

# Finetuning Large Language Models

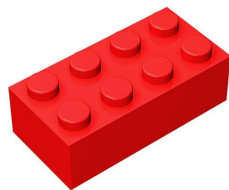
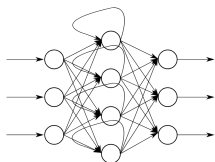
Caglar Gulcehre and Volkan Cevher

- **Lecture 1:** Recap, What is finetuning? ICL, Prompt tuning,...
- **Lecture 2:** Instruction tuning, SFT and Toolformer
- **Lecture 3:** Parameter efficient finetuning (PEFT), Quantization, and Pruning

## Quick Recap

# Basic Building Blocks

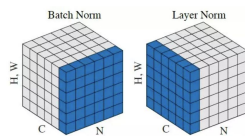
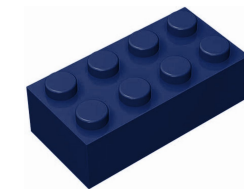
Recurrence



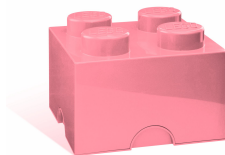
Self-Attention



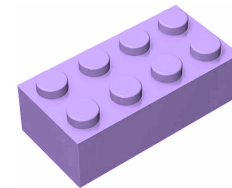
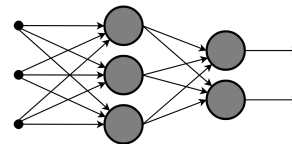
Cross-attention



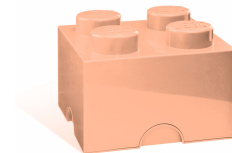
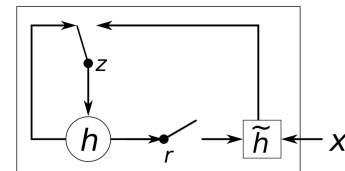
Positional Encodings



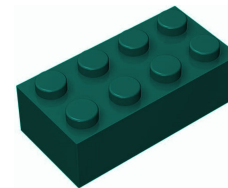
MLP



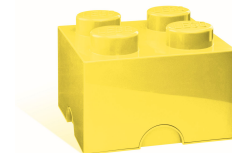
Gates



Normalization Layers



Residual connections







*Despite our preference to believe algorithms succeed or fail in isolation, history tells us that most computer science breakthroughs follow the Anna Karenina principle. Successful breakthroughs are often distinguished from failures by benefiting from multiple criteria aligning surreptitiously. For computer science research, this often depends upon winning what this essay terms the hardware lottery — avoiding possible points of failure in downstream hardware and software choices.*

## The Hardware Lottery

Sara Hooker

Google Research, Brain Team  
shooker@google.com

### Abstract

Hardware, systems and algorithms research communities have historically had different incentive structures and fluctuating motivation to engage with each other explicitly. This historical treatment is odd given that hardware and software have frequently determined which research ideas succeed (and fail). This essay introduces the term hardware lottery to describe when a research idea wins because it is suited to the available software and hardware and *not* because the idea is superior to alternative research directions. Examples from early computer science history illustrate how hardware lotteries can delay research progress by casting successful ideas as failures. These lessons are particularly salient given the advent of domain specialized hardware which make it increasingly costly to stray off of the beaten path of research ideas. This essay posits that the gains from progress in computing are likely to become even more uneven, with certain research directions moving into the fast-lane while progress on others is further obstructed.

# Bitter lesson

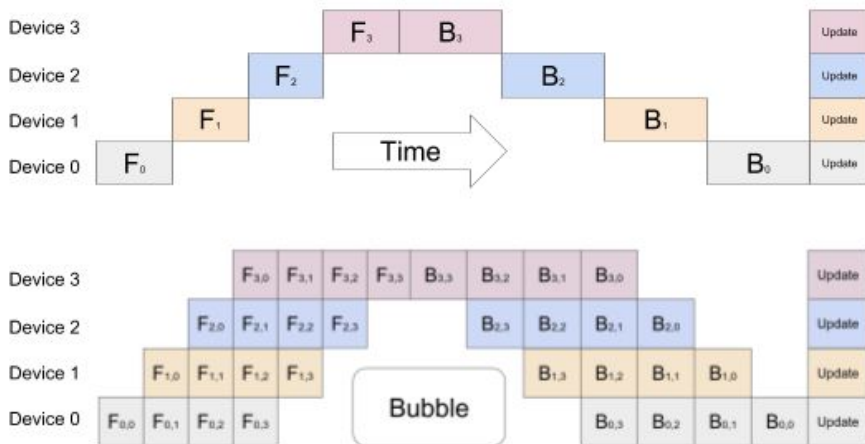
The biggest lesson that can be read from 70 years of AI research is that general methods that leverage computation are ultimately the most effective, and by a large margin...



The bitter lesson is based on the historical observations that:

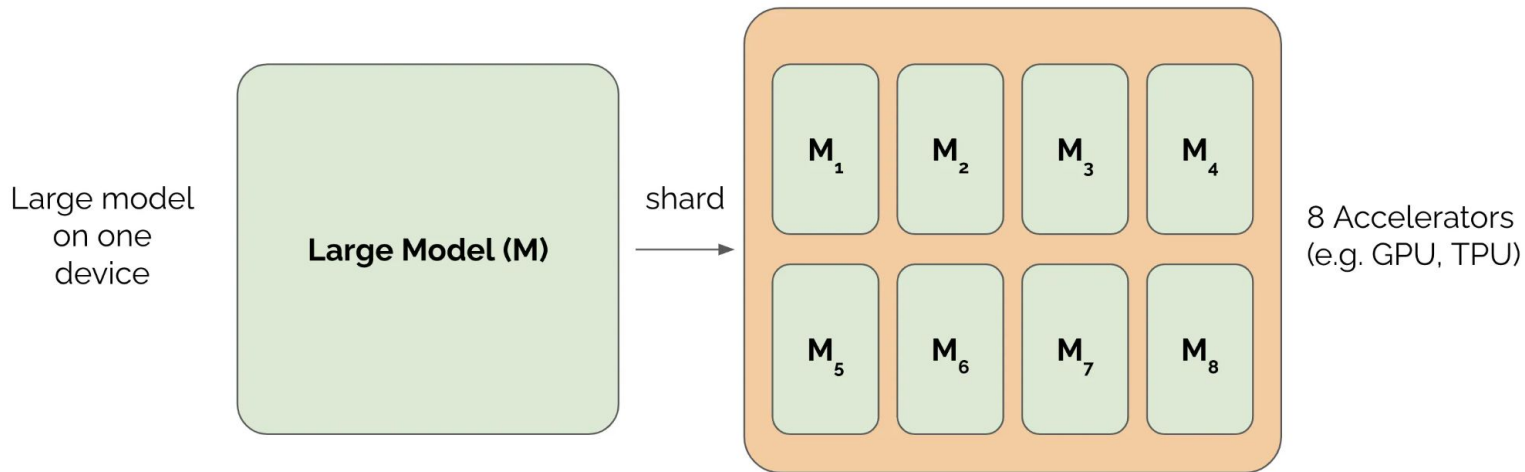
- 1) AI researchers have often tried to build knowledge into their agents,
- 2) this always helps in the short term, and is personally satisfying to the researcher, but
- 3) in the long run it plateaus and even inhibits further progress, and
- 4) breakthrough progress eventually arrives by an opposing approach based on scaling computation by search and learning

# Pipeline parallelism



*Top: The naive model parallelism strategy leads to severe underutilization due to the sequential nature of the network. Only one accelerator is active at a time. Bottom: GPipe divides the input mini-batch into smaller micro-batches, enabling different accelerators to work on separate micro-batches at the same time.*

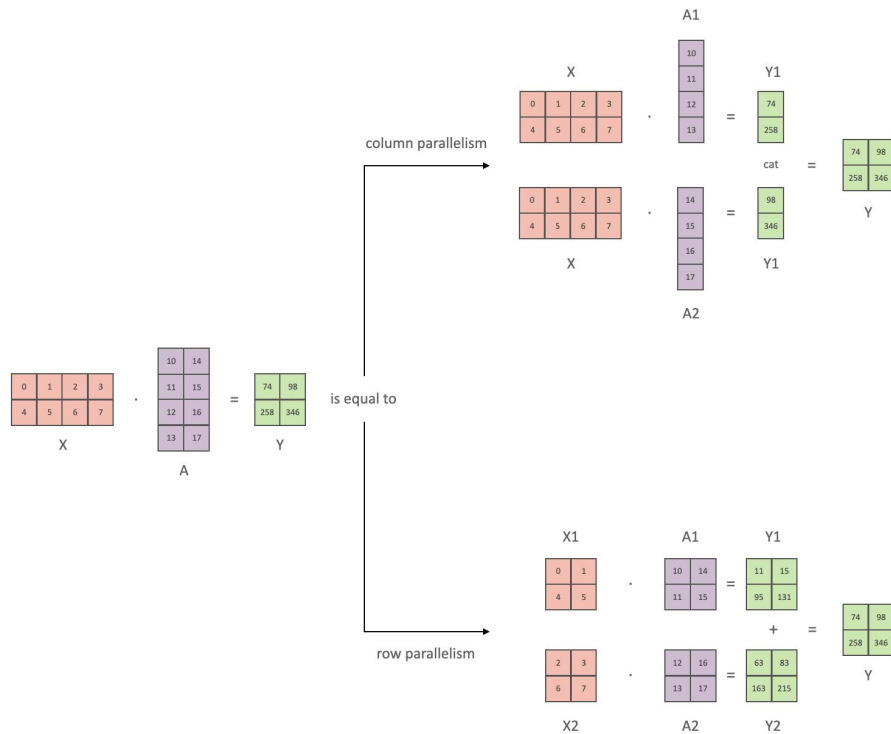
## Tensor parallelism

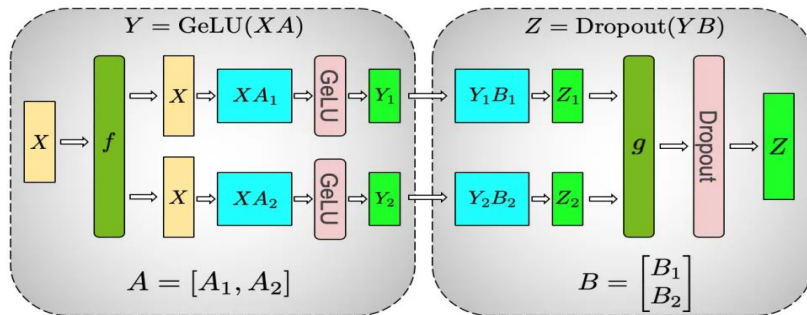


**Q:** How do we actually split the model?

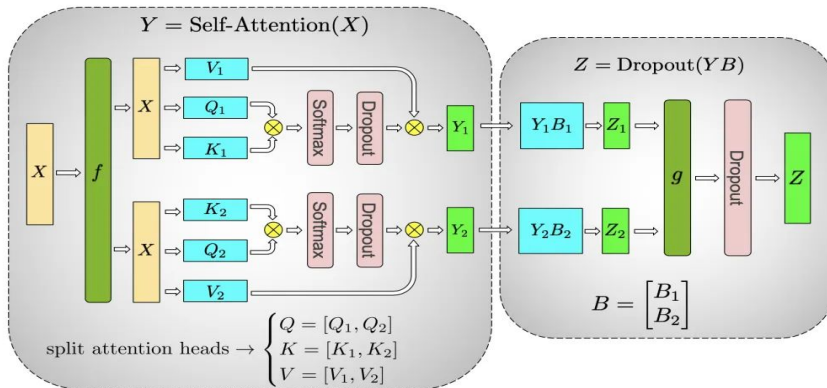
**Source:** [https://www.mishalaskin.com/posts/tensor\\_parallel](https://www.mishalaskin.com/posts/tensor_parallel)

# Tensor Parallelism-Megatron





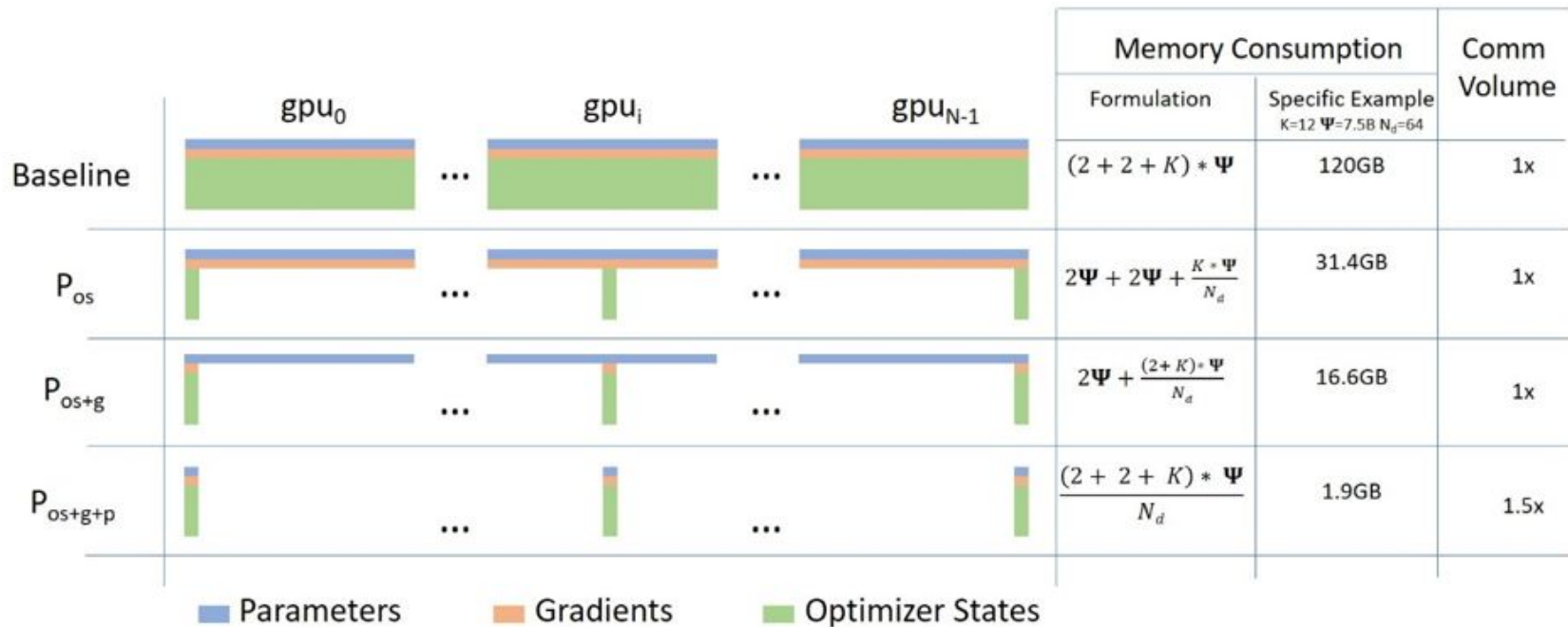
(a) MLP



(b) Self-Attention

**Figure 3.** Blocks of Transformer with Model Parallelism.  $f$  and  $g$  are conjugate.  $f$  is an identity operator in the forward pass and all reduce in the backward pass while  $g$  is an all reduce in the forward pass and identity in the backward pass.

# Zero (Rajbhandari et al. 2019)



*Zero enables data replicas to store parts of the model and optimizer state not currently in use.*



# Scaling laws for LLMs

Given:

1. **N:** The number of parameters you put in the model.
2. **D:** The total number of tokens being trained on.

For example, to answer:

- If we had 10x or 1000x more data, what would change?
- If we had 10x or 1000x more compute, what would change?
- How much data (or compute) is needed to really move the needle? Does enough even exist?

# Scaling laws - Chinchilla laws (Hoffman et al., 2022)

Scaled up to a model of size 70B params with 1.4T tokens.

Addresses some of the issue with the original scaling laws paper by Kaplan et al 2019:

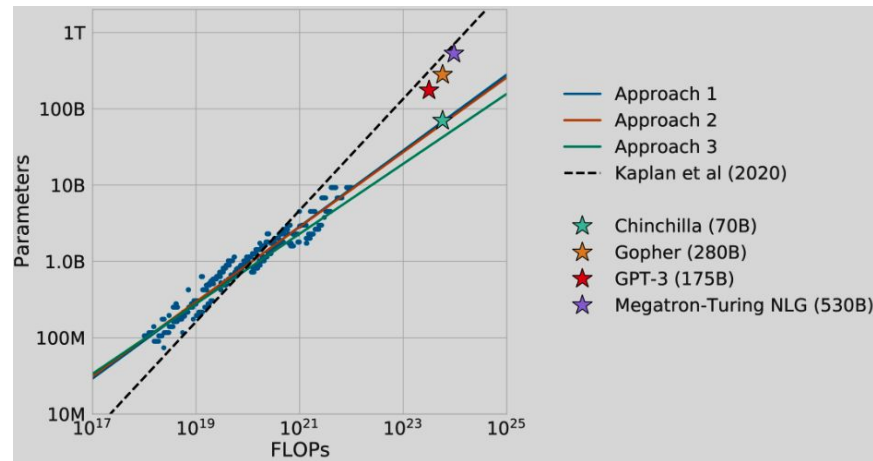
- Assumes fixed hparams, specifically LR schedules.

Chinchilla revisits the question:

- Given fixed compute, how to tradeoff param/data?

## TLDR:

N (# params) and D (# data) should be scaled roughly with a constant factor, e.g. 1B at 20B tokens to 30B at 600B tokens.



The water can get  
murky sometimes...



Mid-training can be thought as continual pre-training of an LLM.

It is not very well-known or appreciated in the academic community yet but a very important part of the LLM training and widely adapted in the industry.

Mid-training is the point where the boundary between the pre-training and post-training becomes blurry.

Possible to see it as finetuning the pre-trained model on the high-quality data that is sometimes augmented with the synthetic datasets.

Can be considered as a curriculum learning.

There are two recipes we will talk about next for:

- OLMO 1.7 recipe by [AllenAI](#).
- OLMO 2 recipe by [OLMO team](#).

# Mid-training OLMO recipe

OLMO 1.7 for instance had two stages of training:

- In the finetuning stage, they trained on Dolma 1.7 dataset over 2T tokens.
- Finetune the model on a high-quality subset with a small learning rate. With a mixture including sources like:
  - Wikipedia
  - OpenWebMath

Flan	OpenWeb Math	Stack Exchange	StarCoder	Dolma Reddit	Refined Web	C4	Algebraic Stack	Semantic Scholar	Project Gutenberg	Wikipedia Wikibooks	Total
8.0 G	6.2 G	5.6 G	5.6 G	4.6 G	4.4 G	4.4 G	3.5 G	3.3 G	2.6 G	1.8 G	50 G
16.0 %	12.4 %	11.2 %	11.2 %	9.2 %	8.8 %	8.8 %	7.0 %	6.6 %	5.2 %	3.6 %	100 %

# Mid-training OLMO 2 Recipe

After the initial pre-training on mostly web-data, we would further train the model on a mixture of web data that is heavily filtered for quality and then they use a collection of high-quality data that is mainly synthetically generated.

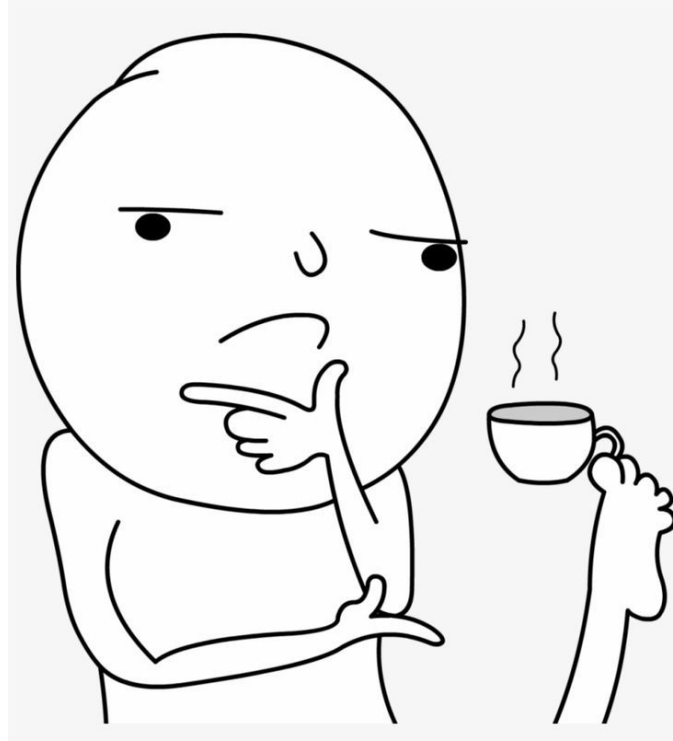
The model used to generate the datasets change based on the task. For example, they used the best open-weights model for a specific task of interest.

They call their mid-training dataset as DolmaMix and used an approach called micro-annealing on math datasets.

Source	Type	Tokens	Words	Bytes	Docs
Mid-Training ♦ Dolmino High Quality Subset					
DCLM-Baseline FastText top 7% FineWeb ≥ 2	High quality web	752B	670B	4.56T	606M
FLAN from Dolma 1.7 decontaminated	Instruction data	17.0B	14.4B	98.2B	57.3M
peS2o from Dolma 1.7	Academic papers	58.6B	51.1B	413B	38.8M
Wikipedia & Wikibooks from Dolma 1.7	Encyclopedic	3.7B	3.16B	16.2B	6.17M
Stack Exchange 09/30/2024 dump curated Q&A data	Q&A	1.26B	1.14B	7.72B	2.48M
High quality total		832.6B	739.8B	5.09T	710.8M
Mid-training ♦ Dolmino Math Mix					
TuluMath	Synthetic math	230M	222M	1.03B	220K
Dolmino SynthMath	Synthetic math	28.7M	35.1M	163M	725K
TinyGSM-MIND	Synthetic math	6.48B	5.68B	25.52B	17M
MathCoder2 Synthetic Ajibawa-2023 M-A-P Matrix	Synthetic Math	3.87B	3.71B	18.4B	2.83M
Metamath OWM-filtered	Math	84.2M	76.6M	741M	383K
CodeSearchNet OWM-filtered	Code	1.78M	1.41M	29.8M	7.27K
GSM8K Train split	Math	2.74M	3.00M	25.3M	17.6K
Math total		10.7B	9.73B	45.9B	21.37M

# What is finetuning?

# Why not just use pre-trained LLMs?





# Why not just use pre-trained LLMs?

The pre-trained models are just generalized next-token predictors trained on vast amount corpus.

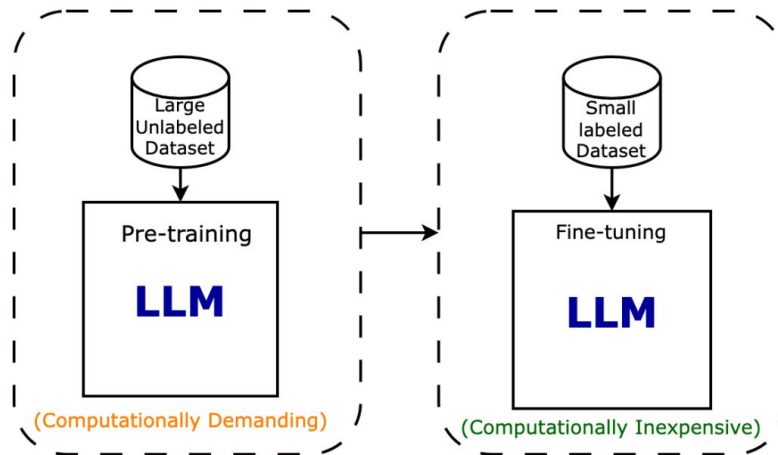
They can be:

- Biased.
- Hallucinate.
- Not necessarily aware of tasks or goal.
- Can output harmful/toxic content.
- Don't necessarily follow instructions.
- Not good at conversations.

# What is finetuning?

Purpose of the finetuning LLMs is to adapt the pre-trained base LLMs for the specific tasks or domains, enhancing their performance and improving their relevance to the applications of interest.

- For example, making them a conversational model.
- Finetuning LLMs is a way to make them useful in real-world applications.



# Key-purposes of finetuning



# Key-purposes of finetuning

Key purposes of finetuning LLMs include:

- **Task specific adaptation:** Can we finetune the model on data collected for a specific task to improve their performance on that task? Such as translation, math, coding and reasoning.
- **Domain-specific expertise:** It tailors the model to understand and generate content for particular industries or fields, like healthcare, finance, or legal domains.
- **Safety:** e.g. debiasing the models, removing toxic behaviors.
- **Improved abilities** and performance.
- **Tool-use:** for example, the ability to use calculator, search engine or code interpreter.
- **Reducing hallucinations**

# In-context learning (ICL) in LLMs

ICL is an emergent property of LLMs where the model learns to perform a task that it hasn't seen during training just by observing input-output pairs in its context without any parameter optimization:

The three settings we explore for in-context learning

## Zero-shot

The model predicts the answer given only a natural language description of the task. No gradient updates are performed.

```
1 Translate English to French: ← task description
2 cheese => ..... ← prompt
```

## One-shot

In addition to the task description, the model sees a single example of the task. No gradient updates are performed.

```
1 Translate English to French: ← task description
2 sea otter => loutre de mer ← example
3 cheese => ..... ← prompt
```

## Few-shot

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.

```
1 Translate English to French: ← task description
2 sea otter => loutre de mer ← examples
3 peppermint => menthe poivrée ←
4 plush girafe => girafe peluche ←
5 cheese => ..... ← prompt
```

Traditional fine-tuning (not used for GPT-3)

## Fine-tuning

The model is trained via repeated gradient updates using a large corpus of example tasks.

```
1 sea otter => loutre de mer ← example #1
↓
gradient update
↓
1 peppermint => menthe poivrée ← example #2
↓
gradient update
↓
...
↓
1 plush giraffe => girafe peluche ← example #N
↓
gradient update
↓
1 cheese => ..... ← prompt
```

Figure is taken from GPT-3 paper, “Language models are few shot learners”.

# What can in-context learning do?

On many benchmark NLP benchmarks, in-context learning is competitive with models trained with much more labeled data and is state-of-the-art on LAMBADA (commonsense sentence completion) and TriviaQA (question answering).

In-context learning typically requires few examples only.

Circulation revenue has increased by 5%  
in Finland. // Positive

Panostaja did not disclose the purchase  
price. // Neutral

Paying off the national debt will be  
extremely painful. // Negative

The company anticipated its operating  
profit to improve. // \_\_\_\_\_

LM

Circulation revenue has increased by  
5% in Finland. // Finance

They defeated ... in the NFC  
Championship Game. // Sports

Apple ... development of in-house  
chips. // Tech

The company anticipated its operating  
profit to improve. // \_\_\_\_\_

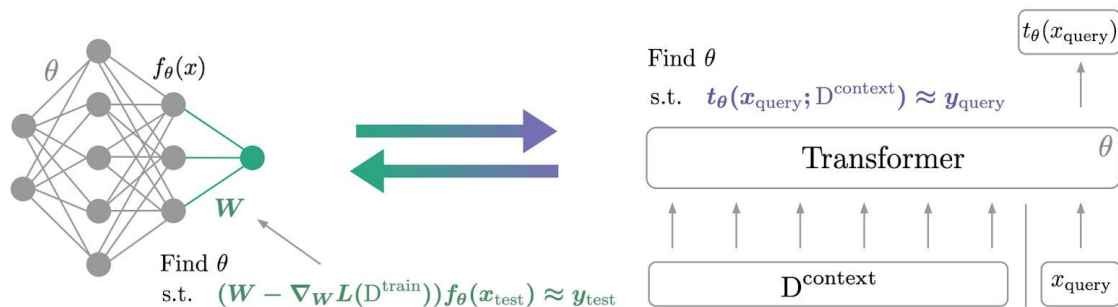
LM

# In-context learning and SGD

Here, we will explore the equivalence of SGD updates ICL between data transformations induced by:

- Single attention layer.
- Gradient descent (GD) on a regression loss.

Our notation and derivations are based on van Oswald et. al, 2022.



# In-context learning - Setup

Assume  $N$  training pairs  $\{(x_i, y_i)\}_{i=1}^N$  where  $x_i \in \mathbb{R}^{N_x}$  and  $y_i \in \mathbb{R}^{N_y}$  with a single test query  $(x_{test}, y_{test})$ . We can merge each pair into a single query:

$$e_i = (x_i, y_i), \text{ for } i = (1, \dots, N) \in \mathbb{R}^{N_x + N_y}$$

and for query token, we set:

$$e_{N+1} = (x_{test}, -W_0 x_{test})$$

Here  $W_0$  is the "initial" weight matrix (often set to zero) before performing the gradient descent updates.



# In-context learning - Setup

Assume  $N$  training pairs  $\{(x_i, y_i)\}_{i=1}^N$  where  $x_i \in \mathbb{R}^{N_x}$  and  $y_i \in \mathbb{R}^{N_y}$  with a single test query  $(x_{test}, y_{test})$ . We can merge each pair into a single query:

$$e_i = (x_i, y_i), \text{ for } i = (1, \dots, N) \in \mathbb{R}^{N_x + N_y}$$

and for query token, we set:

$$e_{N+1} = (x_{test}, -W_0 x_{test})$$

Here  $W_0$  is the "initial" weight matrix (often set to zero) before performing the gradient descent updates.

# In-context learning-Gradient descent on Linear model

Given a linear model as:

$$y = Wx$$

under the MSE regression loss:

$$L(W) = \frac{1}{2N} \sum_{i=1}^N \|Wx_i - y_i\|_2^2$$

A single-step of gradient with learning rate  $\eta$  updates  $W$ :

$$\Delta W = -\eta \nabla_W L(W) = -\frac{\eta}{N} \sum_{i=1}^N (Wx_i - y_i) x_i^\top.$$

Hence, the new weights would be:  $W_{new} = W + \Delta W$ .

# In-context learning-Gradient descent on Linear model

Given a linear model as:

$$y = Wx$$

under the MSE regression loss:

$$L(W) = \frac{1}{2N} \sum_{i=1}^N \|Wx_i - y_i\|_2^2$$

A single-step of gradient with learning rate  $\eta$  updates  $W$ :

$$\Delta W = -\eta \nabla_W L(W) = -\frac{\eta}{N} \sum_{i=1}^N (Wx_i - y_i) x_i^\top.$$

Hence, the new weights would be:  $W_{new} = W + \Delta W$ .

# In-context learning-Linear Attention

■  
Consider a single-head self-attention layer of the following form:

$$e_j \leftarrow e_j + PV(K^\top q_j),$$

for  $P$  being a linear projection matrix and each token will be mapped into learnable "query", "key", "value" matrices are typically called:

$$q_j = W_Q e_j, k_j = W_K e_j, v_j = W_V e_j.$$

then combined in the usual self-attention formula.

We can write the self-attention formula as:

$$e_j \leftarrow e_j + P \sum_{i=1}^N v \otimes k q_j = e_j + PW_v \sum_{i=1}^N e_i \otimes e_i W_K^\top W_Q e_j.$$

where  $\otimes$  represents the [Outer product](#) between two vectors.

# Linear attention to GD - Choosing parameters

For instance, write each token  $e_j$  as  $(x_i, y_i) \in \mathbb{R}^{N_x + N_y}$ . Then define:

$$W_K = W_Q = \begin{pmatrix} I_{N_x} & 0 \\ 0 & 0 \end{pmatrix},$$

As this ensures that the keys, and queries look into "x" or input part of each token.

We can choose the weights of values as:

$$W_V = \begin{pmatrix} 0 & 0 \\ W_0 & -I_{N_y} \end{pmatrix},$$

Here,  $W_0$  is the initial linear model weight being finetuned and  $-I_{N_y}$  subtracts the target from the value.

Moreover, we can define the projection matrix  $P$  as follows:

$$P = \frac{\eta}{N} I_{N_x + N_y}$$

Let us put the pieces together and put the construction of the key, value and query matrices together:

$$\begin{pmatrix} x_j \\ y_j \end{pmatrix} \leftarrow \begin{pmatrix} x_j \\ y_j \end{pmatrix} + \frac{\eta}{N} I \sum_i \left( \begin{pmatrix} 0 & 0 \\ W_0 & -I_y \end{pmatrix} \begin{pmatrix} x_i \\ y_i \end{pmatrix} \right) \otimes \left( \begin{pmatrix} I_x & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} x_i \\ y_i \end{pmatrix} \right) \begin{pmatrix} I_x & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} x_j \\ y_j \end{pmatrix}$$

The left-hand side of this update function can be further simplified to:

$$\begin{pmatrix} x_j \\ y_j \end{pmatrix} \leftarrow \begin{pmatrix} x_j \\ y_j \end{pmatrix} + \frac{\eta}{N} I \sum_i \begin{pmatrix} 0 \\ W_0 x_i - y_i \end{pmatrix} \otimes \begin{pmatrix} x_i \\ 0 \end{pmatrix} \begin{pmatrix} x_j \\ 0 \end{pmatrix} = \begin{pmatrix} x_j \\ y_j \end{pmatrix} + \begin{pmatrix} 0 \\ -\Delta W x_j \end{pmatrix}.$$

# ICL as a Bayesian Inference - What is Bayesian Inference?

## Core idea:

- Updates the beliefs about unknown parameters (posterior) of a model according to the Bayes' theorem.

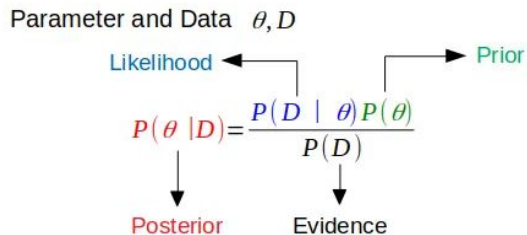
Given observed data  $D$  and a model with parameters  $\theta$ , Bayesian inference is based on **Bayes' theorem**:

$$P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)}$$

where:

- $P(\theta)$  is the **prior distribution**, encoding prior beliefs about  $\theta$  before observing data.
- $P(D|\theta)$  is the **likelihood**, describing how probable the data is given a parameter value.
- $P(D)$  (marginal likelihood or evidence) is the probability of the data under all possible parameter values.
- $P(\theta|D)$  is the **posterior distribution**, representing updated beliefs about  $\theta$  after seeing data.

# ICL as a Bayesian Inference - What is Bayesian Inference?



$$P(\theta | x) = \frac{P(x | \theta) \cdot P(\theta)}{\int_{\theta} P(x | \theta) \cdot P(\theta) d\theta}$$

Handwritten annotations in red:

- ↗ posterior (pointing to  $P(\theta | x)$ )
- ↗ sampling (pointing to  $P(x | \theta)$ )
- ↗ prior (pointing to  $P(\theta)$ )
- normalizing constant (pointing to the denominator integral)

Handwritten annotation in blue:

- for every possible  $\theta$  (pointing to the integration variable  $\theta$  in the denominator)

$$p(\text{hypothesis} | \text{data}) \propto p(\text{data} | \text{hypothesis}) p(\text{hypothesis})$$

In short, we can summarize the Bayes rule as which leads to the Bayesian updating as follows:

$$\text{posterior} \propto \text{prior} \times \text{likelihood}$$



# ICL as a Bayesian Inference - Marginalizing over concepts

[Xie et al. 2022](#), has shown a way to relate ICL to Bayesian Inference.

**Hypothesis:** Text documents in the pre-training has “long-coherence.” Namely, sentences or paragraphs in the same document share the same underlying concepts/topics or formatting.

- **During pre-training:** Learn to model distribution  $p$ , as document is generated by first sampling a latent concept, and then the document is generated by conditioning on the latent concept. LM must infer (“locate”) the latent concept for the document using evidence from the previous sentences.
- **ICL:** If the LM also infers the prompt concept (the latent concept shared by examples in the prompt) using in-context examples in the prompt, then in-context learning occurs!

**Remark:** The process of locating learned abilities can be viewed as Bayesian inference of a prompt concept that every example in a prompt shares.

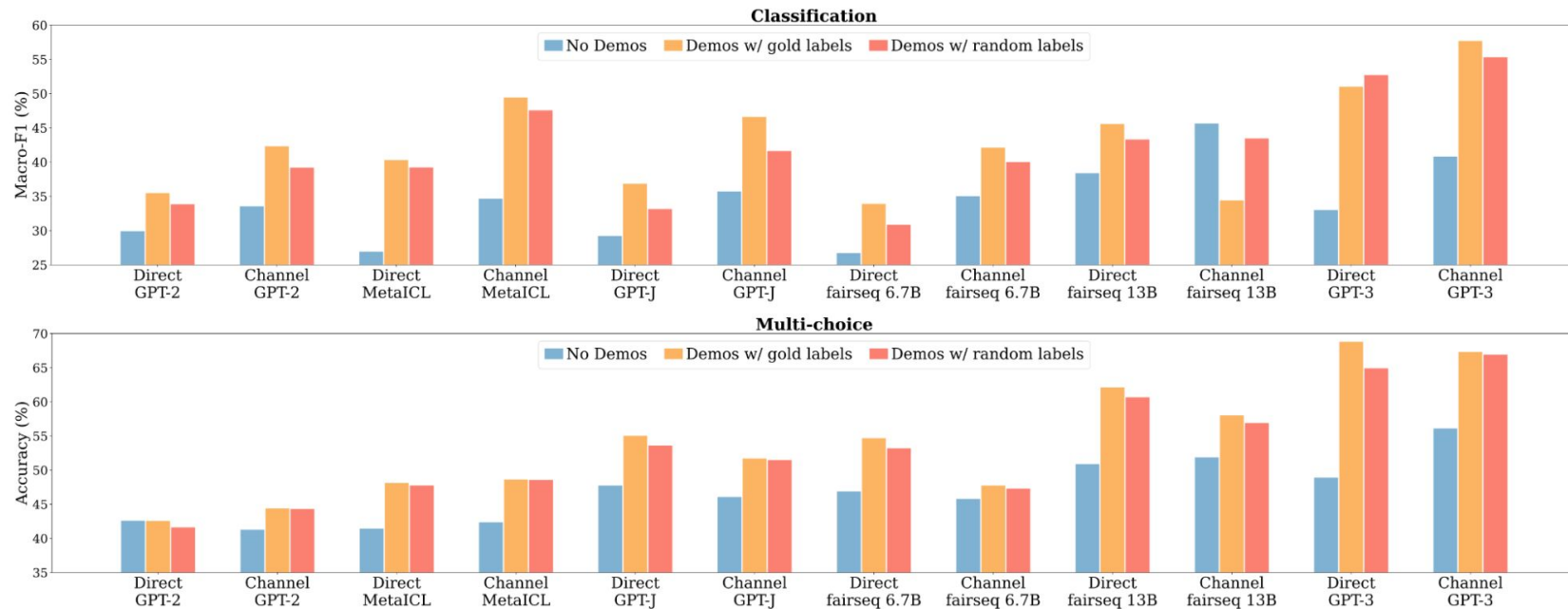
Mathematically, the prompt provides evidence for the model ( $p$ ) to sharpen the posterior distribution over concepts:  $p(\text{concept}|\text{prompt})$

$$p(\text{output}|\text{prompt}) = \int_{\text{concept}} p(\text{output}|\text{concept}, \text{prompt}) p(\text{concept}|\text{prompt}) d(\text{concept})$$

Ideally,  $p(\text{concept}|\text{prompt})$  concentrates on the prompt concept with more examples in the prompt so that the prompt concept is “selected” through marginalization.

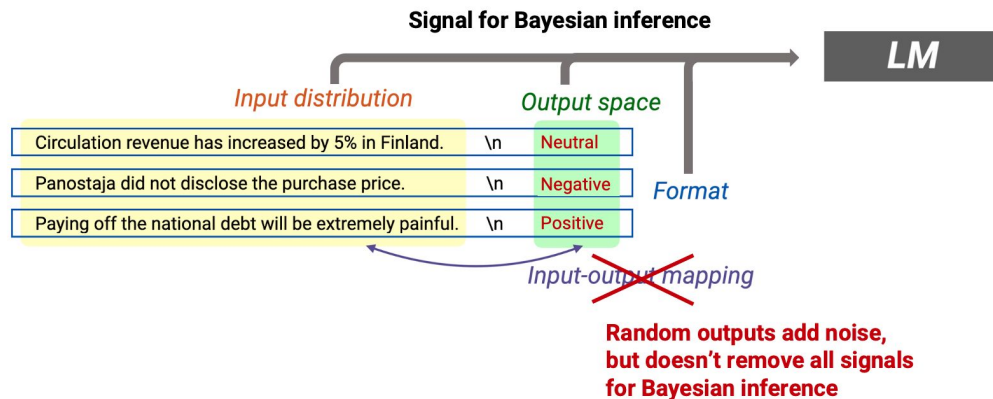
# ICL as a Bayesian Inference - Results

## Random labels in few-shot examples work



# ICL as a Bayesian Inference - Results

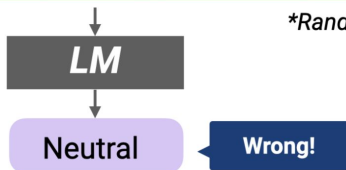
## Random sentences hurt performance



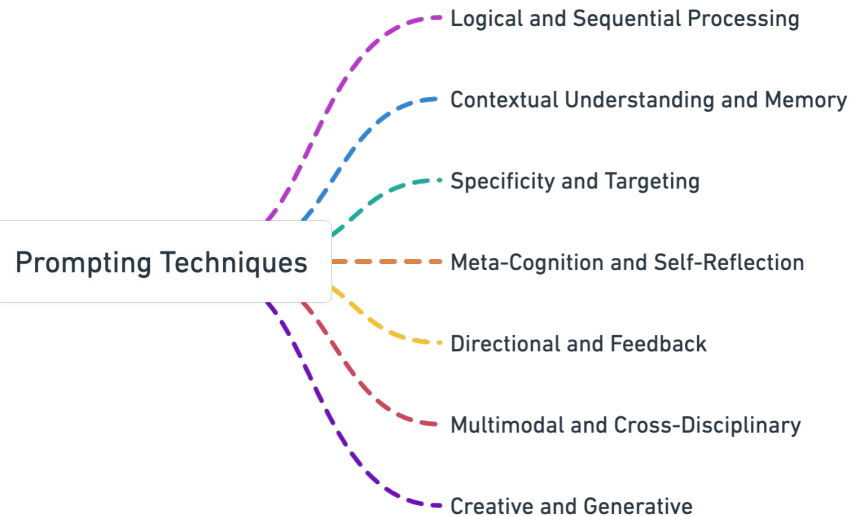
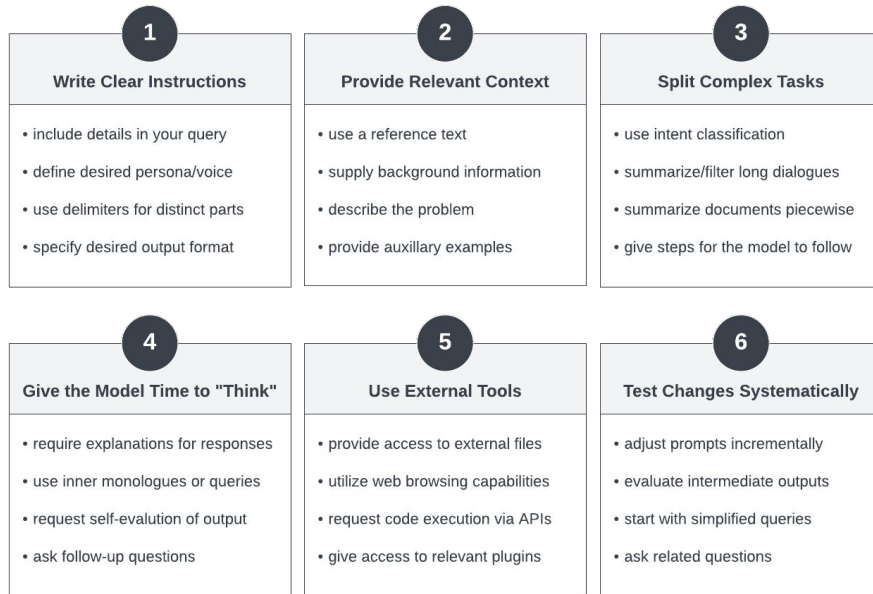
Up to 16% worse in absolute perf.

Circulation revenue has increased by 5% in Finland.	\n	Unanimity
Panostaja did not disclose the purchase price.	\n	Wave
Paying off the national debt will be extremely painful.	\n	Guana
The company anticipated its operating profit to improve. \n _____		

\*Random English unigrams



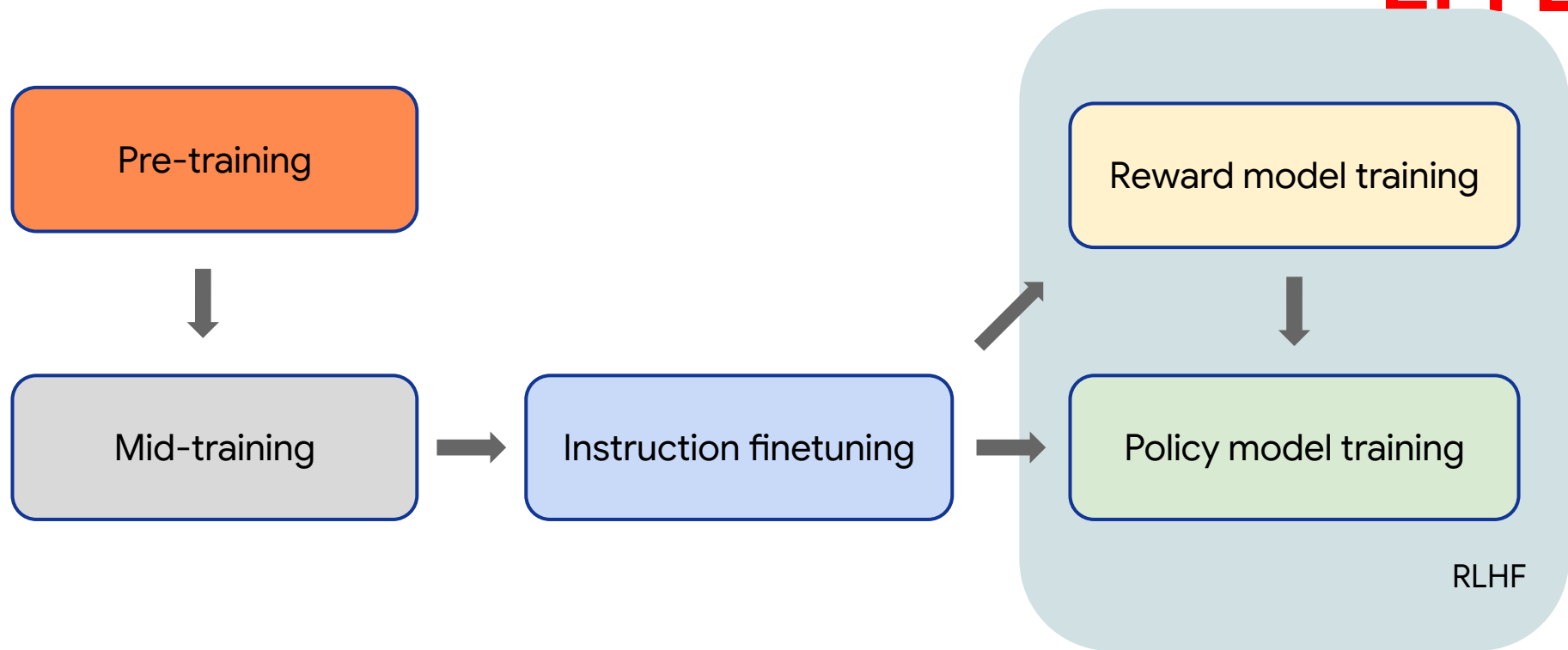
# Prompting techniques

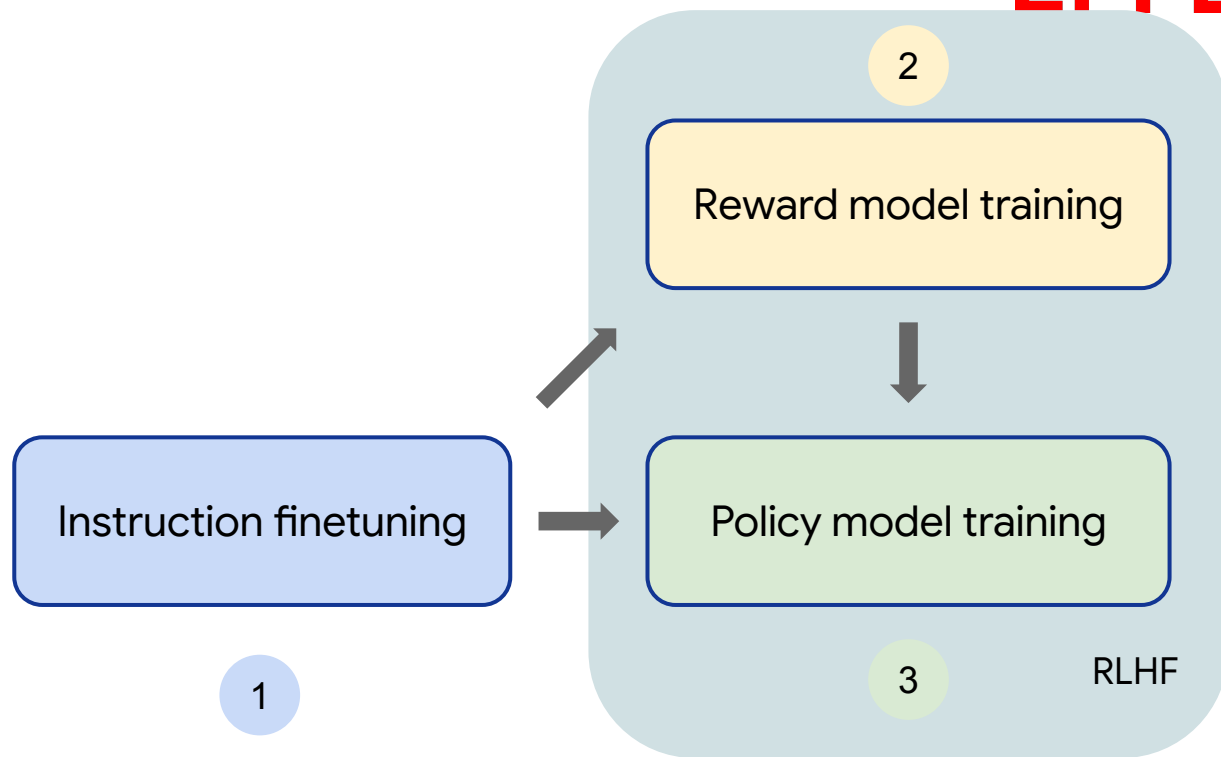


# Prompting vs Finetuning

	Prompting	Finetuning
Pros	<ul style="list-style-type: none"><li>• No data to get started</li><li>• Smaller upfront cost</li><li>• No technical knowledge needed</li><li>• Connect data through retrieval (RAG)</li></ul>	<ul style="list-style-type: none"><li>• Nearly unlimited data fits</li><li>• Learn new information</li><li>• Correct incorrect information</li><li>• Less cost afterwards if smaller model</li><li>• Use RAG too</li></ul>
Cons	<ul style="list-style-type: none"><li>• Much less data fits</li><li>• Forgets data</li><li>• Hallucinations</li><li>• RAG misses, or gets incorrect data</li></ul>	<ul style="list-style-type: none"><li>• More high-quality data</li><li>• Upfront compute cost</li><li>• Needs some technical knowledge, esp. data</li></ul>
	Generic, side projects, prototypes	Domain-specific, enterprise, production usage, ...privacy!

# Training Cycle of an LLM



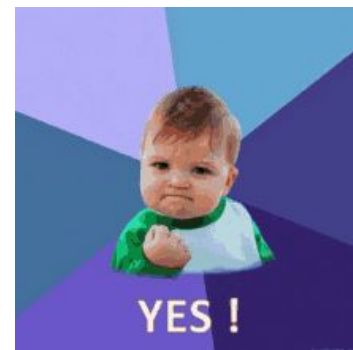




# Instruction Tuning LLMs

**Question:** Can we tune the models so that they can follow the instructions?

**Question:** Can we tune the models so that they can follow the instructions?



**Hack:** we can frame the question so that the answer is the next token

**Q:** The square root of  $x$  is the cube root of  $y$ . What is  $y$  to the power of 2, if  $x = 4$ ?

**A:**

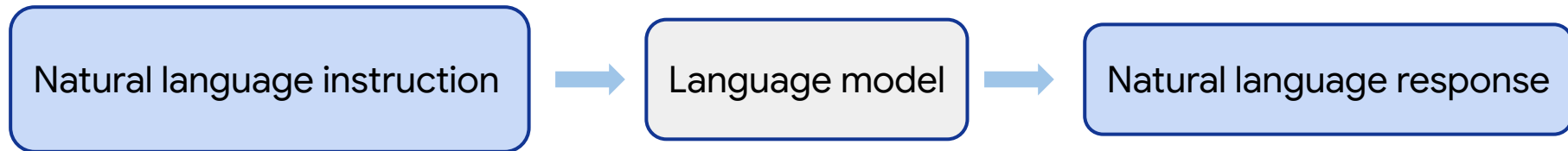


Pretrained model just predicts the next token, which happens to be the answer

Pre-trained models always generate something that is a natural continuation of the prompts *even if the prompts are malicious*

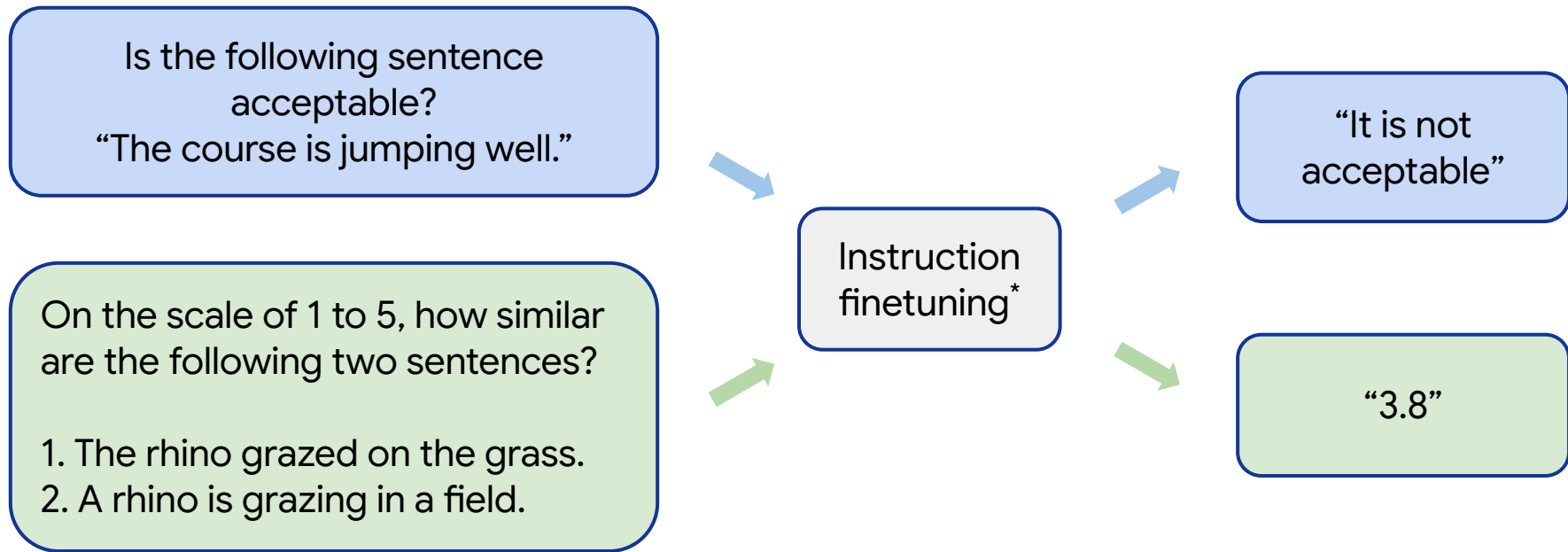
Frame **all** tasks in the form of

natural language instruction to natural language response mapping



**Input:** text

**Output:** text **EPFL**



Tasks are unified. So for an unseen task, the model just needs to respond to the natural language instruction

*Instruction fine-tuning is highly effective but it has inherent limitations*



# What is the learning objective in instruction finetuning?

For a given input, the target is the single correct answer.

- In RL, this is called “behavior cloning”

# What is the learning objective in instruction finetuning?

For a given input, the target is the single correct answer.

- In RL, this is called “behavior cloning”
- If we have enough data, the hope is that the model will be able to generalize.

# What is the learning objective in instruction finetuning?

For a given input, the target is the single correct answer.

- In RL, this is called “behavior cloning”
- If we have enough data, the hope is that the model will be able to generalize.
- This requires formalizing the correct behavior for a given input

## Exercise: think about the single correct answer

Input

$2 + 3?$

Target

5

## Exercise: think about the single correct answer

### Input

Translate this to Korean:  
“I should have studied instead of watching this movie”

### Target

나는 이 영화 보는 대신 공부를 했어야 했다

## Exercise: think about the single correct answer

### Input

Write a letter to a 5-year-old boy from Santa Clause explaining that Santa is not real. Convey gently so as not to break his heart

### Target



## Exercise: think about the single correct answer

### Input

Implement logistic regression with gradient descent in Python

### Target

```
class LogisticRegression:  
    ...
```

# What is Imitation Learning?

## An example: Behavior cloning

- Learning to mimic another expert agent.
- Typically pure supervised-learning algorithms.
- The student can't do better than teacher.



Alvinn: Pomerleau et al., (1989)



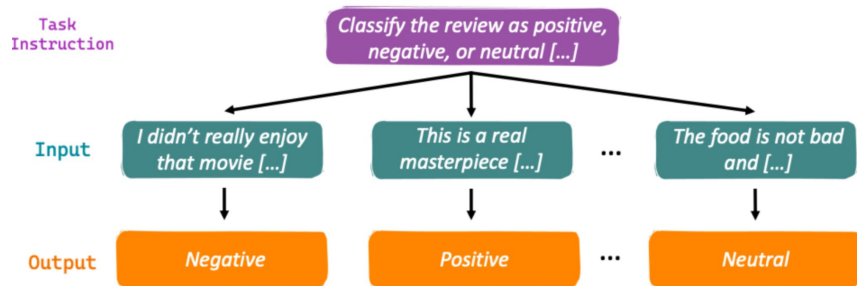
# Reasons for using imitation learning

- Reward functions are often not easy to come up with.
- Reward can be sparse and the task can be hard-exploration.



*Instruction finetuning is like behavior cloning!*

# Instruction tuning dataset designs



## a) Scaling inputs

For each task instruction, scaling up various input-output pairs.

- Conventional multi-task learning paradigm.



## b) Scaling input-free tasks

Scaling up instruction-output pairs directly. (e.g., Self-Instruct and Alpaca).

- Input contexts are omitted / tightly combined with instructions.

## SFT

- The model is trained on input-output pairs of data without any specific formatting on its output or inputs in a supervised manner.
- **Goal:** improve the model's performance on a **specific** task.
- **Example:** Finetune llama-3 on competitive-coding programming challenges.

## IFT

- More specific than SFT. The model is being provided an instruction in the prompt that it is supposed to follow on a diverse set of tasks.
- **Goal:** Teaching model to follow instructions on a diverse set of tasks and generalize to unseen instructions.
- **Example:** Fine-tuning on a dataset where each example consists of an instruction (e.g., "Provide a fix for the bug in this code") and the correct output.

# Insights to the finetuning datasets

A typical format of an instruction dataset is as follows:

Below is an instruction that describes a task, paired with an input that provides further context. Write a response that appropriately completes the request.



### Instruction:

{instruction}

### Input:

{input}

### Response:

These messages are typically formatted in JSON:

```
{
  "messages": [
    {"role": "system", "content": "You are a helpful and friendly AI assistant."},
    {"role": "user", "content": "What is the capital of France?"},
    {"role": "assistant", "content": "The capital of France is Paris."}
  ]
}
```

The roles typically are:

- **System:** Provides instructions on the assistant's behavior and response style.
- **User:** Represents the user's input or query.
- **Assistant:** Represents the chatbot's response given the instruction.

# SFT Datasets

# Self-instruct

[Wang et. al., 2023](#) proposed a dataset generated with bootstrapping method to automatically create instruction-response pairs for training instruction-tuned language models without extensive human annotation.

## Step 1: Form the seed set

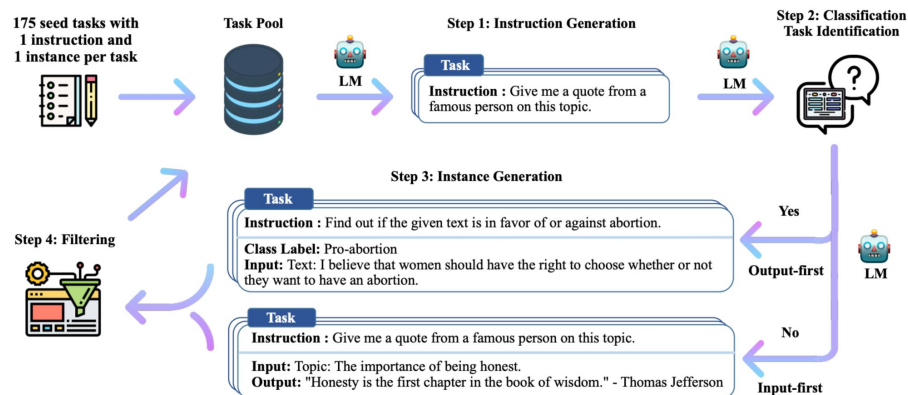
Create a seed set with a small set of manually created human-written instructions and responses.

## Step 2: Instruction Generation (Self-Instruct)

Existing large language model (such as GPT-3) is prompted with examples from the seed set to generate new instructions.

### *Example prompt:*

Here are some tasks and their responses:  
Task: Explain what the greenhouse effect is.  
Response: The greenhouse effect is...  
Task: Give three examples of mammals.  
Response: Examples include...  
Now generate another new task:



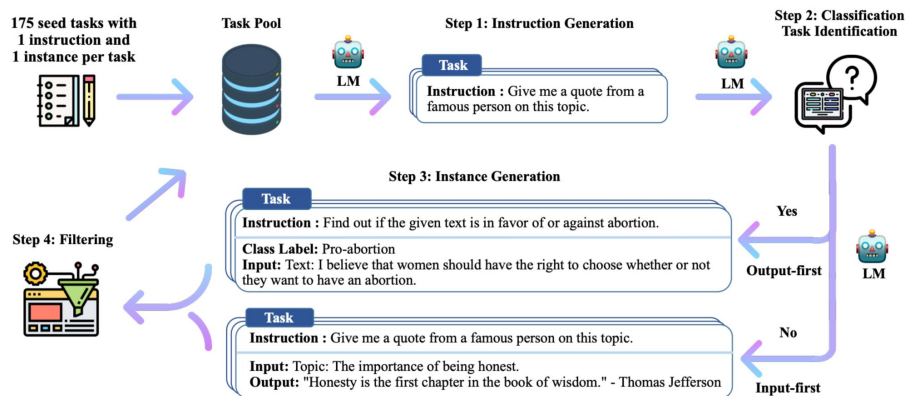
# Self-instruct

## Step 3: Filtering and Selection

- Automatically filter and prune instructions that are too similar, low quality, ambiguous, or repetitive, using heuristic or model-based criteria.
- The filtering ensures instruction diversity and quality.

## Step 4: Response Generation

- For each new instruction retained after filtering, the same language model (or another capable model) is prompted to generate a suitable response.



### Example prompt:

Task: Write a short summary of the movie "Inception".

Response:

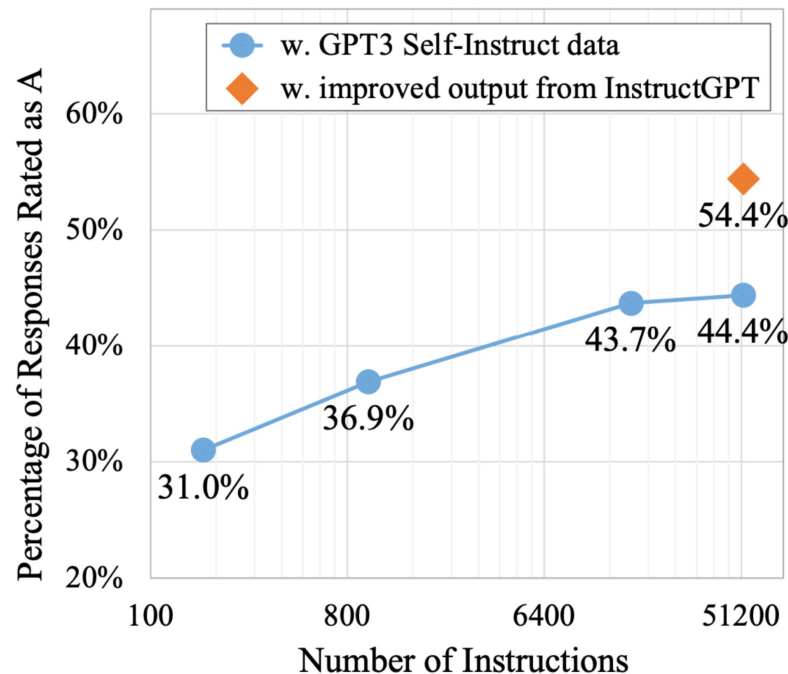


## Step 5: Bootstrapping

- Newly generated instruction-response pairs are added to the existing set, forming an expanded dataset.
- Repeat steps 2–4, progressively enlarging the dataset while ensuring quality through automated or human-in-the-loop checks.

### Benefits:

- Cost-effective: Dramatically reduces annotation costs compared to purely human-curated datasets.
- Scalable: Easily generates large and diverse instruction-response pairs.
- Adaptable: Allows rapid creation of datasets tailored to specific domains or tasks.



## Finetuning LLMs for longer length - The problem

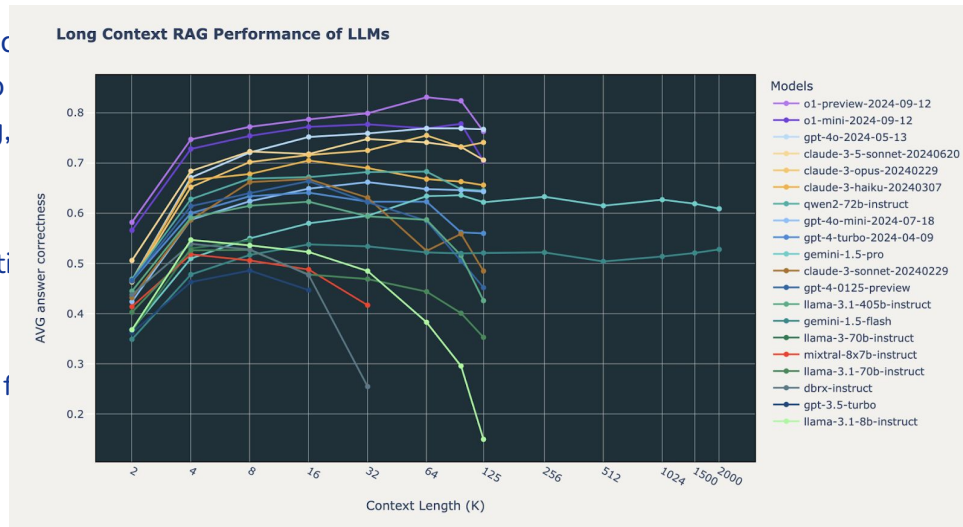
## Why?

Typically training LLMs on long-sequences is very hard  
3.2 is trained on sequence of length 8k and finetuned to  
*Several applications possible*: RAG, doc-level reasoning,

## The problem

*Lost in the middle problem:* Models struggle with information far from the beginning of the context.

*Repetition task:* Asking the model to output the last k words from the input. The task becomes longer.



[Leng et al, 2024, Long Context RAG Performance of LLMs](#)

# Finetuning on longer sequences - The solution

Finetune the model on longer sequences using datasets like:


- Books
- Legal Proceedings
- Government Reports


Often curriculum-like finetuning is done for example, for [llama3](#) models initial training is done over sequences of length 8k and model is gradually finetuned on sequences of length 131k.

[Gemma 3 report](#) pre-trained models over sequences up to length 32k and then finetuned them over sequences of length 128k. They reported that 8x increase in the sequence length is the sweet spot between efficiency and performance.

Useful resources:

- [Together.ai blogpost on long context finetuning.](#)
- [Notebook on example for Long context Fine-tuning for summarization](#)
- [Notebook on Long-context Fine-tuning for Repetition Task](#)

 Open in Colab

 Open in Colab



52k instructions and demos generated by OpenAI's `text-davinci-003` engine.

Built using the Self-Instruct recipe with the following modifications:

- The `text-davinci-003` engine to generate the instruction data instead of `davinci`.
- A new prompt was written that explicitly gave the requirement of instruction generation to `text-davinci-003`.
- Much more aggressive batch decoding was used, i.e., generating 20 instructions at once, which significantly reduced the cost of data generation.
- The data generation pipeline was simplified by discarding the difference between classification and non-classification instructions.
- Only a single instance was generated for each instruction, instead of 2 to 3 instances as in Self-Instruct.

Stanford  
Alpaca



# Alpaca - An Example

```
{  
  "instruction": "Create a classification task by clustering the given list of items.",  
  "input": "Apples, oranges, bananas, strawberries, pineapples",  
  "output": "Class 1: Apples, Oranges\nClass 2: Bananas, Strawberries\nClass 3: Pineapples",  
  "text": "Below is an instruction that describes a task, paired with an input that provides further context."  
}
```

- *instruction*: describes the task the model should perform. Each of the 52K instructions is unique.
- *input*: optional context or input for the task. For example, when the instruction is "Summarize the following article", the input is the article. Around 40% of the examples have an input.
- *output*: the answer to the instruction as generated by text-davinci-003.
- *text*: the instruction, input and output formatted with the [prompt template](#) used by the authors for fine-tuning their models.

One of the largest publicly available instructional datasets

**Diverse Tasks:**

- NLP tasks like sentiment analysis, text classification, etc.

Focuses on **Multi-task learning** and instruction-response pairs.

Mostly curated from existing resources: such as OpenAI GPT-3, GPT-4, etc.  
reformatted into instruction-response pairs.

Contains CoT, few-shot and zero-shot tasks

Release	Collection	Model	Model Details			Data Collection & Training Details			
			Base	Size	Public?	Prompt Types	Tasks in Flan	# Exs	Methods
2020 05	UnifiedQA	UnifiedQA	RoBerta	110-340M	P	ZS	46 / 46	750k	
2021 04	CrossFit	BART-CrossFit	BART	140M	NP	FS	115 / 159	71.M	
2021 04	Natural Inst v1.0	Gen. BART	BART	140M	NP	ZS / FS	61 / 61	620k	+ Detailed k-shot Prompts
2021 09	Flan 2021	Flan-LaMDA	LaMDA	137B	NP	ZS / FS	62 / 62	4.4M	+ Template Variety
2021 10	P3	T0, T0+, T0++	T5-LM	3-11B	P	ZS	62 / 62	12M	+ Template Variety + Input Inversion
2021 10	MetalCL	MetalCL	GPT-2	770M	P	FS	100 / 142	3.5M	+ Input Inversion + Noisy Channel Opt
2021 11	ExMix	ExT5	T5	220M-11B	NP	ZS	72 / 107	500k	+ With Pretraining
2022 04	Super-Natural Inst.	Tk-Instruct	T5-LM, mT5	11-13B	P	ZS / FS	1556 / 1613	5M	+ Detailed k-shot Prompts + Multilingual
2022 10	GLM	GLM-130B	GLM	130B	P	FS	65 / 77	12M	+ With Pretraining + Bilingual (en, zh-cn)
2022 11	xP3	BLOOMz, mT0	BLOOM, mT5	13-176B	P	ZS	53 / 71	81M	+ Massively Multilingual
2022 12	Unnatural Inst.†	T5-LM-Unnat. Inst.	T5-LM	11B	NP	ZS	~20 / 117	64k	+ Synthetic Data
2022 12	Self-Instruct†	GPT-3 Self Inst.	GPT-3	175B	NP	ZS	Unknown	82k	+ Synthetic Data + Knowledge Distillation
2022 12	OPT-IML Bench†	OPT-IML	OPT	30-175B	P	ZS + FS CoT	~2067 / 2207	18M	+ Template Variety + Input Inversion + Multilingual
2022 10	Flan 2022 (ours)	Flan-T5, Flan-PaLM	T5-LM, PaLM	10M-540B	P NP	ZS + FS CoT	1836	15M	+ Template Variety + Input Inversion + Multilingual

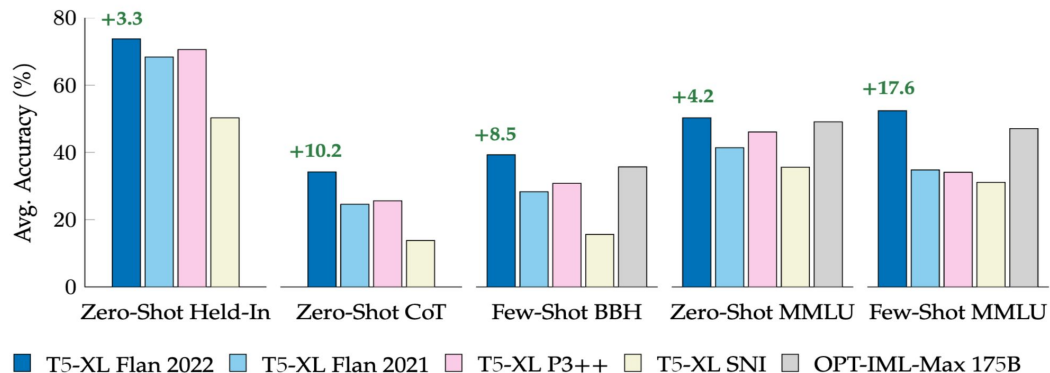
Two Variants:

- **FLAN (Original):** Initial dataset introduced with the FLAN models.
- **FLAN v2:** A significantly expanded version with more tasks, greater diversity, and improved instruction coverage, widely used for fine-tuning models such as FLAN-T5, FLAN-UL2, and FLAN-PaLM.

Works really well!

Papers:

1. Chung et al, 2022, ["Scaling Instruction-Finetuned Language Models"](#).
2. Longpre et al, 2023, ["The Flan Collection: Designing Data and Methods for Effective Instruction Tuning"](#).



An instruction tuning dataset aimed at enabling to develop chat-based assistant that **understands tasks**, can interact with **third-party systems**, and **retrieve information** dynamically to do so.

Project developed by LAION and available on huggingface as [oasst2](#).

The data quality is somewhat questionable. Contains approximately 129k training messages.

```
{
  "message_id": "218440fd-5317-4355-91dc-d001416df62b",
  "parent_id": "13592dfb-a6f9-4748-a92c-32b34e239bb4",
  "user_id": "8e95461f-5e94-4d8b-a2fb-d4717ce973e4",
  "text": "It was the winter of 2035, and artificial intelligence (..)",
  "role": "assistant",
  "lang": "en",
  "review_count": 3,
  "review_result": true,
  "deleted": false,
  "rank": 0,
  "synthetic": true,
  "model_name": "oasst-sft-0_3000,max_new_tokens=400 (..)",
  "labels": {
    "spam": { "value": 0.0, "count": 3 },
    "lang_mismatch": { "value": 0.0, "count": 3 },
    "pii": { "value": 0.0, "count": 3 },
    "not_appropriate": { "value": 0.0, "count": 3 },
    "hate_speech": { "value": 0.0, "count": 3 },
    "sexual_content": { "value": 0.0, "count": 3 },
    "quality": { "value": 0.416, "count": 3 },
    "toxicity": { "value": 0.16, "count": 3 },
    "humor": { "value": 0.0, "count": 3 },
    "creativity": { "value": 0.33, "count": 3 },
    "violence": { "value": 0.16, "count": 3 }
  }
}
```



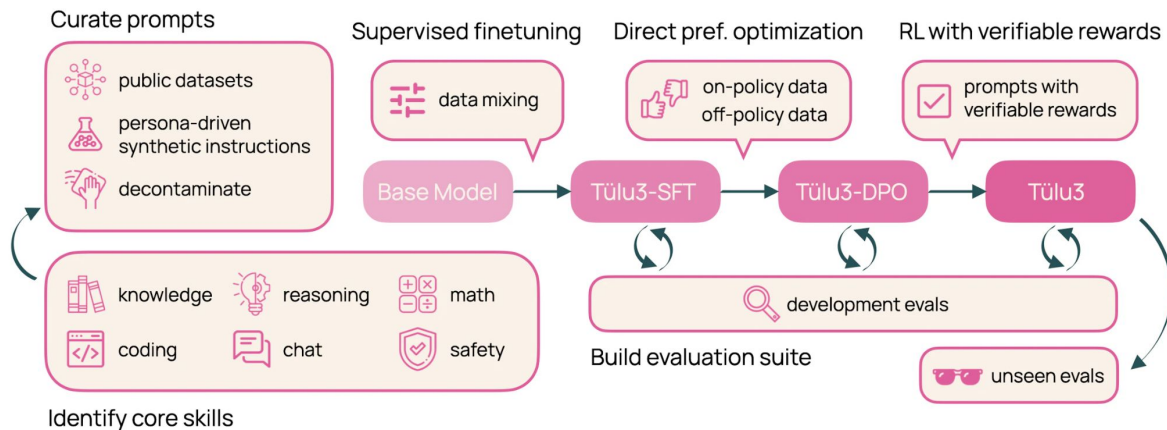
# Finetuning recipes

# Tulu 3 overview



# TULU 3 Steps

- (1) careful prompt curation and synthesis targeting core skills,
- (2) supervised finetuning (SFT) on our carefully selected mix of prompts and their completions,
- (3) Direct Preference Optimization (DPO) on both off- and on-policy preference data
- (4) a new RL-based method to enhance specific skills with verifiable rewards, and
- (5) a standardized evaluation suite for development, decontamination, and final evaluation stage.



# Finetuning for tool use

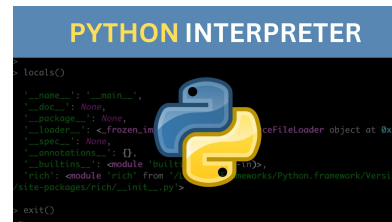
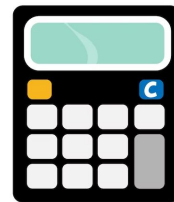
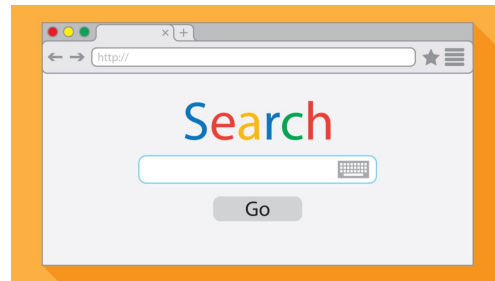
# Toolformer

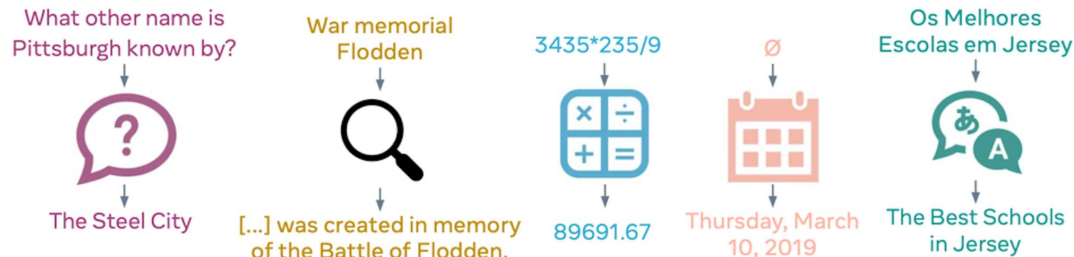
The New England Journal of Medicine is a registered trademark of [QA("Who is the publisher of The New England Journal of Medicine?") → Massachusetts Medical Society] the MMS.

Out of 1400 participants, 400 (or [Calculator(400 / 1400) → 0.29] 29%) passed the test.

The name derives from "la tortuga", the Spanish word for [MT("tortuga") → turtle] turtle.

The Brown Act is California's law [WikiSearch("Brown Act") → The Ralph M. Brown Act is an act of the California State Legislature that guarantees the public's right to attend and participate in meetings of local legislative bodies.] that requires legislative bodies, like city councils, to hold their meetings open to the public.





Who is the current President of the United States?



I don't have real-time data or internet access.

What day of the week is it today?



I do not have the capability to access the current time or date.

What is the result of  $3435 * 235 / 9$ ?

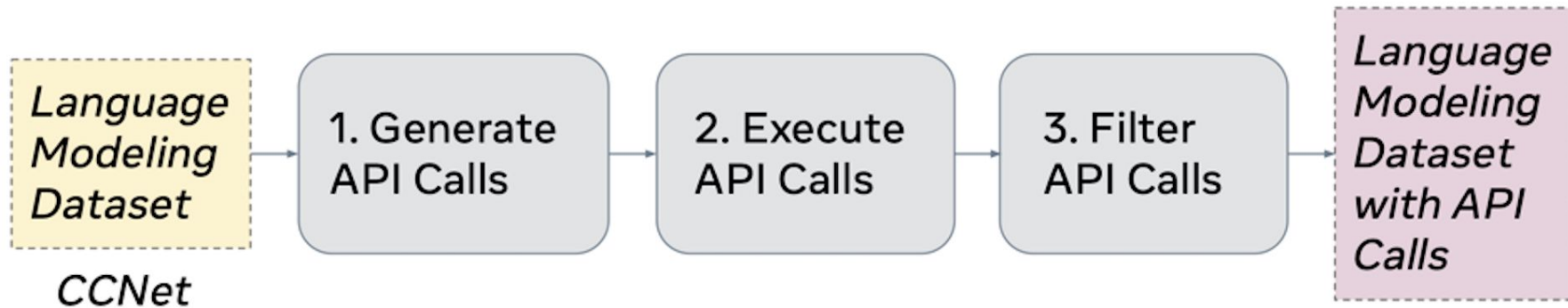


This is approximately 89,937.22.

True answer: 89691.66

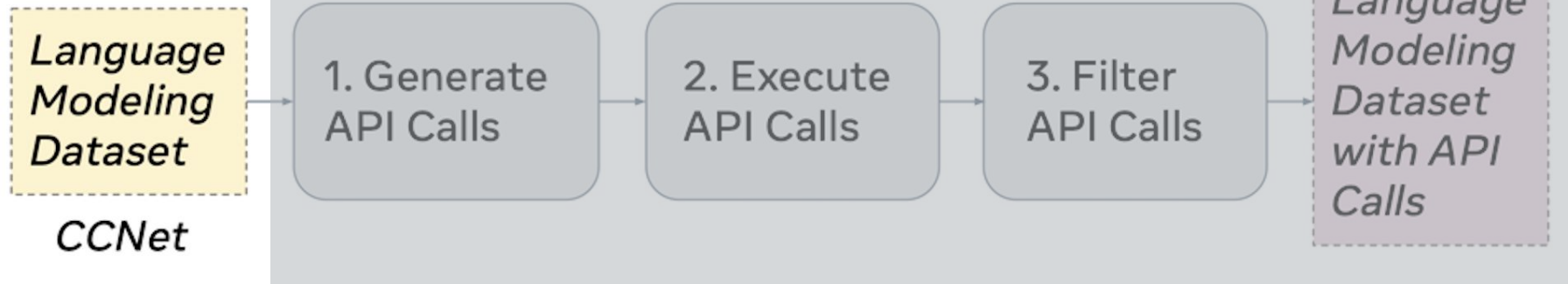
# Toolformer overview

Very fast step is to train an LLM on a large amounts of datasets



# Toolformer overview

Very first step is to train an LLM on a large amounts of datasets.





# Toolformer overview

Language  
Modeling  
Dataset

CCNet

1. Generate  
API Calls

2. Execute  
API Calls

3. Filter  
API Calls

Language  
Modeling  
Dataset  
with API  
Calls

$\epsilon$  : empty string

$c = (a_c, i_c)$  : API call which is a tuple of,

$a_c$  : the name of the API,

$i_c$  : input corresponding to the API call,

$\mathcal{C} = \{x^1, \dots, x^{|\mathcal{C}|}\}$  : A dataset of plaintext,

$P(x)$  : prompt,

$p_M(z_{n+1}|z_{1:n})$  : the probability assigned to the next token in the sequence by the model  $M$ ,

$I = \{i|p_i > \tau_s\}$ ,

$e(c) = \langle \text{API} \rangle a_c(i_c) \langle / \text{API} \rangle$ : API call on  $c$ ,

$r$  : the result of the API query,

$e(c) = \langle \text{API} \rangle a_c(i_c) \rightarrow r \langle / \text{API} \rangle$ : API call with the result,

$L_i(\epsilon)$  : The loss obtained by not doing an API call,

$L_i(e(c_i, \epsilon))$  : the loss incurred by making an API call without providing a response

# Toolformer - Sampling API Calls

For each token at every position  $i$  in an input sequence compute the follow

$$p_i = p_M(< API > | P(x), x_{1:i-1}),$$

that model  $M$  assigns to starting an API call at position  $i$ . We define a probability threshold  $\tau_s$  where we keep all the positions above the threshold:

For each position  $i \in I$ , we then obtain up to  $M$  API calls  $I = \{i | p_i > \tau_s\}$  by sampling from  $M$  given the sequence  $c_i^1, \dots, c_i^m$  as a prefix and  $</API>$  as an end of API call token.  $[P(x), x_1, \dots, x_{i-1}, < API >]$

Your task is to add calls to a Question Answering API to a piece of text. The questions should help you get information required to complete the text. You can call the API by writing "[QA(question)]" where "question" is the question you want to ask. Here are some examples of API calls:

**Input:** Joe Biden was born in Scranton, Pennsylvania.

**Output:** Joe Biden was born in [QA("Where was Joe Biden born?")] Scranton, [QA("In which state is Scranton?")] Pennsylvania.

**Input:** Coca-Cola, or Coke, is a carbonated soft drink manufactured by the Coca-Cola Company.

**Output:** Coca-Cola, or [QA("What other name is Coca-Cola known by?")] Coke, is a carbonated soft drink manufactured by [QA("Who manufactures Coca-Cola?")] the Coca-Cola Company.

**Input: x**

**Output:**

# Toolformer overview

Language  
Modeling  
Dataset

CCNet

1. Generate  
API Calls

2. Execute  
API Calls

3. Filter  
API Calls

Language  
Modeling  
Dataset  
with API  
Calls

What other name is Pittsburgh known by?



The Steel city

Pittsburgh is known as [QA("What other name is Pittsburgh known by?") →  
the Steel City] the Steel City.

# Toolformer overview

*Language  
Modeling  
Dataset*

CCNet

1. Generate  
API Calls

2. Execute  
API Calls

3. Filter  
API Calls

*Language  
Modeling  
Dataset  
with API  
Calls*

# Toolformer Filtering overview

Given,

$$\begin{aligned} L_i^+ &= L_i(e(c_i, r_i)), \\ L_i^- &= \min(L_i(\epsilon), L_i(e(c_i, \epsilon))). \end{aligned}$$

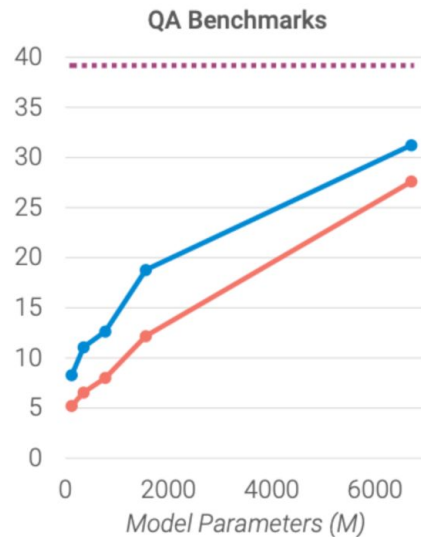
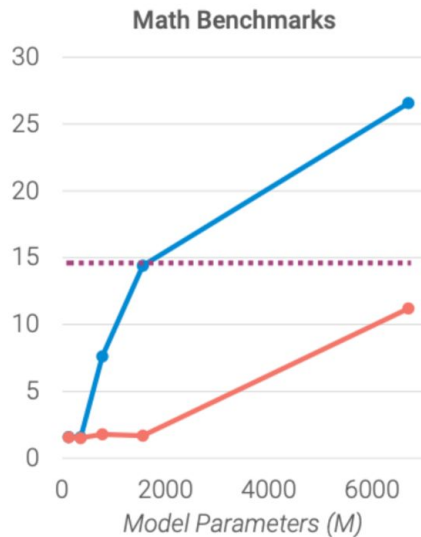
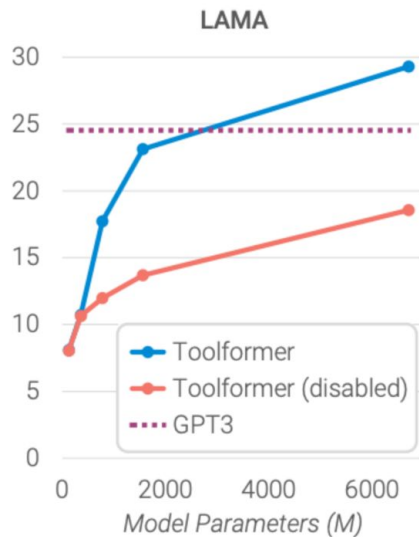
the filtering can be done by thresholding and looking at the difference in loss between the loss incurred after inserting API response and the loss incurred without API response:

$$L_i^- - L_i^+ \geq \tau_f$$

## Last step: Model finetuning

Once the filtering is done, we finetune the model on a dataset of texts paired with API calls and the responses to those calls.

# Toolformer Results



# Parameter Efficient Tuning

# What is PEFT?

Parameter efficient fine-tuning typically corresponds to a family of finetuning approaches that freezes the parameters of the original pretrained network and only trains small set of parameters introduced for the finetuning.

These techniques are in particular helpful with very large scale models such as LLMs where the full-scale finetuning of the model prohibitively can be expensive in terms of computational resources and time.

Some examples we will discuss are:

1. LoRA
2. DoRA
3. QLoRA
4. Adapter Layers



Given a set of pre-trained weight matrix  $W$  and the weights after the pre-training the  $W'$  with the finetuning  $\Delta W$  updates. We could write the updates as:

$$W' = W + \Delta W,$$

**Hypothesis:** Finetuning updates to the weight matrices ( $\Delta W$ ) has an *implicit low-rank*.

Instead of learning a full-rank update  $\Delta W$ , LoRA parameterizes the update using two *low-rank matrices*, reducing the number of trainable parameters.

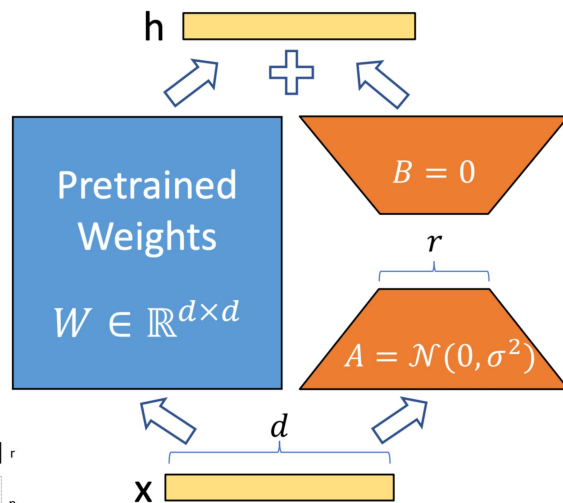
LoRA assumes that weight updates can be effectively **approximated** using a low-rank decomposition:  $\Delta W = AB$  where,

- $A \in \mathbb{R}^{d_{out} \times r}$  (rank  $r$ ),
- $B \in \mathbb{R}^{r \times d_{in}}$  (rank  $r$ ),
- $r \ll \min(d_{in}, d_{out})$  (small rank).

LoRA

$$W = \underbrace{W_0}_{\text{frozen}} + \underbrace{A}_{\text{trainable}} \underbrace{B}_{\text{trainable}}$$

Diagram illustrating the LoRA decomposition: A large orange matrix  $W_0$  of size  $n \times n$  is added to the product of a green matrix  $A$  of size  $n \times r$  and a green matrix  $B$  of size  $r \times n$ . A note indicates "low rank  $r \ll n$ ".



# Advantages of LoRA finetuning

1. **Reduced Memory Usage:** LoRA requires significantly fewer trainable parameters, making it ideal for fine-tuning **large models** on resource-constrained devices.
2. **No Need to Store Full Updates:** Since the base model remains frozen, **LoRA avoids maintaining multiple copies** of large model weights.
3. **Modular & Efficient:** LoRA updates can be applied selectively to specific layers (e.g., attention layers), reducing computational overhead.

[Liu et al. 2024](#) proposed DoRA to improve the performance and stability of LoRA by decoupling the **direction** and the **magnitude** of the weight updates.

## Hypothesis:

- Learning directional updates is an easier problem.
- Ignoring the magnitude of the weights simplifies the optimization procedure.

### 1. Weight decomposition

Instead of directly finetuning weights of  $W_0$ , DoRA first decomposes the weights and then the magnitude component:

$$W = m \frac{V}{\|V\|_c},$$

- $\|V\|_c$  is the column-wise norm.
- $m \in \mathbb{R}^{1 \times k}$  is the magnitude vector (a learnable vector)
- $V \in \mathbb{R}^{d \times k}$ , directional matrix.

LoRA updates are applied only to the directional matrix  $V$ , while magnitude vector is trained independently.

To efficiently tune  $V$ , we can use a learnable low-rank decomposition:  $\Delta V = AB$

- $A \in \mathbb{R}^{d \times r}$  and  $B \in \mathbb{R}^{r \times k}$  are low-rank matrices.
- $r \ll \min(d, k)$  meaning only small subset of params are trained.

The final fine-tuned weight formulation would be:  $W' = m \frac{V + \Delta V}{\|V + \Delta V\|_c^2} = m \frac{V + AB}{\|V + AB\|_c^2}$ ,

## Efficiency

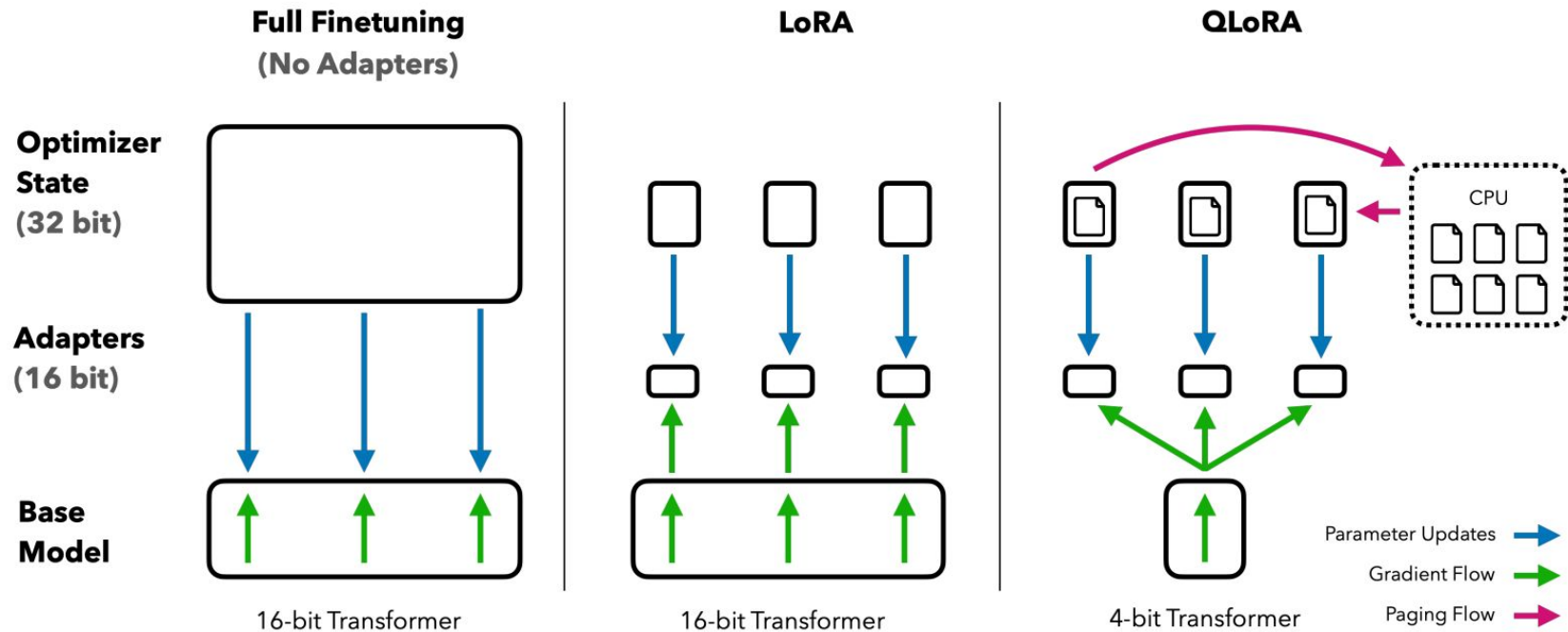
- Standard full-finetuning updates  $d \times k$  parameters.
- LoRA updates  $r(d+k)$  parameters.
- DoRA updates  $r(d+k) + k$  parameters.

During training due to the normalization DoRA would be slower.

During inference all the operations can be merged, DoRA has zero additional latency when compared to LoRA.

## Takeaways

- Empirically performs better than LoRA.
- Training is more stable and more efficient than full finetuning.



Major innovations:

- 4-bit Normal Float
- Double Quantization
- Page Optimizer to use when the parameters of the model don't fit into the GPU memory.

Q-LoRA uses:

- 4 bit storage data-type.
- bfloat16 computational data type.

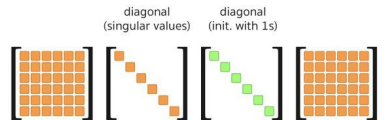
Main findings:

- It reduces the average memory requirements of finetuning a 65B LLM from 780GB of memory to 48GB memory which enables to finetune on a single GPU.
- Preserving the performance compared to a 16-bit fully finetuned baseline while keeping the pre-trained weights in 4-bit precision.

# SVF (Sun et al 2025)

**Idea:** Do SVD on the pre-trained weights and finetune the singular values:

SVF


$$W = U \quad S \quad \underline{Z} \quad V^T$$

*trainable  
(all the rest frozen)*

Note: illustrated using square matrices but this works for rectangular matrices too.

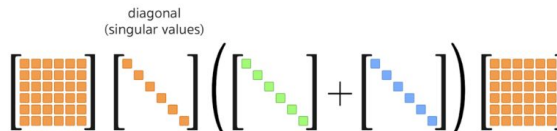
\* Images taken from:

<https://towardsdatascience.com/are-you-still-using-lora-to-fine-tune-your-llm/>

Paper link: <https://arxiv.org/pdf/2501.06252>

The weights would be composable too:

SVF


$$W = U \quad S \quad (\alpha \underline{Z}_1 + \beta \underline{Z}_2) \quad V^T$$

*composing models  
fine-tuned on two different  
tasks e.g. code and math*

# SVFT (Sun et al 2025)

Instead of diagonal matrix only integrate other sparse matrices:

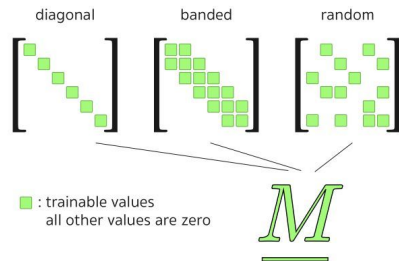
SVFT

diagonal (singular values)    options: diagonal, banded, random, top-k, ...

$$W = U (S + M) V^T$$

Note: illustrated using square matrices but this works for rectangular matrices too.

SVFT



\* Images taken from:

<https://towardsdatascience.com/are-you-still-using-lora-to-fine-tune-your-llm/>

Paper link: <https://arxiv.org/pdf/2501.06252>

Table 5: Results on fine-tuning Gemma-2B with SVFT using different  $M$  parameterizations.

Structure	#Params	GSM-8K	MATH
Plain <sup>(diagonal)</sup>	<u>0.2M</u>	40.34	14.38
Banded	<u>3.3M</u> 6.4M	46.47 47.84	<b>16.04</b> 15.68
Random	3.3M <u>6.4M</u>	47.76 <b>50.03</b>	<u>15.98</u> 15.56
Top- $k$	3.3M 6.4M	48.00 <u>49.65</u>	15.80 15.32



# PISSA (Meng et al, 2024)

PiSSA (Principal Singular values and Singular Vectors Adaptation) claims that you should only tune the large principal values.

**SVD (the Math)**

$$\begin{bmatrix} U_p & U_m \end{bmatrix} + \begin{bmatrix} S_p & \\ & S_m \end{bmatrix} + \begin{bmatrix} V_p^T \\ V_m^T \end{bmatrix}$$

diagonal (singular values)

$$W = U_p S_p V_p^T + U_m S_m V_m^T$$

large singular values      small singular values

**PiSSA**

trainable (low rank  $r$ )      frozen

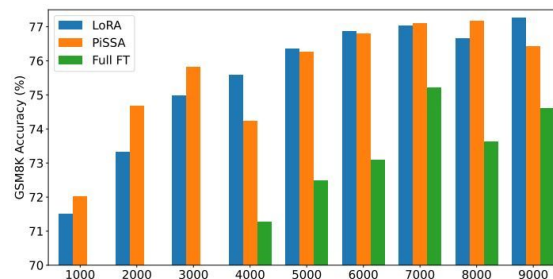
$$W = \underline{A} \underline{B} + U_m S_m V_m^T$$

init. with top  $r$  singular values and vectors

$$\begin{cases} A = U_p \sqrt{S_p} \\ B = \sqrt{S_p} V_p^T \end{cases}$$

## Interesting finding:

Full fine-tuning is prone to over-fitting. You might get better results in the absolute with a low-rank fine-tuning technique.



(c) Accuracy on GSM8K over training steps.

Figure 11: Fine-tuning Mistral-7B-v0.1 on the MetaMathQA-395K dataset for 3 epochs.

# MiLoRA (Wang et al, 2025)

MiLoRA (Minor singular component LoRA [arxiv.org/abs/2406.09044](https://arxiv.org/abs/2406.09044)), on the other hand, claims that you should only tune the small principal values. It uses a similar mechanism to PiSSA:

**MiLoRA**

$$W = \overbrace{U_p S_p V_p^T}^{\text{frozen}} + \overbrace{\underline{A} \underline{B}}^{\text{trainable (low rank } r \text{)}}$$
$$\begin{aligned} A &= U_m \sqrt{S_m} \\ B &= \sqrt{S_m} V_m^T \end{aligned}$$

init. with lower  $r$  singular values and vectors

MiLoRA performs better on math datasets which are probably fairly aligned with the original pre-training. Arguably, PiSSA should be better for altering the behavior of the LLM further from its original weights.

PEFT	GSM8K	MATH	Avg.
LoRA	56.6	10.8	33.7
PiSSA	51.3	10.4	30.8
MiLoRA	<b>58.6</b>	<b>11.6</b>	<b>35.1</b>

Table 9: Math reasoning evaluation results for LLaMA2- 7B

# LORA-XS (Balazy et al, 2024)

LoRA-XS is similar to PiSSA. It also shows good results with significantly fewer params than LoRA.

LoRA-XS

$$W = W_0 + U_p S_p \underline{R} V_p^T$$

init: very small random values ( $\sim 10^{-5}$ )

top-r singular values

trainable

The paper offers a mathematical explanation of why this setup is “ideal” under two conditions:

- truncating the bottom principal values from the SVD offers a good approximation of the weights
- the fine-tuning data distribution is close to the pre-training one

Model	Method	Rank	# Trainable Parameters	GSM8K	MATH
Mistral (7B)	Full FT	-	7242M	67.02	18.60
	LoRA	64	168M	67.70	19.68
	LoRA-XS	64	0.92M	68.01	17.86
		32	0.23M	63.23	15.88
		16	0.057M	57.92	14.44
Gemma (7B)	Full FT	-	8538M	71.34	22.74
	LoRA	64	200M	74.90	31.28
	LoRA-XS	64	0.80M	74.22	27.62
		32	0.20M	71.72	27.32
		16	0.050M	68.46	26.38

Table 3: Instruction tuning performance on GSM8K and MATH Benchmarks for Mistral-7B and Gemma-7B models using full fine-tuning, LoRA, and LoRA-XS. Full fine-tuning and LoRA performance values are taken from prior work (Meng et al., 2024). Higher is better for all metrics. LoRA-XS performs competitively or better than both LoRA and full fine-tuning while being significantly more parameter efficient in the studied settings.

# Related works: Prompt tuning (Lester et al. 2021)

Simple lightweight finetuning focusing on only finetuning the "soft-token" embeddings.

## Method

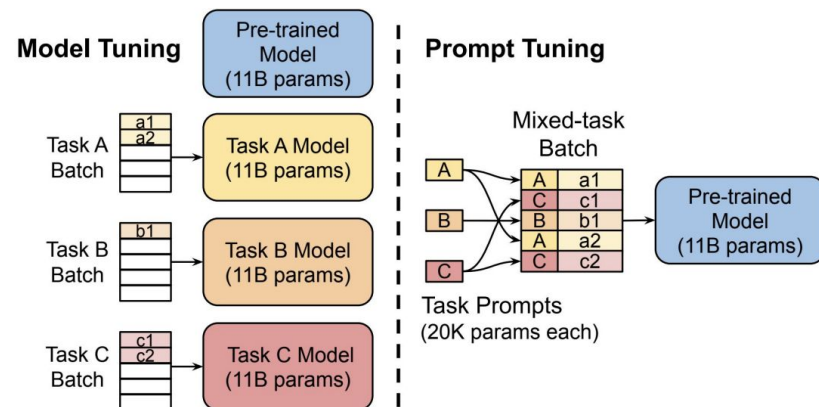
1. **Initialize:** Init the soft-token embeddings.
2. **Training:** Prepend the soft-token embeddings to the beginning of each training inputs and only finetune those embeddings.
3. **Inference:** Prepend the trained soft-token embeddings to the beginning of each input sequence.

## Pros

- Simple and efficient.
- Works well for simple task adaptation.

## Cons

- The performance is worse than LoRA.
- Worse generalization performance than LoRA.



# Related Works: Adapters (Houlsby et al. 2019)

Adapter layers are light-weight MLPs inserted into transformer blocks after attention and FFN layers.

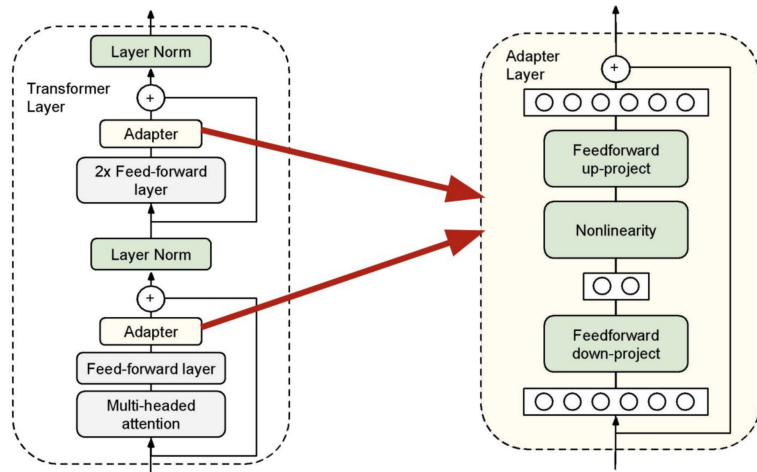
**Principle:** Insert a small MLP with a bottleneck into the transformer block and only finetune the params of those layers while rest of the params of pre-trained model are frozen.

## Architecture:

1. Linear down projection that projects to the lower-dim bottleneck using  $W_{down}$ .
2. Nonlinear activation function such as ReLU and GeLU.
3. Linear projection that restores the original dim with a linear projection using  $W_{up}$ .

Ignoring the biases, it can be parameterized as:

$$Adapter(x) = x + W_{up} Activation(W_{down}x)$$



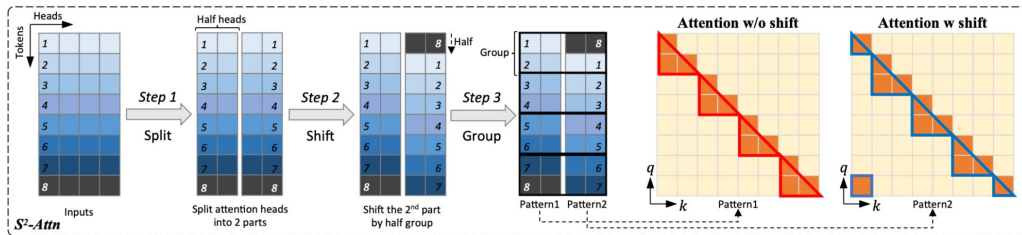
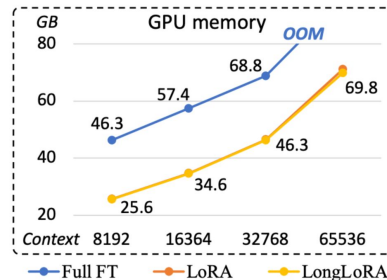
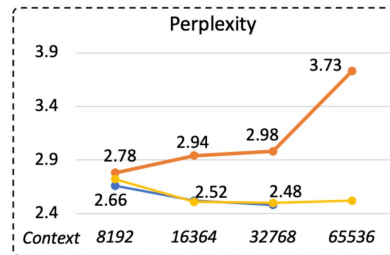
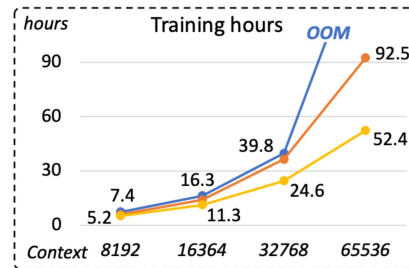
## Use cases:

1. Multimodal models like Flamingo.
2. Multitask learning where you learn a separate adapter layer for each task.

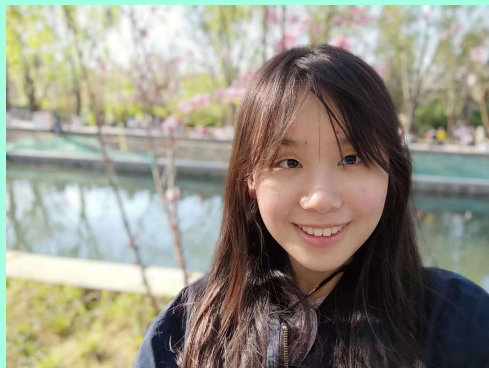
# LongLoRA: Finetuning LLMs for long context

[Chen et al. 2024](#), proposed:

1. **LORA+** in addition to the LoRA weights also train the embedding and normalization params:
2. **Shifted sparse attention** divides input tokens into several groups and computes attention separately within each group to reduce computational costs.
  - introduces a token-shifting mechanism across half of the attention heads, enabling information flow between neighboring groups, thereby approximating global attention effectively during fine-tuning without increasing computational overhead.



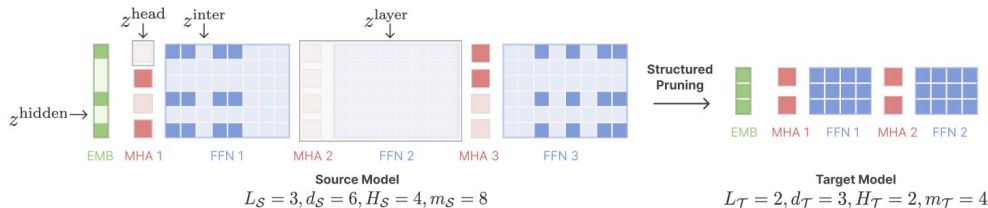
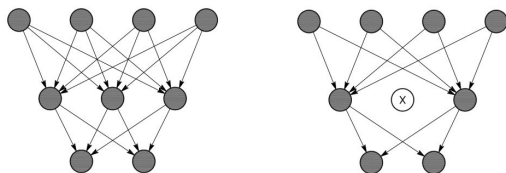
# Pruning and Quantization



# What is Network Pruning?

**Goal:** Pruning is to identify and remove weights or neurons that have little impact on the model's output, with the benefit of reduced storage and FLOPs.

## Where to prune



**1. unstructured pruning:** individual weight  
high performance but requires hardware optimization

**2. structured pruning:** channel, filter, attention head, layer, ...  
lower performance but more efficient

Different pruning criteria to determine which weights/ channels/ heads/layers/.. are less important:  
magnitude, Taylor, Hessian, Fisher, learnable, ...

## When to prune

pruning from scratch

Iterative pruning during training

**Pruning after pre-training without any retraining**

**Pruning after pre-training with efficient retraining**

....



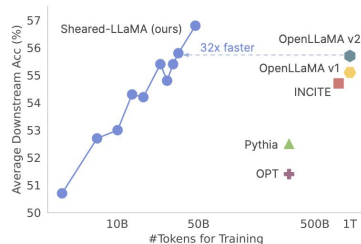
# Efficiency matters in pruning LLMs!

general-purpose pruned model with much less data and retraining compute

**ShearedLlama (Xia et al. 2023):** structured pruning with learnable pruning mask, then continue pre-training

Mid-training on the pruned model reduces loss at different rates across domains compared to training from scratch, which signifies an inefficient use of data.

Dynamic batch loading, which dynamically updates the composition of sampled data in each training batch based on varying losses across different domains

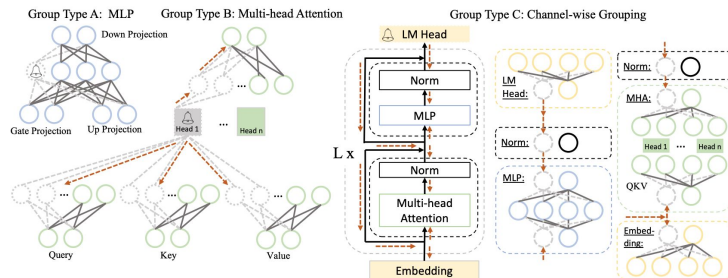


**LLM-Pruner (Ma et al. 2023):** structured pruning with group importance, then fine-tuning with LoRA

Identifying groups of interdependent structures and estimate the importance of each group via gradient and hessian information.

Use 50k publicly available samples to post-train the pruned model, which only needs 3 hours on a single GPU

Pruning the LLaMA2-7B model down to 1.3B parameters with 50B tokens and 3% extra training compute.



# Efficiency matters in pruning LLMs!

## general-purpose pruned model with real hardware efficiency gain

### Structured pruning

Many works focus on removing the whole layer rather than fine-grained weights removal.

- [1].Shortened LLaMA: A Simple Depth Pruning for Large Language Models
- [2].ShortGPT: Layers in Large Language Models are More Redundant Than You Expect
- [3].A deeper look at depth pruning of LLMs
- [4].The Unreasonable Ineffectiveness of the Deeper Layers
- [5].What Matters in Transformers? Not All Attention is Needed
- [6].Rethinking the Impact of Heterogeneous Sublayers in Transformers
- [7].Streamlining Redundant Layers to Compress Large Language Models
- [8].Reassessing Layer Pruning in LLMs: New Insights and Methods

....

Many submissions about layer pruning to ICLR'25!

Heads and dimension reduction is also promising in achieving good inference efficiency ([Xia et al. 2023](#), [Ma et al. 2023](#), [S Muralidhara et al. 2024](#), [S Guo et al. 2023](#),... ).

### Unstructured pruning

Fast cuda kernels are needed. Example: Flash-LLM ([H Xia et al. 2023](#))

It proposes a general Load-as-Sparse and Compute-as-Dense method, which addresses the significant memory bandwidth bottleneck, and achieves 3.8× and 3.6× improvement over DeepSpeed and FasterTransformer on OPT-30B/66B/175B models.

# Other new things in pruning LLMs

Besides neurons pruning, token pruning is also important in LLMs

Not all tokens are important. Removing tokens in computation can alleviate KVCache problem [M Adnan et al. 2024](#), [Z Zhang et al. 2023](#).

Combine with knowledge distillation and quantization

Scaling law study ([E Frantar et al. 2023](#))

It incorporates the sparsity ratio  $S$  into the dense scaling law formulation. This helps identify the optimal sparsity with lowest loss for a fixed number of non-zero parameters  $N$  and fixed training cost  $C$ .

Survey	Main work description	Year
[41]	survey of pruning methods	1993
[42]	overview 81 papers and propose ShrinkBench, an open-source framework for evaluation	2020
[43]	pruning+weight sharing+low-rank matrix+KD+quantization	2020
[44]	pruning criteria+procedure	2020
[45]	pruning+quantization+KD+low-rank factor.	2020
[32]	pruning+KD+NAS+Tensor-Decompose+quantization+hardware acceleration	2020
[46]	work on sparsity in deep learning up to 2020	2021
[36]	overview pruning at initialization	2022
[33]	pruning+KD+NAS+quantization+Tensor-Decompose+hardware accel.	2022
[37]	structured pruning	2023
[40]	pruning+KD+quantization+others for LLM	2023
[47]	pruning+KD+quantization+others for LLM	2024
[34]	pruning+KD+quantization+others for LLM	2024
[35]	model compression+others for LLMs	2024

A survey paper: [H Cheng et al. 2024](#)

$$L(S, N, D) = \left( a_S(1 - S)^{b_S} + c_S \right) \cdot \left( \frac{1}{N} \right)^{b_N} + \left( \frac{a_D}{D} \right)^{b_D} + c.$$

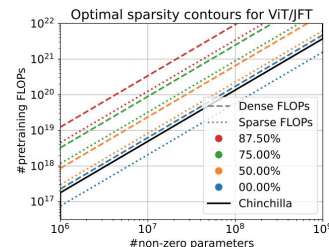
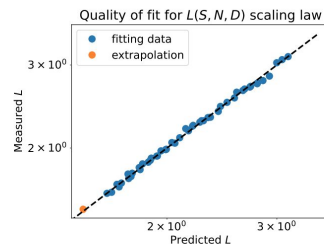


Figure 1: (Left) Fit and extrapolation quality of the  $L(S, N, D)$  scaling law on T5/C4. (Right) Optimal sparsity  $S_{\text{opt}}$  contours fitted on ViT/JFT, for sparse and dense costs (details in Section 3.3).

# What is network quantization?

**Principle:** Quantization is to map floating-point values to fixed-point integers, and then enable fast matrix multiplication by calculating integer multiplication first.

$$x_{\text{int}} = \text{Clamp} \left( \left\lfloor \frac{x}{s} \right\rfloor + z, 0, 2^b - 1 \right),$$

$$x_{\text{quant}} = s(x_{\text{int}} - z)$$

$$y = \text{MatMul}(x, w) \approx \text{MatMul}(x_{\text{quant}}, w_{\text{quant}})$$

$$= \text{MatMul}(s_x(x_{\text{int}} - z_x), s_w(w_{\text{int}} - z_w))$$

$$= s_x s_w \text{MatMul}(x_{\text{int}}, w_{\text{int}}) + C$$

Quantization scheme: scale factor  $s$ , zero point  $z$ , bit-width  $b$



apply to both weights and activations

Integer matrix multiplication,  $C$  includes other terms

## Methods:

**I. Quantization-aware training with large-scale data and compute:** fully train weights to make it adapt to quantization

**II. Post-training quantization with limited unlabeled data and compute:** 1) optimizing the quantization parameters.  
2) slightly tune weights

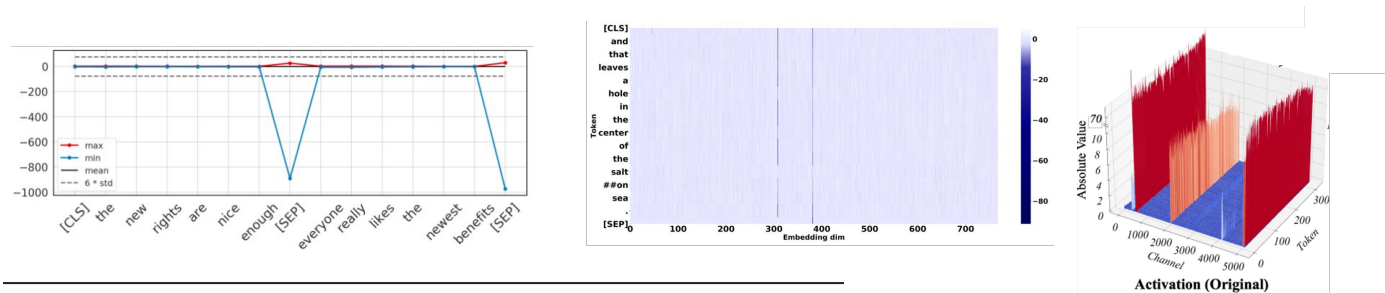
# Outliers in LLM quantization

$$x_{\text{int}} = \text{Clamp} \left( \left\lfloor \frac{x}{s} \right\rfloor + z, 0, 2^b - 1 \right),$$

trade-off between clipping error and rounding error  
with long-tail tensor distribution, there can be large quantization error!

$$x_{\text{quant}} = s(x_{\text{int}} - z)$$

Unfortunately, there are obvious outliers with certain patterns in activations of LLM. This worsens even INT8 (8-bit quantization) post-training quantization performance ([Y Bondarenko et al. 2021](#), [X Wei et al. 2022](#), [G Xiao et al. 2022](#)). Some examples:



Configuration	CoLA	SST-2	MRPC	STS-B	QQP	MNLI	QNLI	RTE	GLUE
FP32	57.27	93.12	88.36	89.09	89.72	84.91	91.58	70.40	<b>83.06</b>
W8A8	54.74	92.55	88.53	81.02	83.81	50.31	52.32	64.98	<b>71.03</b>

12% accuracy drop on easy datasets of a 110M BERT

OPT-175B	LAMBADA	HellaSwag	PIQA	WinoGrande	OpenBookQA	RTE	COPA	Average↑	WikiText↓
FP16	74.7%	59.3%	79.7%	72.6%	34.0%	59.9%	88.0%	66.9%	10.99
W8A8	0.0%	25.6%	53.4%	50.3%	14.0%	49.5%	56.0%	35.5%	93080

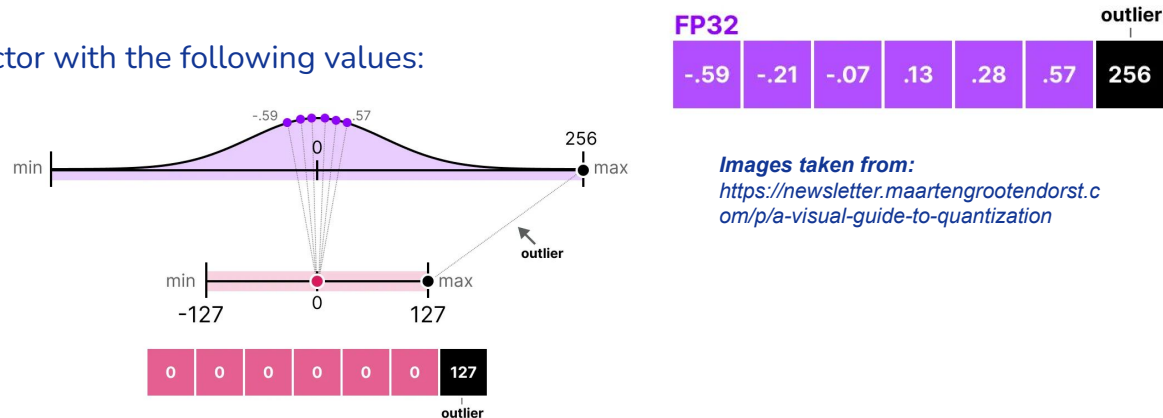
Catastrophic drop of OPT-175B

# Outliers in LLM quantization

**What are outliers?** Outliers in quantization refer to values (activations, weights, or gradients) that significantly deviate from the typical data distribution and are much larger or smaller in magnitude than most values in the tensor being quantized.

## Example:

Imagine that you have a vector with the following values:



*Images taken from:*  
<https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-quantization>

**Issue:** Because quantization maps a continuous range of values to a limited discrete set (e.g., 8-bit integers), these outliers can disproportionately affect the quantization scale and reduce the overall precision of the quantized representation.

# Outliers in LLM usually emerge in certain dimensions

## Two paradigms to solve them

### Fine-grained quantization

Normally, the scale factor and zero-point are scalars for each activation tensor to enable full integer matrix multiplication.

But there are several ways to bypass the outlier problem but requires customized cuda kernel implementation meanwhile.

- 1) Use high-precision (FP16) for problematic hidden dimensions as they're just a few, [T Dettmers et al. 2022](#)
- 2) Group quantization that groups hidden dimensions and uses different quantization parameters [Y Bondarenko et al. 2021](#)
- 3) Per-token quantization that uses different quantization parameters to different tokens [Z Yao et al. 2022](#)

### Outlier burden transfer

- 1) Transfer to weights by equivalent channel-wise scaling: [X Wei et al. 2022](#) [G Xiao et al. 2022](#), [X Wei et al. 2023](#), [W Shao et al. 2023](#). The main difference between them is on how to determine the proper  $\gamma$ .

$$XW^T = (X \oslash \gamma)(W^T \odot \gamma^T)$$

- 2) Transfer to other hidden dimensions: [J Liu et al. 2023](#) which uses channel disassembly for outlier channels and assembly for less important channels

Method	PTQ	Bits (W-E)	LLaMA			
			7B	13B	30B	65B
FP16	×	16/16	5.68	5.09	4.10	3.56
SmoothQuant* [18]	✓	8/8	11.56	10.08	7.56	6.20
LLM-QAT* [80]	×	8/8	10.30	9.50	7.10	-
OS+ [74]	✓	6/6	5.76	5.22	4.30	3.65
OmniQuant [19]	✓	6/6	5.96	5.28	4.38	3.75
QLLM [75]	✓	6/6	5.89	5.28	4.30	3.73
OS+ [74]	✓	4/4-	14.17	18.95	22.61	9.33
OmniQuant [19]	✓	4/4	11.26	10.87	10.33	9.17
QLLM [75]	✓	4/4	9.65	8.41	8.37	6.87
PEQA [81]	×	4/4	5.84	5.30	4.36	4.02

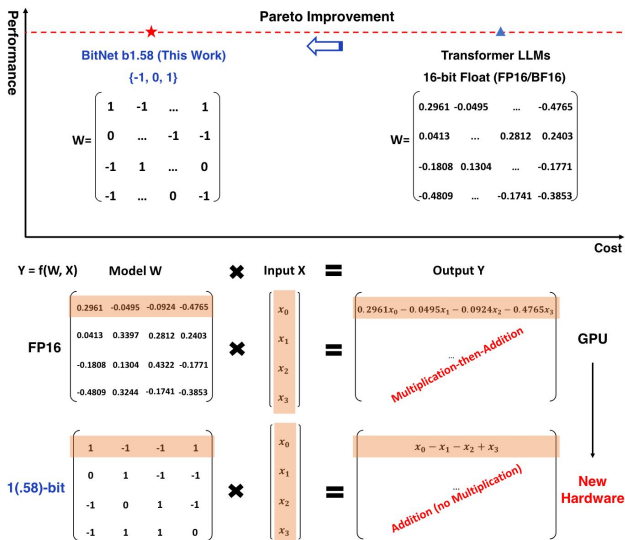
A survey paper [Y Tang et al. 2024](#)

# Training quantized LLMs

quantization-aware training of 1.(58)-bit network [S Ma et al. 2024](#)

1.58-bit weight and 8-bit activation, pre-training on 100B tokens from scratch

No matrix multiplication saves orders of energy cost



Models	Size	Memory (GB)↓	Latency (ms)↓	PPL↓
LLaMA LLM	700M	2.08 (1.00x)	1.18 (1.00x)	12.33
<b>BitNet b1.58</b>	700M	0.80 (2.60x)	0.96 (1.23x)	12.87
LLaMA LLM	1.3B	3.34 (1.00x)	1.62 (1.00x)	11.25
<b>BitNet b1.58</b>	1.3B	1.14 (2.93x)	0.97 (1.67x)	11.29
LLaMA LLM	3B	7.89 (1.00x)	5.07 (1.00x)	10.04
<b>BitNet b1.58</b>	3B	<b>2.22 (3.55x)</b>	<b>1.87 (2.71x)</b>	<b>9.91</b>
<b>BitNet b1.58</b>	3.9B	<b>2.38 (3.32x)</b>	<b>2.11 (2.40x)</b>	<b>9.62</b>

Table 1: Perplexity as well as the cost of BitNet b1.58 and LLaMA LLM.

Models	Size	ARCe	ARCc	HS	BQ	OQ	PQ	WGe	Avg.
LLaMA LLM	700M	54.7	23.0	37.0	60.0	20.2	68.9	54.8	45.5
<b>BitNet b1.58</b>	700M	51.8	21.4	35.1	58.2	20.0	68.1	55.2	44.3
LLaMA LLM	1.3B	56.9	23.5	38.5	59.1	21.6	70.0	53.9	46.2
<b>BitNet b1.58</b>	1.3B	54.9	24.2	37.7	56.7	19.6	68.8	55.8	45.4
LLaMA LLM	3B	62.1	25.6	43.3	61.8	24.6	72.1	58.2	49.7
<b>BitNet b1.58</b>	3B	<b>61.4</b>	<b>28.3</b>	<b>42.9</b>	<b>61.5</b>	<b>26.6</b>	<b>71.5</b>	<b>59.3</b>	<b>50.2</b>
<b>BitNet b1.58</b>	3.9B	<b>64.2</b>	<b>28.7</b>	<b>44.2</b>	<b>63.5</b>	<b>24.2</b>	<b>73.2</b>	<b>60.5</b>	<b>51.2</b>

Table 2: Zero-shot accuracy of BitNet b1.58 and LLaMA LLM on the end tasks.

BitNet b1.58 starts to match full precision LLaMA LLM at 3B model size in terms of perplexity, while being 2.71 times faster and using 3.55 times less GPU memory



# Training quantized LLMs

FP8 training is on the way!

## Principle

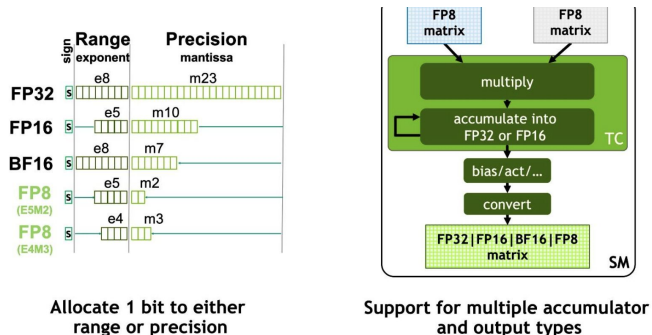


Figure 6. New NVIDIA Hopper FP8 precisions: 2x throughput and half the footprint of H100 FP16 or BF16

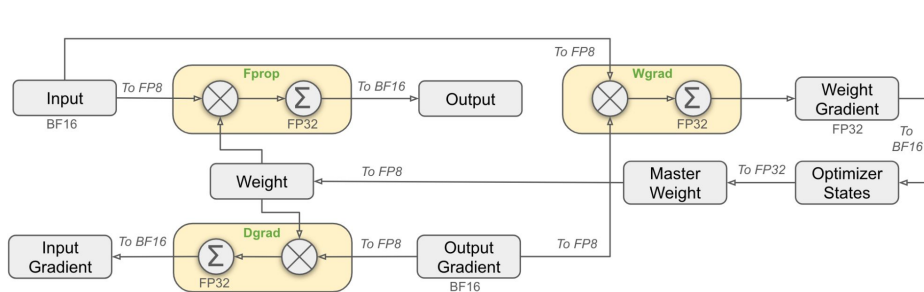


Figure 6 | The overall mixed precision framework with FP8 data format. For clarification, only the Linear operator is illustrated.

**Example:** Deepseek-v3 adopts a mixed-precision FP8 training framework

- 1) the majority of core computation kernels, i.e., GEMM operations, are implemented in FP8 precision.
- 2) maintain the original precision (e.g., BF16 or FP32) for the embedding module, the output head, MoE gating modules, normalization operators, and attention operators.
- 3) adopt fine-grained quantization with per-group scale factors,
- 4) compress cached activations and optimizer states into lower-precision formats to reduce the memory consumption and communication overhead.

Check the original paper for more details!

# Other new things in quantizing LLMs

## Alleviate memory bottleneck

Previously, memory was not a major concern, but in LLMs, alleviating the memory bottleneck with the same or a few increased FLOPs can even significantly enhance speed and throughput.

- 1) KVCache quantization: recent study even pushes it into 1-bit setting: [Q Tao et al. 2024](#), [T Zhang et al. 2024](#) [Y Zhao et al. 2023](#), [C Hooper et al. 2024](#)
- 2) Weight-only quantization: [J Lin et al. 2023](#) [C Lee et al. 2023](#)

## Scaling law study ([T Kumar et al. 2024](#))

What are the tradeoffs between precision, parameters, and data? How do they compare for pretraining and inference?

