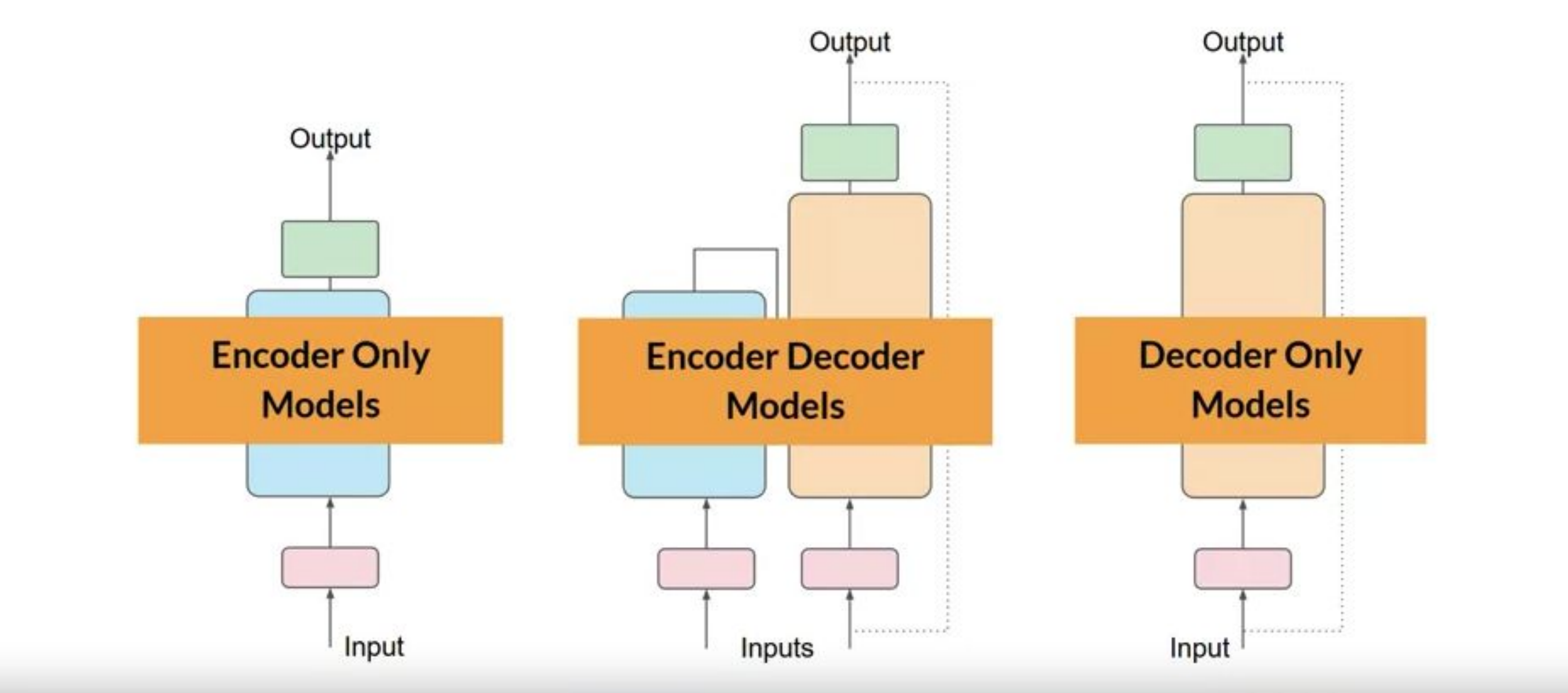
Overview

Topics we'll cover

- Evolution of LLMs
- LLM Pre-training
  - Data collection
  - Data cleaning
  - Tokenization
  - Causal Language Modeling
- LLM Training Considerations
  - Memory requirements
  - Mixed Precision training
  - Distributed training
  - Scaling laws
- Fine-tuning
  - Types of fine-tuning
  - Instruction datasets
  - PEFT, LoRA, QLoRA

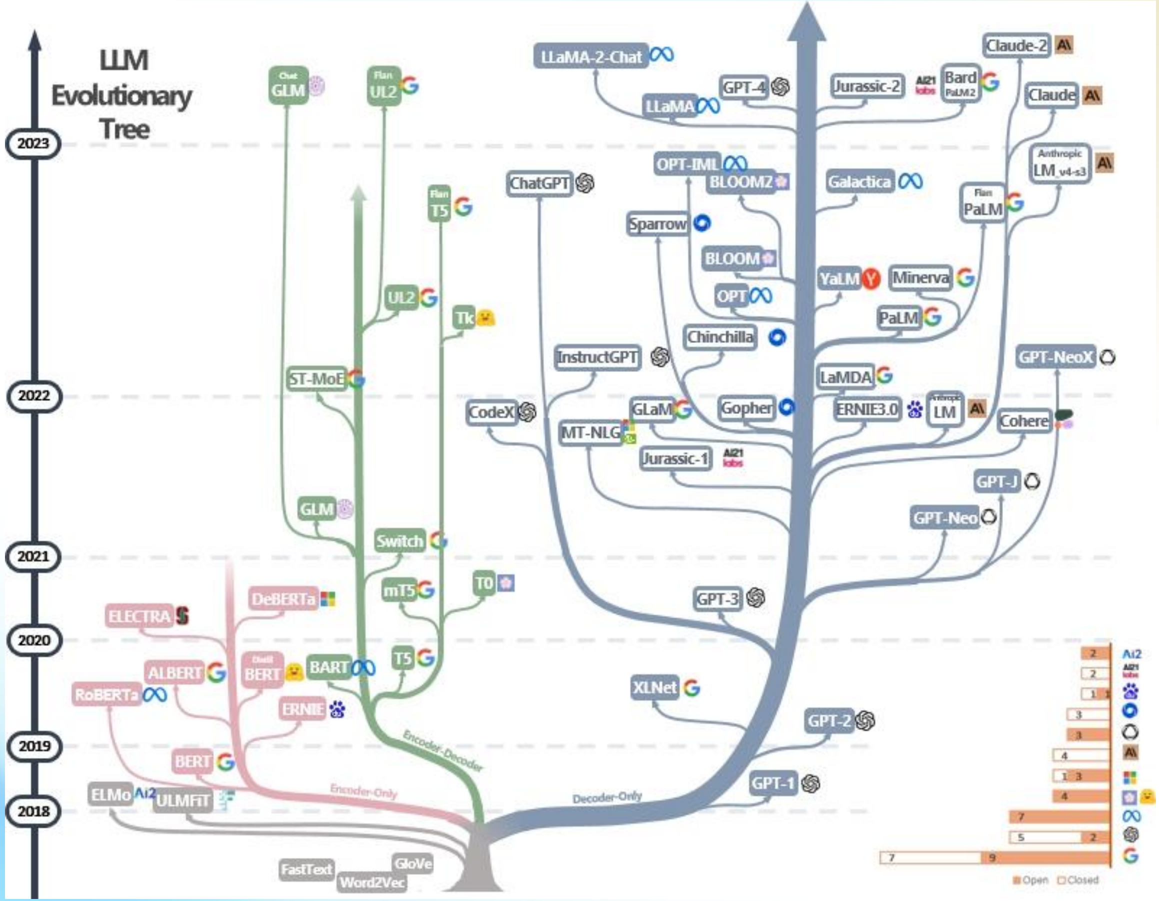
The Evolution of LLMs

A rapidly evolving ecosystem



https://www.coursera.org/learn/generative-ai-with-llms/

A rapidly evolving ecosystem



https://github.com/Mooler0410/LLMsPracticalGuide

Recipe for LLM Development

Stage	Pretraining	Supervised Fine-tuning	Reward Modeling	Reinforcement Learning
Dataset	**Raw (Web, Books)** Trillions of tokens Low quality, large quantity	**Instructions/Chat** ~10-100k Handwritten (prompt + response), High quality, low quantity	**Comparison** ~100k- 1M (prompt1 > prompt2) High quality, large quantity	**Prompts** ~10k - 100k (inputs) High quality, low quantity
Algorithm	**Language Modeling** Next Token prediction	**Language Modeling** Next Token prediction	**Binary Classification** Score Comparisons	**Reinforcement Learning** Maximize Reward for Gen
Model	**Base Model** From scratch	**SFT Model** Init from Base	**RM Model** Init from SFT	**RL Model** Init from SFT + RM Model
Compute	**100s - 1000s of GPUs** Weeks of training	**1 - 100s of GPUs** Days of training	**1-100s of GPUs** Days of training	**1-100s of GPUs** Days of training
Cost	**$5-10M**	**$100-50 000**	**$100-50 000**	**$100-50 000**

https://github.com/Mooler0410/LLMsPracticalGuide

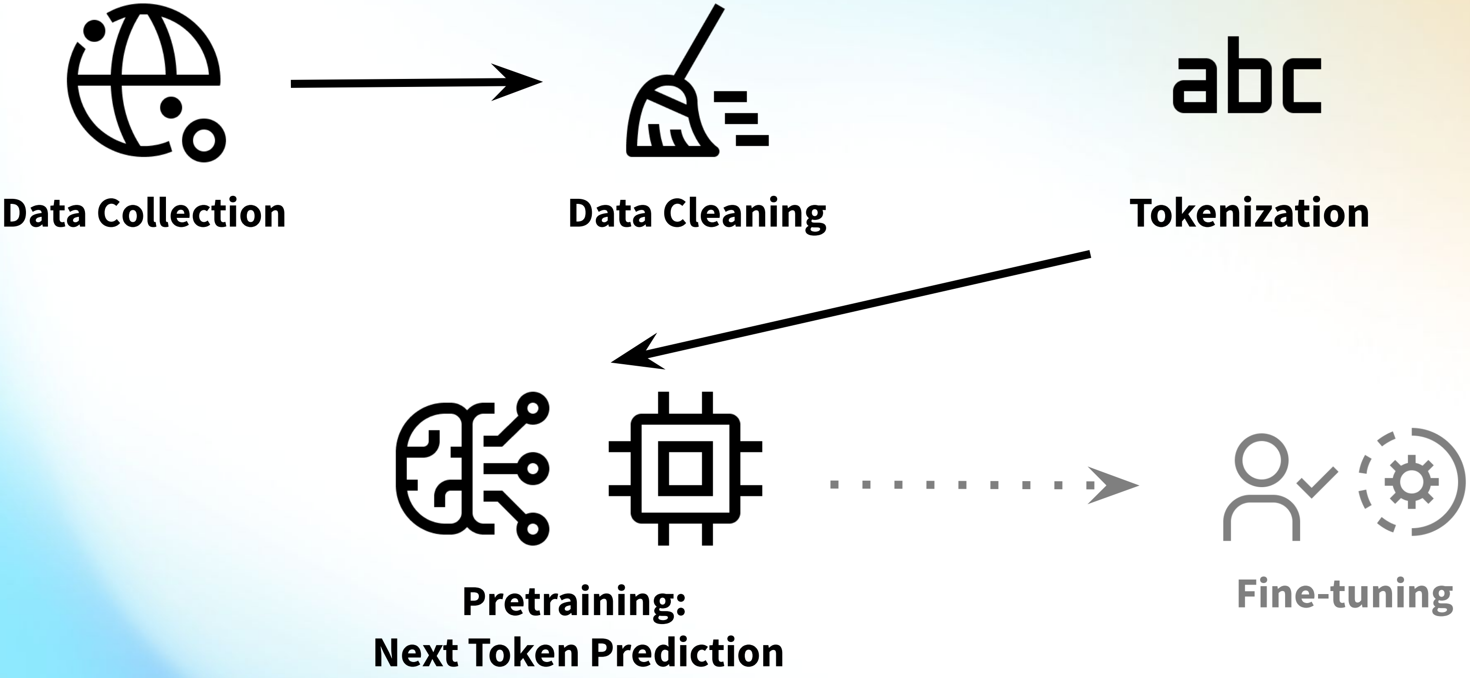
LLM Pre-training

Recipe for LLM Development

Stage	Pretraining	Supervised Fine-tuning	Reward Modeling	Reinforcement Learning
Dataset	**Raw (Web, Books)** Trillions of tokens Low quality, large quantity	**Instructions/Chat** ~10-100k Handwritten (prompt + response), High quality, low quantity	**Comparison** ~100k- 1M (prompt1 > prompt2) High quality, large quantity	**Prompts** ~10k - 100k (inputs) High quality, low quantity
Algorithm	**Language Modeling** Next Token prediction	**Language Modeling** Next Token prediction	**Binary Classification** Score Comparisons	**Reinforcement Learning** Maximize Reward for Gen
Model	**Base Model** From scratch	**SFT Model** Init from Base	**RM Model** Init from SFT	**RL Model** Init from SFT + RM Model
Compute	**100s - 1000s of GPUs** Weeks of training	**1 - 100s of GPUs** Days of training	**1-100s of GPUs** Days of training	**1-100s of GPUs** Days of training
Cost	**$5-10M**	**$100-50 000**	**$100-50 000**	**$100-50 000**

Adapted from: "State of GPT" talk

LLM Pre-training Workflow



Data Collection

- Before we can train a model, we need to collect data
- Large text corpus
  - Web scrapes (high quantity)
  - Books (high quality)
  - Academic literature
  - Code
- Also can be domain specific data (e.g. BloombergGPT)
  - Mix of public and private data

Llama Data Mixture

Dataset	Sampling prop.	Epochs	Disk size
CommonCrawl	67.0%	1.10	3.3 TB
C4	15.0%	1.06	783 GB
Github	4.5%	0.64	328 GB
Wikipedia	4.5%	2.45	83 GB
Books	4.5%	2.23	85 GB
ArXiv	2.5%	1.06	92 GB
StackExchange	2.0%	1.03	78 GB

Table 1: **Pre-training data.** Data mixtures used for pre-training, for each subset we list the sampling proportion, number of epochs performed on the subset when training on 1.4T tokens, and disk size. The pre-training runs on 1T tokens have the same sampling proportion.

Adapted from: "State of GPT" talk

Data Cleaning

Goal is to increase quality, address bias, remove harmful content.

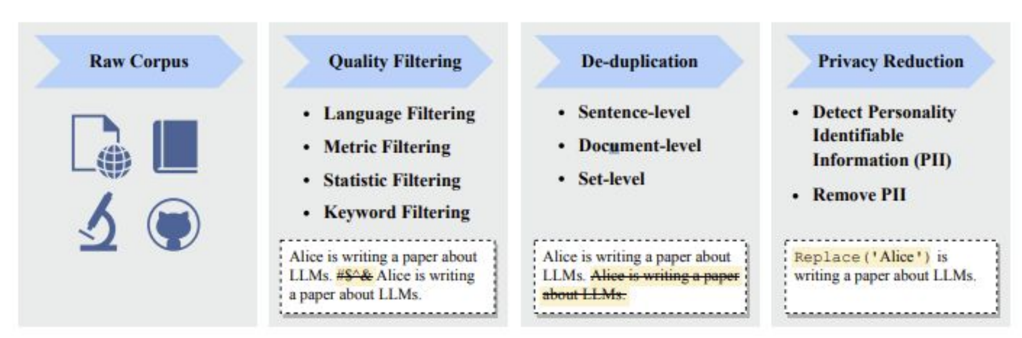
General steps

- **Language detection** - filter out undesired languages
- **Quality Filtering**
  - Metric based - use perplexity to remove unnatural sentences
  - Statistic based - punctuation distribution, symbol-to-word ratio, etc
  - Keyword based - explicit content, HTML, hyperlinks, boilerplate,
- **Deduplication** - crucial to avoid imbalance distribution, improve performance, reduce training steps needed (especially large models)
  - Exact deduplication (at document level)
  - Near-deduplication with MinHash + LSH
- **PII-removal**

Cleaned corpora

- RefinedWeb (600B tokens)
- CCNet (360B tokens)
- The Pile (340B tokens)

Usually 1-10% of original data is actually used!



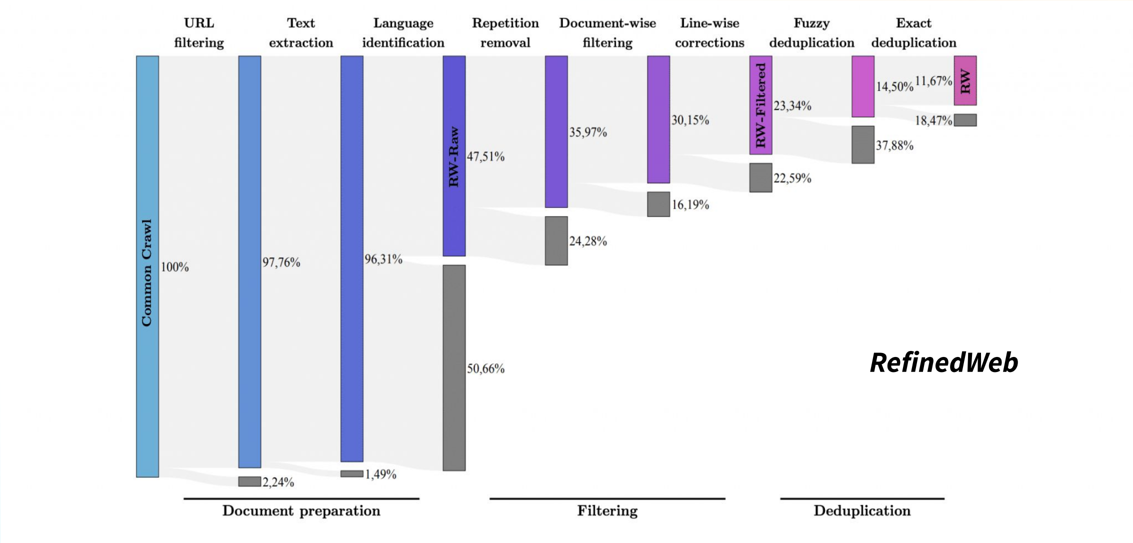


Figure 2. **Subsequent stages of Macrodata Refinement remove nearly 90% of the documents originally in CommonCrawl.** Notably,

[Appreciating the complexity of LLM data pipelines](#)
https://arxiv.org/pdf/2303.18223.pdf

Tokenization

Convert all text into list of integers

Typical vocab size

- ~30-100k tokens
- 1 token ~= 0.75 words

Typical algorithm

- Byte Pair Encoding
- Tokenizer is learned on the corpus
  - Learns rules to merge tokens based on most common character/sub-word pairs

Raw Text

Hugging Face is a company that develops tools for building applications using machine learning. It is most notable for its transformers library built for natural language processing applications and its platform that allows users to share machine learning models and datasets.

Tokens

Hugging Face is a company that develops tools for building applications using machine learning. It is most notable for its transformers library built for natural language processing applications and its platform that allows users to share machine learning models and datasets.

Inputs Ids

[48098, 2667, 15399, 318, 257, 1664, 326, 21126, 4899, 329, 2615, 5479, 1262, 4572, 4673, 13, 632, 318, 749, 12411, 329, 663, 6121, 364, 5888, 3170, 329, 3288, 3303, 7587, 5479, 290, 663, 3859, 326, 3578, 2985, 284, 2648, 4572, 4673, 4981, 290, 40522, 13]

Pre-training: next token prediction / causal language modeling

Raw Text Dataset (batch_size, context_length)

Row 1: Here is an example document 1 <|endoftext|>

Row 2: Example document 2 <|endoftext|> Example document 3 <|endoftext|> Example

Row 3: Document 4 <|endoftext|> Example Document 5

Processed Dataset

48098	2667	15399	318	257	1664	326	21126	**50526**
1262	4572	13	**50526**	632	290	**50526**	663	284
2748	4673	**50526**	3859	2985	15399	2899	2615	329

Batch Size (B)

Context Length (C)

Pre-training: next token prediction / causal language modeling

Each cell only "sees" cells in its row (on the left of it)

Trying to predict the next cell (on the right of it)

- Yellow = its context
- Green = current position prediction
- Red = its target

Processed Dataset

48098	2667	15399	318	257	1664	326	21126	50526
1262	4572	13	50526	632	290	50526	663	284
2748	4673	50526	3859	2985	15399	2899	2615	329

Batch Size (B)

Context Length (C)

Pre-training: next token prediction / causal language modeling

Random tokens, just garbage

After 0 steps

```
2u'T-t'wMOZeVsa.f0JC1hpndrsR6?to817dCVCyHwrWFYYGr"X8,lOWC!WAE
1!LtZf8&0r6d'KDiD77Wg'Y4NtV: 'NP"iPaWx6J"AEPADPrWMPbm"PB(2**S&0-
swgJlu:QmY
```

Random words, wrong grammar

After 500 steps

```
Cornon he this ther sall attred brendibled be on be lasible nothe fare
gorn ond free fartion, of the wholtthev had connes,nevers perss press,
forre prove the somaribliane.
```

Sentence with correct grammar

After 30 000 steps

```
"My dear Fanny, who is a match of your present satisfaction,and I am at
liberty and dinner, for everybody can be happy to you again; and now when
I think I used to be capable of other people,
```

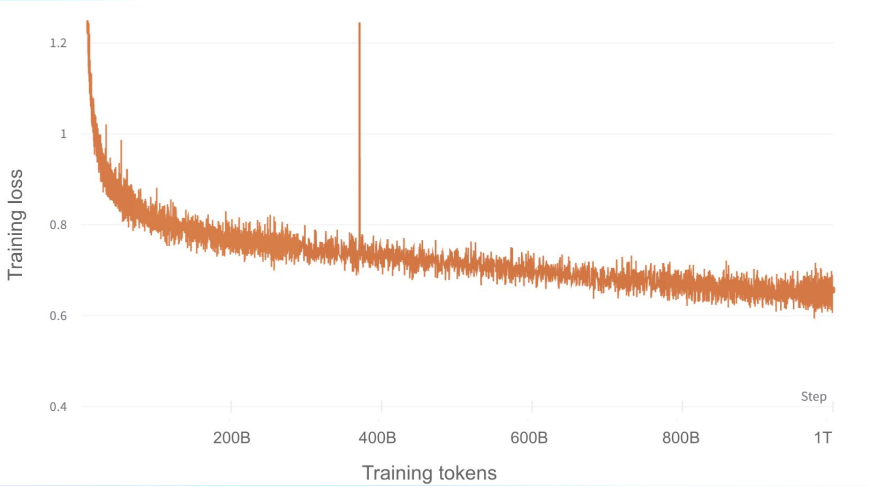
Pre-training: Examples

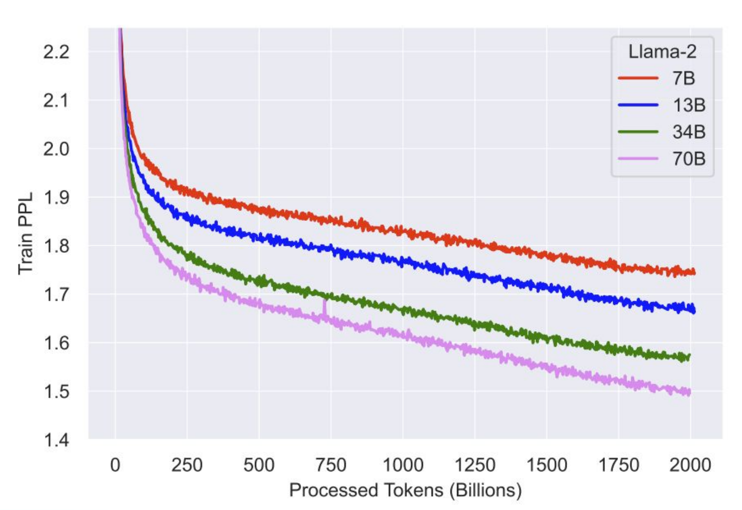
StarCoder

49,152 vocabulary size
8192 context length
15.5B parameter
Trained on 1T tokens
Trained for
86,016 GPU hours

Llama2

32,000 vocabulary size
4096 context length
7-70B parameter
Trained on 2T tokens
Trained for
1,720,320 GPU hours (70B)





LLM Training Considerations

Computational challenges

Approximate GPU RAM needed

1 parameter = 4 bytes (32-bit float)
1B parameters = 4x10^9 bytes = **4GB**

- Llama2-7B → 28GB
- Llama2-13B → 52GB
- Llama2-70B → 280GB

For reference, common GPU Hardware:

GPU	VRAM
NVIDIA Tesla T4	16 GB
NVIDIA A10	24 GB
NVIDIA A100	40 GB | 80 GB

Computational challenges

But that's just to store model weights…

		Bytes per parameter (for fp32 training)
Model parameters (weights)		4
Optimizer states (2 states for AdamW: moving averages of gradient + squared gradient)	Information that an optimization algorithm maintains during the training process	8
Gradients	Derivatives of the loss function with respect to the model's parameters. They represent how much the loss would change in response to small changes in each parameter.	4
Activations (Saved for gradient computation)	Intermediate outputs obtained during the forward pass of data through a neural network	4
Total		20 bytes per param

16 extra bytes per param

https://huggingface.co/docs/transformers/perf_train_gpu_one#anatomy-of-models-memory

Computational challenges

But that's just to store model weights…

	Bytes per parameter (for fp32 training)	**Llama2-7B (GB)**	**Llama2-13B (GB)**	**Llama2-70B (GB)**
Model parameters (weights)	4	28 GB	52 GB	280 GB
Optimizer states (2 states for AdamW)	8	56 GB	104 GB	560 GB
Gradients	4	28 GB	52 GB	280 GB
Activations	4	28 GB	52 GB	280 GB
Total	***20 bytes per param***	***140 GB***	***260 GB***	***1400 GB***

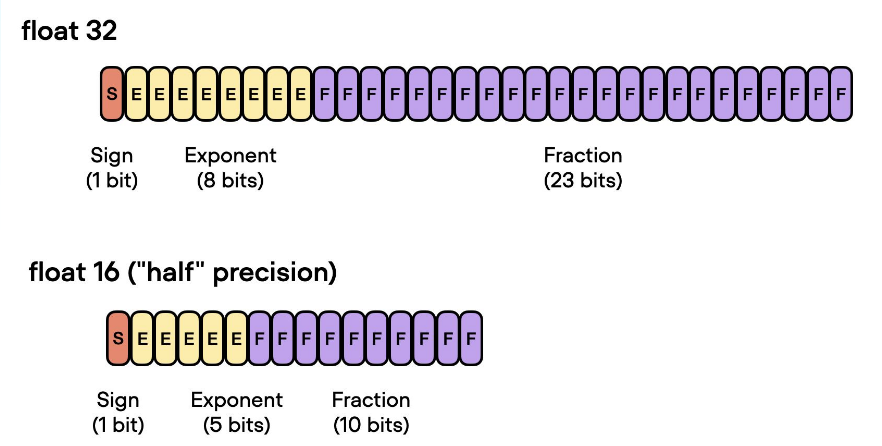
https://www.coursera.org/learn/generative-ai-with-llms

Computational challenges

So why not just use half-precision?

- FP32 computations are 2x slower than FP16, but…

- With fp16 we have a smaller range of possible values, which causes issues with backward pass:
    1. Weight updates are imprecise (unstable loss)
    2. Gradients can underflow (very small numbers replaced by 0)
    3. Gradients can overflow (very large numbers replaced by nan/inf)



Largest value represented by fp32 → 340,282,000,000,000,000,000,000,000,000,000,000,000

Largest value represented by fp16 → 65,504

https://docs.fast.ai/callback.fp16.html

Mixed-precision training

Reducing floating point precision helps reduce model size and speed up computations

Switch between 32 bit and 16 bit operations during training.

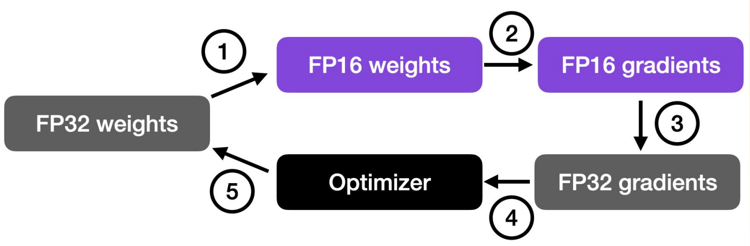
Benefits

- Requires less memory which enables training larger models
- Requires less memory bandwidth with speeds up data transfer operations
- Math operations run faster in reduced precision - speeds up training
- No loss in task-specific accuracy

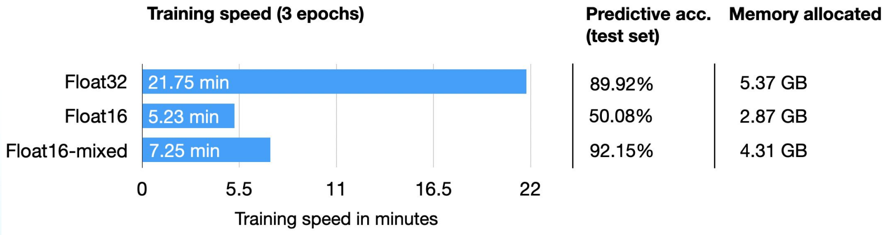
It works by identifying which steps require full precision for numerical stability, and using 16-bit everywhere else

This is standard method for pre-training models today

Forward pass and gradient computation are done in half precision == **fast compute**



Copy gradients back to full precision and update the model == **numerical stability**



Distributed Training

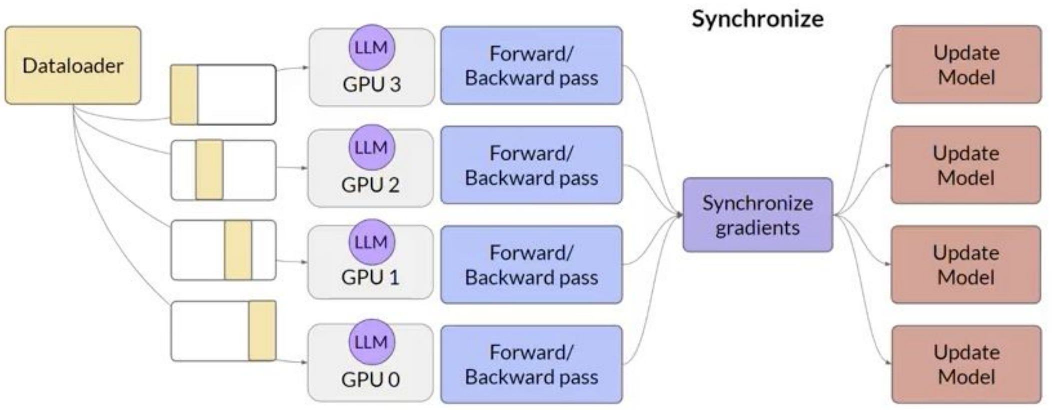
Distributed Data Parallel (DDP)

When to use

- Use this if your model fits on one GPU
- But you want to speed up training

How it works

- Full model is replicated over multiple GPU's
- Each is fed a slice of the data
- Processing is done in parallel, then synchronized after each train step



https://www.coursera.org/learn/generative-ai-with-llms

Distributed Training

Zero Redundancy Optimizer (ZeRO) - Microsoft

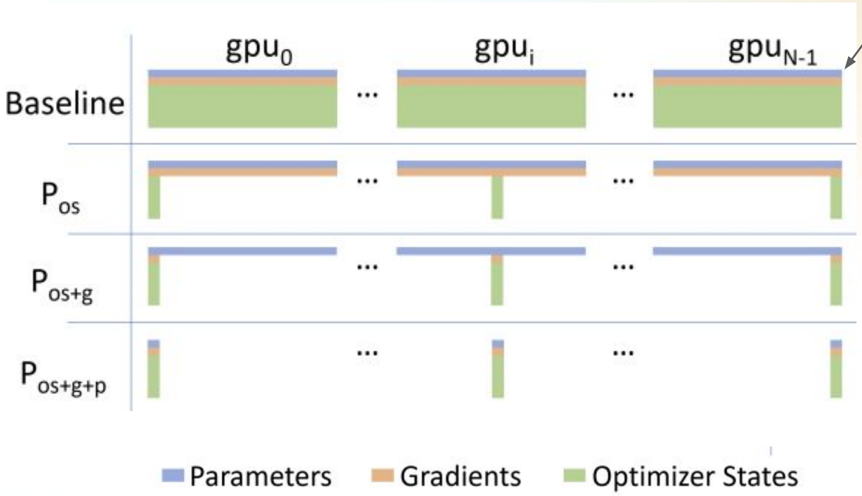
When to use

- When model doesn't fit in VRAM on single GPU

How it works

- Optimize memory by sharding/distributing model states across GPU's with zero data overlap
- Different "stages" of sharding depending on what how much of the memory is distributed
- Trade-off in configuration complexity and I/O communication time to sync sharded outputs

Full model copy



ZeRO Stage 1
Shard Optimizer States
(4x memory savings)

ZeRO Stage 2
Shard Optimizer States and gradients
(8x memory savings)

ZeRO Stage 3
Shard Optimizer States and gradients and model weights
(#GPUx memory savings)

https://arxiv.org/pdf/1910.02054.pdf

Distributed Training

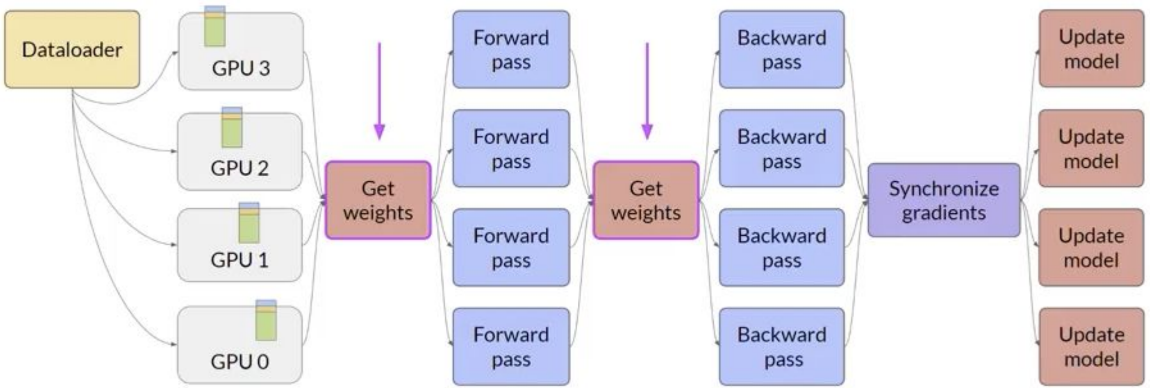
Fully Sharded Data Parallel (FSDP)

When to use

- When model doesn't fit in VRAM on single GPU

How it works

- Optimize memory by sharding/distributing model states across GPU's with zero data overlap
- Different "stages" of sharding depending on what how much of the memory is distributed
- Stage3 aka Fully Sharded Data Parallel (FSDP)



https://www.coursera.org/learn/generative-ai-with-llms

Chinchilla Scaling Laws for Compute-optimal Models (2022)

Deepmind investigated the optimal model size and # tokens for training LLMs under a given compute budget

Experiments

- Previously thought ([from Open AI paper]) that scaling model size was they key to improve model loss
- Trained >400 models from 70M to 16B parameters on 5-500B tokens
- Then, trained Chinchilla-70B with same compute budget as Gopher-280B, but 4x more data

Findings

- Chinchilla outperforms Gopher (and GPT3) on a large range of tasks
- So…
- Many existing large models are **over-parameterized** and **under-trained**
- Smaller models trained on more data could perform as well as larger models
- ***Compute optimal training datasize is ~20x the number of model parameters***

Model	# of parameters	Compute-optimal* # of tokens (~20x)	Actual # tokens
Chinchilla	70B	~1.4T	1.4T
LLaMA-65B	65B	~1.3T	1.4T
GPT-3	175B	~3.5T	300B
OPT-175B	175B	~3.5T	180B
BLOOM	176B	~3.5T	350B

Table 3 | **Estimated optimal training FLOPs and training tokens for various model sizes.** For various model sizes, we show the projections from Approach 1 of how many FLOPs and training tokens would be needed to train compute-optimal models. The estimates for Approach 2 & 3 are similar (shown in Section D.3)

Parameters	FLOPs	FLOPs (in *Gopher* unit)	Tokens
400 Million	1.92e+19	1/29,968	8.0 Billion
1 Billion	1.21e+20	1/4,761	20.2 Billion
10 Billion	1.23e+22	1/46	205.1 Billion
67 Billion	5.76e+23	1	1.5 Trillion
175 Billion	3.85e+24	6.7	3.7 Trillion
280 Billion	9.90e+24	17.2	5.9 Trillion
520 Billion	3.43e+25	59.5	11.0 Trillion
1 Trillion	1.27e+26	221.3	21.2 Trillion
10 Trillion	1.30e+28	22515.9	216.2 Trillion

https://www.coursera.org/learn/generative-ai-with-llms
https://arxiv.org/pdf/2203.15556.pdf

Fine-tuning

Recipe for LLM Development

Stage	Pretraining	Supervised Fine-tuning	Reward Modeling	Reinforcement Learning
Dataset	**Raw (Web, Books)** Trillions of tokens Low quality, large quantity	**Instructions/Chat** ~10-100k Handwritten (prompt + response), High quality, low quantity	**Comparison** ~100k- 1M (prompt1 > prompt2) High quality, large quantity	**Prompts** ~10k - 100k (inputs) High quality, low quantity
Algorithm	**Language Modeling** Next Token prediction	**Language Modeling** Next Token prediction	**Binary Classification** Score Comparisons	**Reinforcement Learning** Maximize Reward for Gen
Model	**Base Model** From scratch	**SFT Model** Init from Base	**RM Model** Init from SFT	**RL Model** Init from SFT + RM Model
Compute	**100s - 1000s of GPUs** Weeks of training	**1 - 100s of GPUs** Days of training	**1-100s of GPUs** Days of training	**1-100s of GPUs** Days of training
Cost	**$5-10M**	**$100-50 000**	**$100-50 000**	**$100-50 000**

Adapted from: "State of GPT" talk

Why supervised fine-tuning?

Base models are optimized to predict the next word based on the corpus they were trained on… which alone isn't all that useful

Prompt	*Explain the moon landing to a 6 year old in a few sentences.*
Completion	GPT-3
	`Explain the theory of gravity to a 6 year old.`
	`Explain the theory of relativity to a 6 year old in a few sentences.`
	`Explain the big bang theory to a 6 year old.`
	`Explain evolution to a 6 year old.`

Base model is not trained to be an assistant (e.g. chatty)

Why supervised fine-tuning?

Instruction fine-tuning makes it easier to access models knowledge in a familiar, conversational way

Prompt	*Explain the moon landing to a 6 year old in a few sentences.*
	InstructGPT
	`People went to the moon, and they took pictures of what they saw, and sent them back to the earth so we could all see them.`

Source: Open AI

Fine tuned models are.

Types of fine-tuning

1. **Continued pre-training:** With domain-specific data, apply the same pre-training regime (next token prediction, masked language modeling) on the base model → Allows for domain adaptation.

2. **Single-task fine-tuning:** The pre-trained model is honed for a narrow and specific task such as toxicity detection or summarization, similar to BERT and T5 → Allows you to use smaller models with fewer labeled examples.
   - Possibility for catastrophic forgetting (aka alignment tax)
   - This may not be a bad thing!

3. **Multi-task / Instruction fine-tuning:** The pre-trained (base) model is fine-tuned on examples of instruction-output pairs to follow instructions, answer questions, and be conversationally be helpful → Allows you to interact with model naturally.

4. **Reinforcement learning with human feedback (RLHF):** This combines instruction fine-tuning with reinforcement learning. It requires collecting human preferences (e.g., pairwise comparisons) which are then used to train a reward model. The reward model is then used to further fine-tune the instructed LLM via RL techniques such as proximal policy optimization (PPO).

Multi-task fine-tuning

- T5 from Google (2019) was the first model that enabled multi-task capabilities from a single model
- Previously, needed a specific fine-tuned model per task
- First example of “prompt engineering” (in retrospect)

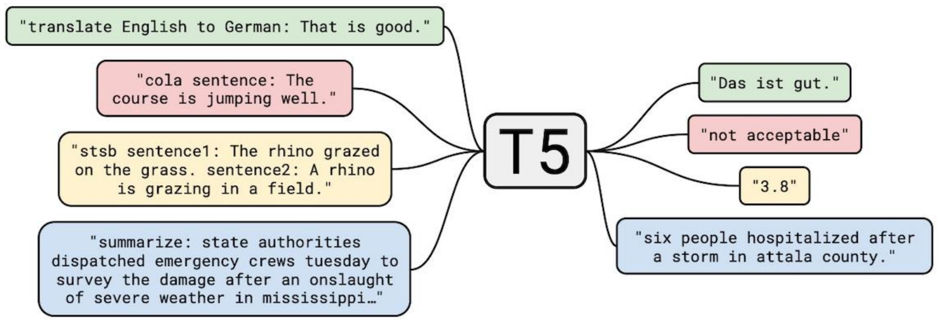
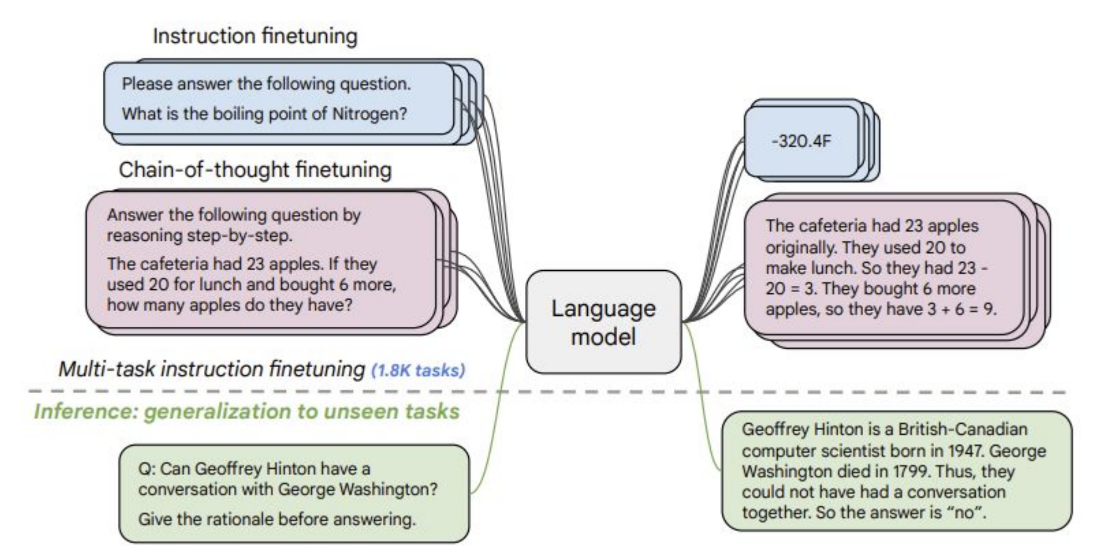


Figure 1: A diagram of our text-to-text framework. Every task we consider—including translation, question answering, and classification—is cast as feeding our model text as input and training it to generate some target text. This allows us to use the same model, loss function, hyperparameters, etc. across our diverse set of tasks. It also provides a standard testbed for the methods included in our empirical survey. “T5” refers to our model, which we dub the “**T**ext-**t**o-**T**ext **T**ransfer **T**ransformer”.

https://arxiv.org/pdf/1910.10683.pdf

Instruction fine-tuning

- FLAN-T5 (2022)
- Fine tuning on *instruction-context-answer* pairs from a variety of tasks
- Improves usability via natural language
- Helps to generalize to unseen tasks



https://arxiv.org/pdf/2210.11416.pdf

Instruction datasets

What makes a good instruction dataset?

- Clear specific instructions
- Diverse set of topics / tasks
- Consistent formatting
- Human feedback

Human-written

- Dolly - 15k crowdsourced examples

Imitation learning (simple instructions)

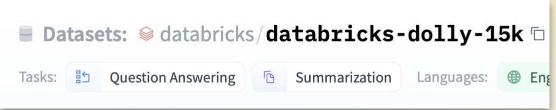
- Alpaca - used [self-instruct](), 52k examples
- Vicuna - 70k examples from ShareGPT
- GPT4All - 800k examples from ChatGPT
- LIMA - 1k examples, high quality

Better imitation learning (complex instructions)

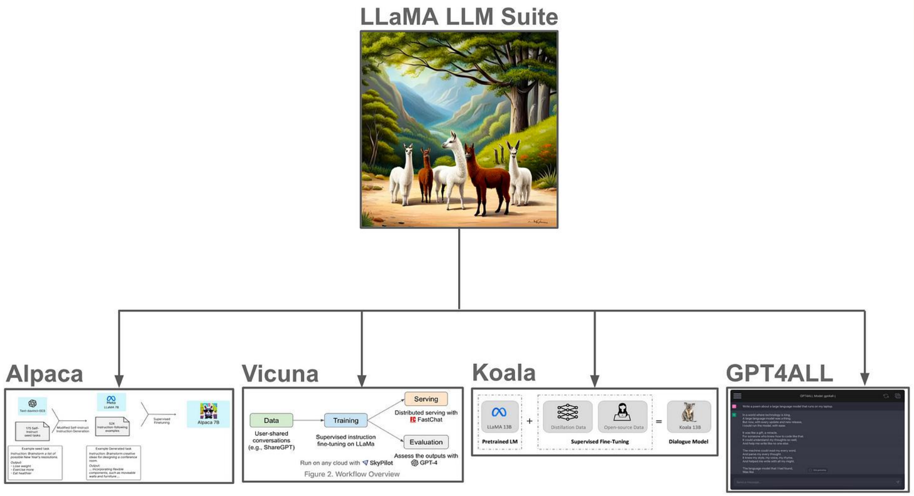
- WizardLM - [EvolInstruct]()
- Orca - Explanation traces

https://arxiv.org/pdf/2210.11416.pdf

Human-written



LLM Generated (imitation learning)



Parameter Efficient Fine-tuning (PEFT)

Full model training is inaccessible without significant compute

	Bytes per parameter (for fp32 training)	**Llama2-7B (GB)**	**Llama2-13B (GB)**	**Llama2-70B (GB)**
Model parameters (weights)	4	28 GB	52 GB	280 GB
Optimizer states (2 states for AdamW)	8	56 GB	104 GB	560 GB
Gradients	4	28 GB	52 GB	280 GB
Activations	4	28 GB	52 GB	280 GB
Total	***20 bytes per param***	***140 GB***	***260 GB***	***1400 GB***

Parameter Efficient Fine-tuning (PEFT)

What?

Parameter-Efficient Fine-Tuning (PEFT) is a technique that allows us to fine-tune a large pretrained model on a specific downstream task while requiring significantly fewer parameters than full fine-tuning.

Why?

- **Recipe:** Pre-training on generic data + fine-tuning on specific downstream task
- **Large LLMs:** full fine-tuning becomes infeasible to train on consumer hardware
- **Catastrophic forgetting:** tuning all model parameters is prone to overfitting
- **Storage:** storing and deploying fine-tuned models independently for each downstream task becomes very expensive

How?

PEFT approaches only fine-tune a small number of (extra) model parameters while freezing most parameters of the pretrained LLMs, thereby greatly decreasing the computational and storage costs, being portable, avoiding catastrophic forgetting and better in low data regimes.

Methods

- Prefix tuning
- Prompt tuning
- LoRA
- etc.

LoRA Overview

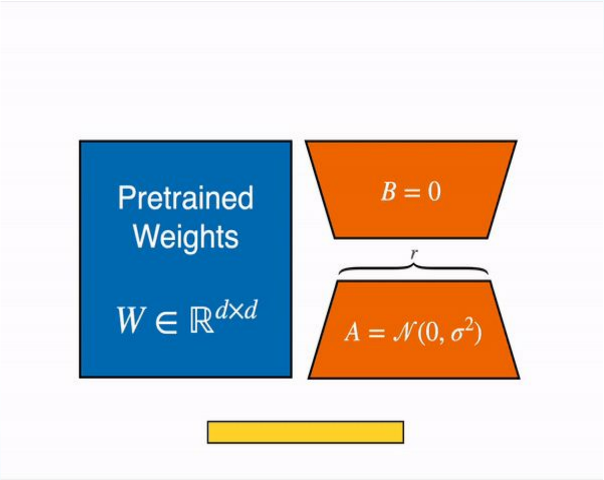


Figure explaining how LoRA layers works: extra parameters (in blue) are added on top of frozen layers (in orange)
Adapted from the original paper, figure 1

Pros:

- Fine tuning of LLMs using a fraction of the memory requirements
- Fine-tuning is accessible on common GPU's
- Tiny checkpoints (example here)
- Performance comparable to full fine-tuning
- No Inference latency addition

Cons:

- (only during training) The forward and backward pass is approximately twice as slow, due to the additional matrix multiplications in the adapter layers.

LoRA Overview

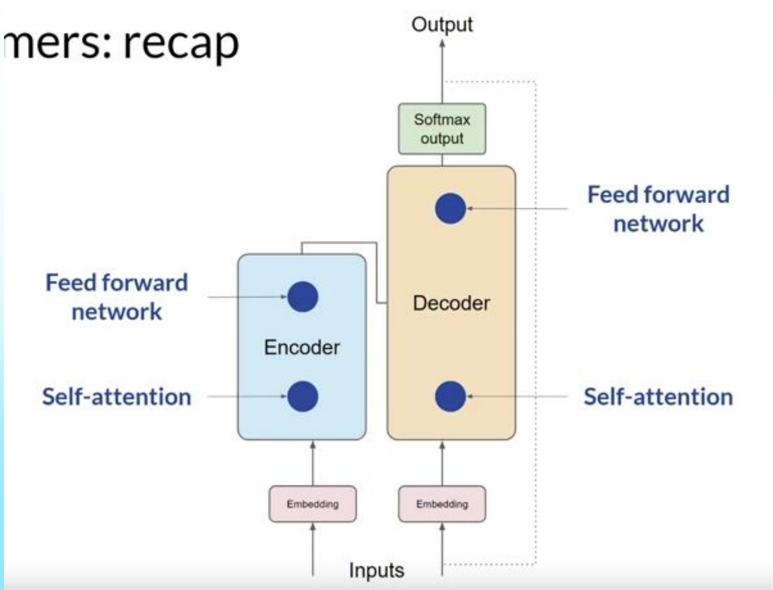
The theory behind it:

- The rank of the matrix is the number of linearly independent column vectors.
- Full rank means all columns are linearly independent.
- Low rank means some columns are linear combinations of the other columns.

- *We take inspiration from Li et al. (2018a); Aghajanyan et al. (2020) which show that the learned over-parametrized models in fact reside on a low intrinsic dimension. We hypothesize that the change in weights during model adaptation also has a low "intrinsic rank"*

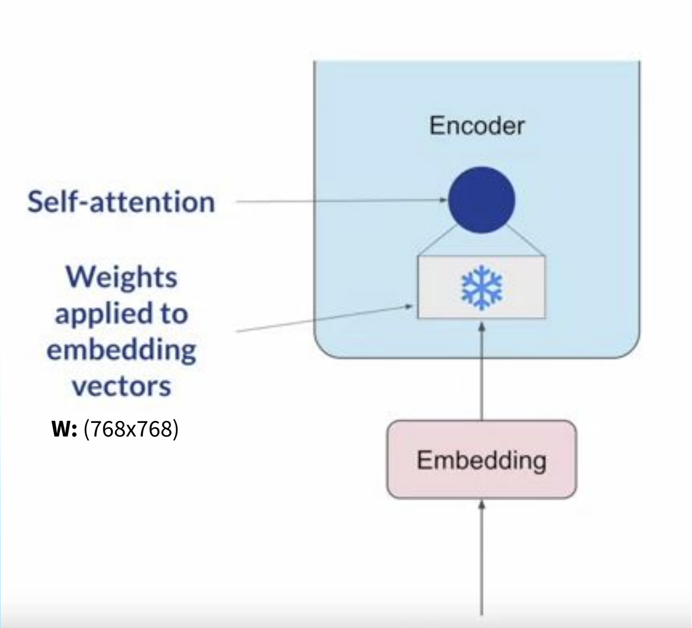
LoRA Overview

During training:



LoRA Overview

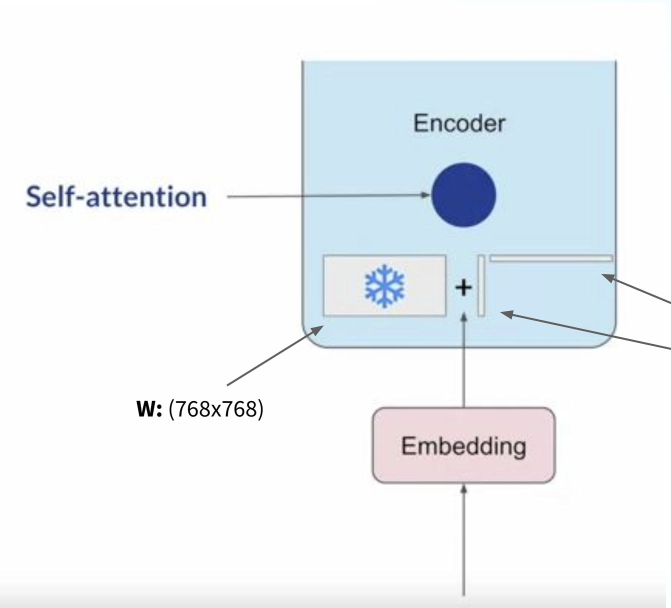
During training:



1. Freeze original self-attention weights (for specified layers)

LoRA Overview

During training:



1. Freeze original self-attention weights (for specified layers)
2. Inject 2 rank decomposition matrices whose product is same size matrix as original

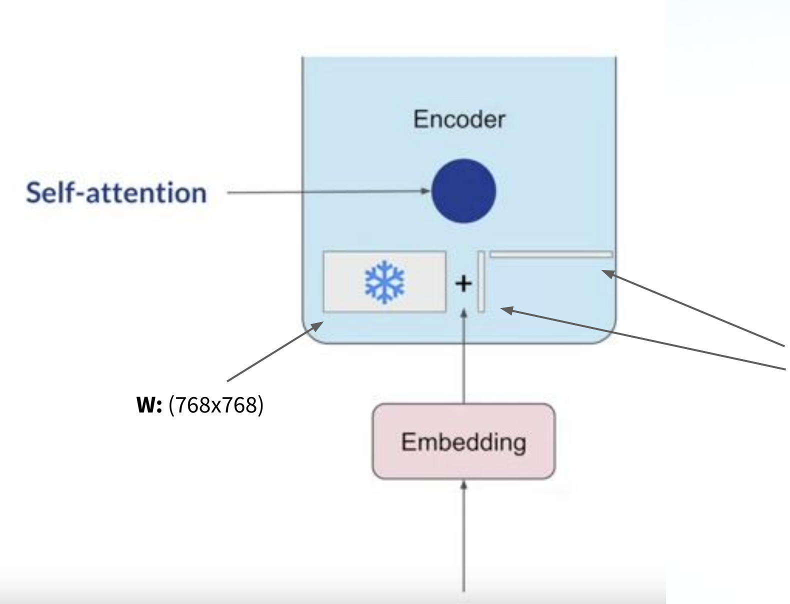
A: (16x768)
B: (768x16)

BA: (768x768)

r = 16

LoRA Overview

During training:



A: (16x768)
B: (768x16)

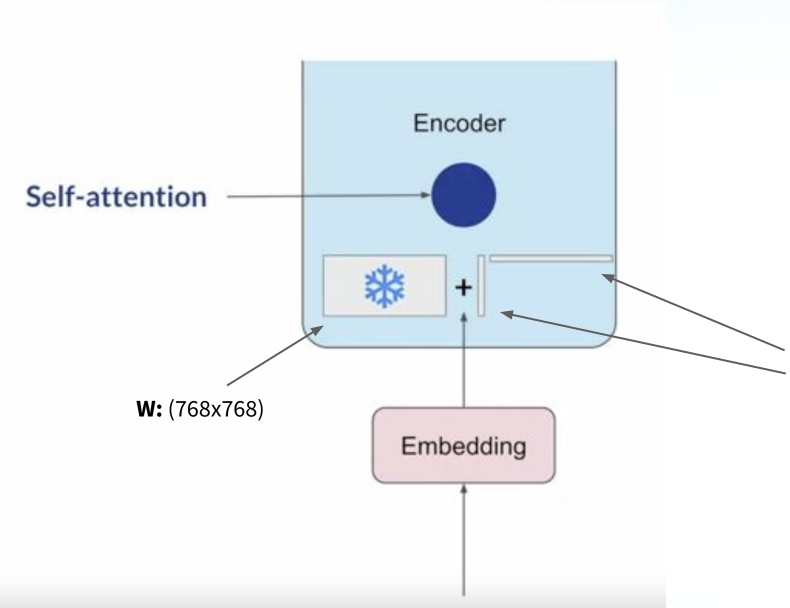
BA: (768x768)

r = 16

1. Freeze original self-attention weights (for specified layers)
2. Inject 2 **rank decomposition matrices** whose product is same size matrix as original
3. Train the weights of just these two low-rank matrices
   - Pass embedding matrix through each separately and add together

LoRA Overview

During training:



A: (16x768)
B: (768x16)

BA: (768x768)

r = 16

1. Freeze original self-attention weights (for specified layers)
2. Inject 2 rank decomposition matrices whose product is same size matrix as original
3. Train the weights of just these two low-rank matrices
    - Pass embedding matrix through each separately and add together

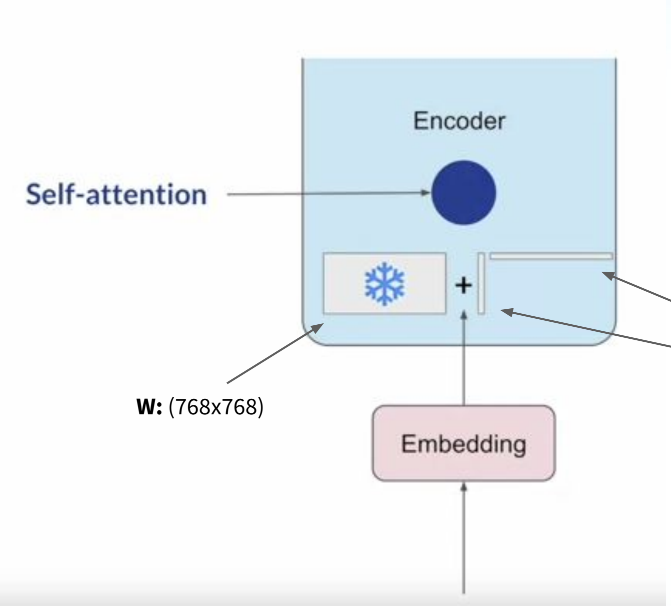
Number of Trainable Params:

- **W:** 590k
- **A, B:** 12k

95% less trainable params

LoRA Overview

At inference:

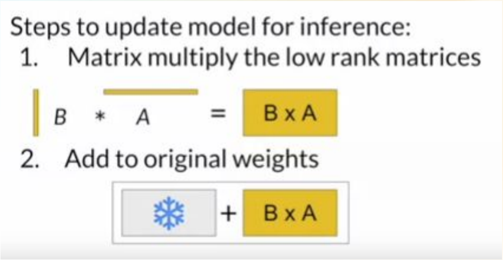


W: (768x768)

A: (16x768)
B: (768x16)

BA: (768x768)

r = 16

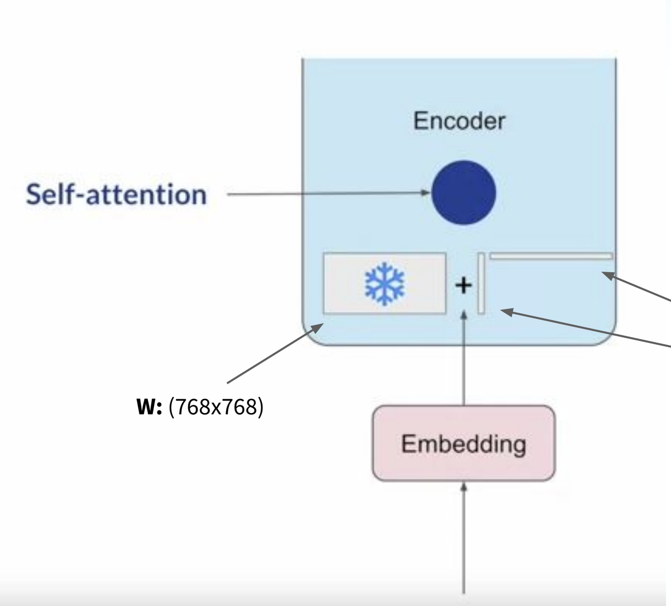


$$h = W(x) + BA(x)$$

$$h = (W + BA)(x)$$

LoRA Overview

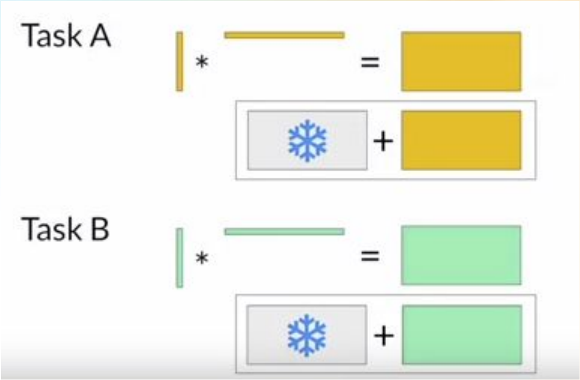
At inference:



A: (16x768)
B: (768x16)

BA: (768x768)

r = 16



$h = W(x) + BA(x)$

$h = (W + BA)(x)$

Easy to swap out LoRA adapters for different tasks!

LoRA Overview

Performance

Model & Method	# Trainable Parameters	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B	Avg.
RoB_{base} (FT)*	125.0M	$\mathbf{87.6}$	94.8	90.2	$\mathbf{63.6}$	92.8	$\mathbf{91.9}$	78.7	91.2	86.4
RoB_{base} (BitFit)*	0.1M	84.7	93.7	$\mathbf{92.7}$	62.0	91.8	84.0	81.5	90.8	85.2
RoB_{base} (Adpt^{D})*	0.3M	$87.1_{\pm.0}$	$94.2_{\pm.1}$	$88.5_{\pm1.1}$	$60.8_{\pm.4}$	$93.1_{\pm.1}$	$90.2_{\pm.0}$	$71.5_{\pm2.7}$	$89.7_{\pm.3}$	84.4
RoB_{base} (Adpt^{D})*	0.9M	$87.3_{\pm.1}$	$94.7_{\pm.3}$	$88.4_{\pm.1}$	$62.6_{\pm.9}$	$93.0_{\pm.2}$	$90.6_{\pm.0}$	$75.9_{\pm2.2}$	$90.3_{\pm.1}$	85.4
RoB_{base} (LoRA)	0.3M	$87.5_{\pm.3}$	$\mathbf{95.1}_{\pm.2}$	$89.7_{\pm.7}$	$63.4_{\pm1.2}$	$\mathbf{93.3}_{\pm.3}$	$90.8_{\pm.1}$	$\mathbf{86.6}_{\pm.7}$	$\mathbf{91.5}_{\pm.2}$	$\mathbf{87.2}$
$\text{RoB}_{\text{large}}$ (FT)*	355.0M	90.2	$\mathbf{96.4}$	$\mathbf{90.9}$	68.0	94.7	$\mathbf{92.2}$	86.6	92.4	88.9
$\text{RoB}_{\text{large}}$ (LoRA)	0.8M	$\mathbf{90.6}_{\pm.2}$	$96.2_{\pm.5}$	$\mathbf{90.9}_{\pm1.2}$	$\mathbf{68.2}_{\pm1.9}$	$\mathbf{94.9}_{\pm.3}$	$91.6_{\pm.1}$	$\mathbf{87.4}_{\pm2.5}$	$\mathbf{92.6}_{\pm.2}$	$\mathbf{89.0}$
$\text{RoB}_{\text{large}}$ (Adpt^{P})†	3.0M	$90.2_{\pm.3}$	$96.1_{\pm.3}$	$90.2_{\pm.7}$	$\mathbf{68.3}_{\pm1.0}$	$\mathbf{94.8}_{\pm.2}$	$\mathbf{91.9}_{\pm.1}$	$83.8_{\pm2.9}$	$92.1_{\pm.7}$	88.4
$\text{RoB}_{\text{large}}$ (Adpt^{P})†	0.8M	$\mathbf{90.5}_{\pm.3}$	$\mathbf{96.6}_{\pm.2}$	$89.7_{\pm1.2}$	$67.8_{\pm2.5}$	$\mathbf{94.8}_{\pm.3}$	$91.7_{\pm.2}$	$80.1_{\pm2.9}$	$91.9_{\pm.4}$	87.9
$\text{RoB}_{\text{large}}$ (Adpt^{H})†	6.0M	$89.9_{\pm.5}$	$96.2_{\pm.3}$	$88.7_{\pm2.9}$	$66.5_{\pm4.4}$	$94.7_{\pm.2}$	$92.1_{\pm.1}$	$83.4_{\pm1.1}$	$91.0_{\pm1.7}$	87.8
$\text{RoB}_{\text{large}}$ (Adpt^{H})†	0.8M	$90.3_{\pm.3}$	$96.3_{\pm.5}$	$87.7_{\pm1.7}$	$66.3_{\pm2.0}$	$94.7_{\pm.2}$	$91.5_{\pm.1}$	$72.9_{\pm2.9}$	$91.5_{\pm.5}$	86.4
$\text{RoB}_{\text{large}}$ (LoRA)†	0.8M	$\mathbf{90.6}_{\pm.2}$	$96.2_{\pm.5}$	$\mathbf{90.2}_{\pm1.0}$	$68.2_{\pm1.9}$	$\mathbf{94.8}_{\pm.3}$	$91.6_{\pm.2}$	$\mathbf{85.2}_{\pm1.1}$	$\mathbf{92.3}_{\pm.5}$	$\mathbf{88.6}$
DeB_{XXL} (FT)*	1500.0M	91.8	$\mathbf{97.2}$	92.0	72.0	$\mathbf{96.0}$	92.7	93.9	92.9	91.1
DeB_{XXL} (LoRA)	4.7M	$\mathbf{91.9}_{\pm.2}$	$96.9_{\pm.2}$	$\mathbf{92.6}_{\pm.6}$	$\mathbf{72.4}_{\pm1.1}$	$\mathbf{96.0}_{\pm.1}$	$\mathbf{92.9}_{\pm.1}$	$\mathbf{94.9}_{\pm.4}$	$\mathbf{93.0}_{\pm.2}$	$\mathbf{91.3}$

Table 2: $\text{RoBERTa}_{\text{base}}$, $\text{RoBERTa}_{\text{large}}$, and $\text{DeBERTa}_{\text{XXL}}$ with different adaptation methods on the GLUE benchmark. We report the overall (matched and mismatched) accuracy for MNLI, Matthew's correlation for CoLA, Pearson correlation for STS-B, and accuracy for other tasks. Higher is better for all metrics. * indicates numbers published in prior works. † indicates runs configured in a setup similar to Houlsby et al. (2019) for a fair comparison.

Model & Method	# Trainable Parameters	E2E NLG Challenge BLEU	NIST	MET	ROUGE-L	CIDEr
GPT-2 M (FT)*	354.92M	68.2	8.62	46.2	71.0	2.47
GPT-2 M ($\text{Adapter}^{\text{L}}$)*	0.37M	66.3	8.41	45.0	69.8	2.40
GPT-2 M ($\text{Adapter}^{\text{L}}$)*	11.09M	68.9	8.71	46.1	71.3	2.47
GPT-2 M ($\text{Adapter}^{\text{H}}$)	11.09M	$67.3_{\pm.6}$	$8.50_{\pm.07}$	$46.0_{\pm.2}$	$70.7_{\pm.2}$	$2.44_{\pm.01}$
GPT-2 M (FT^{Top2})*	25.19M	68.1	8.59	46.0	70.8	2.41
GPT-2 M (PreLayer)*	0.35M	69.7	8.81	46.1	71.4	2.49
GPT-2 M (LoRA)	0.35M	$\mathbf{70.4}_{\pm.1}$	$\mathbf{8.85}_{\pm.02}$	$\mathbf{46.8}_{\pm.2}$	$\mathbf{71.8}_{\pm.1}$	$\mathbf{2.53}_{\pm.02}$
GPT-2 L (FT)*	774.03M	68.5	8.78	46.0	69.9	2.45
GPT-2 L ($\text{Adapter}^{\text{L}}$)	0.88M	$69.1_{\pm.1}$	$8.68_{\pm.03}$	$46.3_{\pm.0}$	$71.4_{\pm.2}$	$\mathbf{2.49}_{\pm.0}$
GPT-2 L ($\text{Adapter}^{\text{L}}$)	23.00M	$68.9_{\pm.3}$	$8.70_{\pm.04}$	$46.1_{\pm.1}$	$71.3_{\pm.2}$	$2.45_{\pm.02}$
GPT-2 L (PreLayer)*	0.77M	70.3	8.85	46.2	71.7	2.47
GPT-2 L (LoRA)	0.77M	$\mathbf{70.4}_{\pm.1}$	$\mathbf{8.89}_{\pm.02}$	$\mathbf{46.8}_{\pm.2}$	$\mathbf{72.0}_{\pm.2}$	$2.47_{\pm.02}$

Table 3: GPT-2 medium (M) and large (L) with different adaptation methods on the E2E NLG Challenge. For all metrics, higher is better. LoRA outperforms several baselines with comparable or fewer trainable parameters. Confidence intervals are shown for experiments we ran. * indicates numbers published in prior works.

LoRA Overview

Choosing the rank parameter

Rank r	val_loss	BLEU	NIST	METEOR	ROUGE_L	CIDEr
1	1.23	68.72	8.7215	0.4565	0.7052	2.4329
2	1.21	69.17	8.7413	0.4590	0.7052	2.4639
4	1.18	**70.38**	**8.8439**	**0.4689**	0.7186	**2.5349**
8	1.17	69.57	8.7457	0.4636	**0.7196**	2.5196
16	**1.16**	69.61	8.7483	0.4629	0.7177	2.4985
32	**1.16**	69.33	8.7736	0.4642	0.7105	2.5255
64	**1.16**	69.24	8.7174	0.4651	0.7180	2.5070
128	**1.16**	68.73	8.6718	0.4628	0.7127	2.5030
256	**1.16**	68.92	8.6982	0.4629	0.7128	2.5012
512	**1.16**	68.78	8.6857	0.4637	0.7128	2.5025
1024	1.17	69.37	8.7495	0.4659	0.7149	2.5090

Table 18: Validation loss and test set metrics on E2E NLG Challenge achieved by LoRA with different rank r using GPT-2 Medium. Unlike on GPT-3 where $r = 1$ suffices for many tasks, here the performance peaks at $r = 16$ for validation loss and $r = 4$ for BLEU, suggesting the GPT-2 Medium has a similar intrinsic rank for adaptation compared to GPT-3 175B. Note that some of our hyperparameters are tuned on $r = 4$, which matches the parameter count of another baseline, and thus might not be optimal for other choices of r.

LoRA Overview

Best practices

Highlights:

- Comparable evaluation performance than full fine-tuning on a variety of tasks
  - Images (diffusion models for different “styles”)
  - Audio (fine-tuning whisper large on a new language)
  - Text (fine-tuning language models)
- No additional inference latency
- Hyper-parameters:
  - Higher learning rate than full fine-tuning (order of magnitude of 10-100x)
  - Rank between 4-16
  - Lora alpha: usually 2-4x the LoRA rank

More details:

- Empirical findings and best practices with PEFT approaches

QLoRA Overview

Frozen model is 4-bit quantized

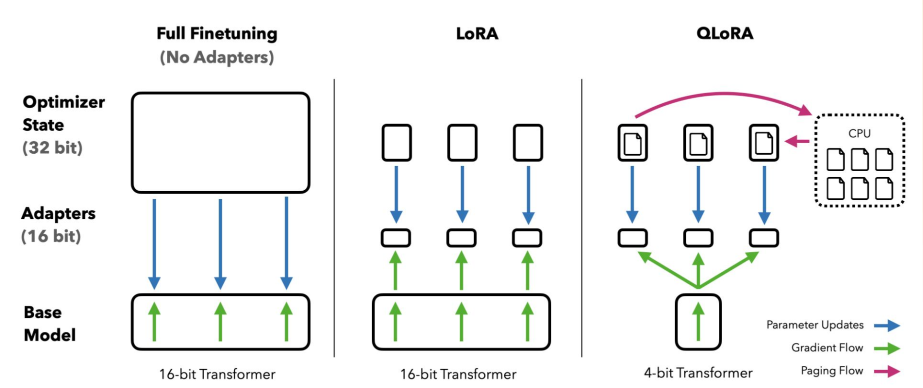
- 30B model on 24GB GPU
- 65B model on 48GB GPU

Uses Paged Optimizers that allow offload to CPU RAM when spikes happen

- Prevents error and loss of training in middle of run

Only for fine-tuning, not for pre-training

[Empirically demonstrated] to preserve full fine-tuning task performance



Thank you!