IMPERIAL

Post-Training (1)
Fine-tuning and Instruction Tuning

08/12/2025

Shamsuddeen Muhammad
Google DeepMind Academic Fellow,
Imperial College London
https://shmuhammadd.github.io/

Idris Abdulmumin
Postdoctoral Research Fellow,
DSFSI, University of Pretoria
https://abumafrim.github.io/

Suggested Readings

- Parameter-Efficient Transfer Learning for NLP
- Prefix-Tuning: Optimizing Continuous Prompts for Generation
- The Power of Scale for Parameter-Efficient Prompt Tuning
- BitFit: Simple Parameter-efficient Fine-tuning for Transformer-based Masked Language-models
- Training Neural Networks with Fixed Sparse Masks
- Parameter-Efficient Fine-Tuning without Introducing New Latency
- LoRA: Low-Rank Adaptation of Large Language Models
- AdaLoRA: Adaptive Budget Allocation for Parameter-Efficient Fine-Tuning
- DoRA: Weight-Decomposed Low-Rank Adaptation
- QLoRA: Efficient Finetuning of Quantized LLMs
- Robust and Efficient Fine-tuning of LLMs with Bayesian Reparameterization of Low-Rank Adaptation
- Step-by-Step Unmasking for Parameter-Efficient Fine-tuning of Large Language Models

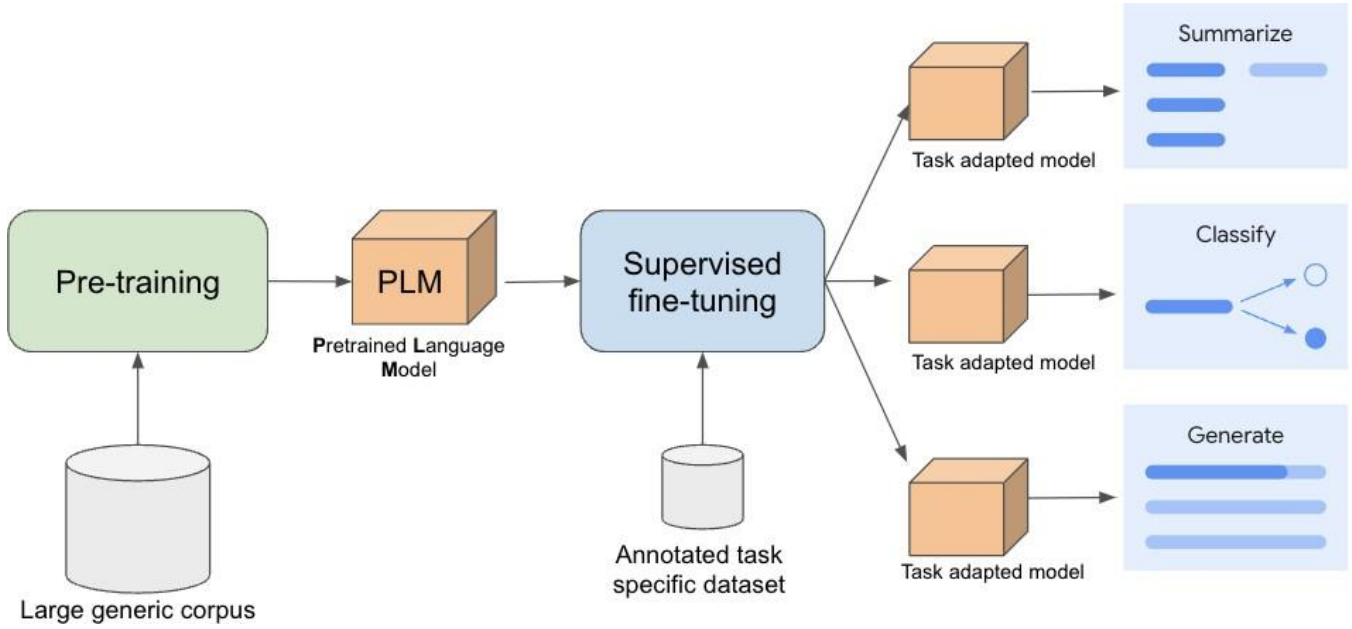
Fine-tuning

Why Fine-Tune LLMs?

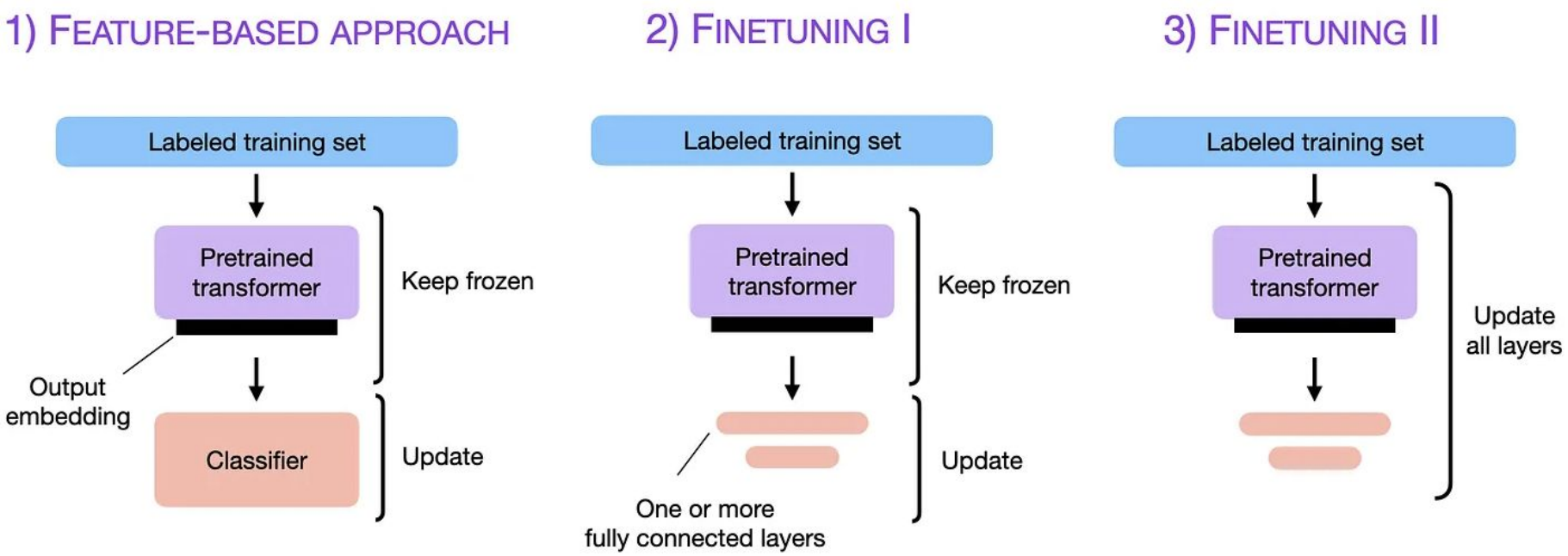
LLMs have shown remarkable generalization ability.
However, to optimize performance for specific downstream tasks, fine-tuning becomes essential.

- Adapt pretrained models to domain- specific or task-specific distributions
- Improve performance on specialized tasks example:
  - Summarization
  - Sentiment analysis
  - Code Generation etc

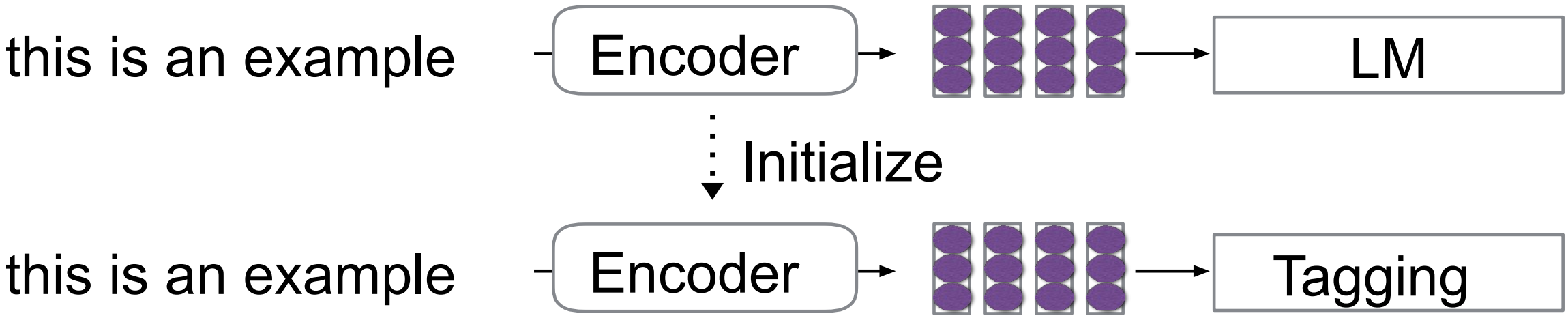
This is known as **Full Fine-tuning**.
Update all model parameters using task- specific data



Fine-Tuning Strategies



Pre-train and Fine-Tune

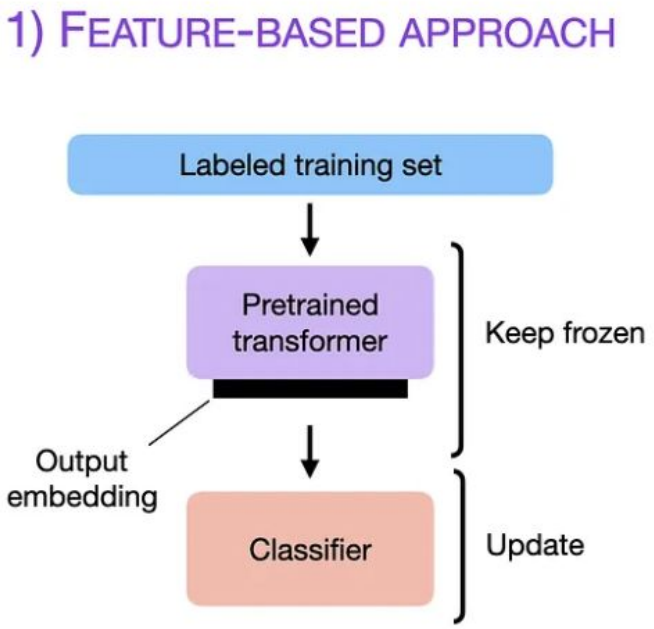


First train on one task, then train on another

Feature-Based Approach

In the feature-based approach, we load a pretrained LLM and apply it to our target dataset.

We are particularly interested in generating the output embeddings for the training set, which we can use as input features to train a classification model.

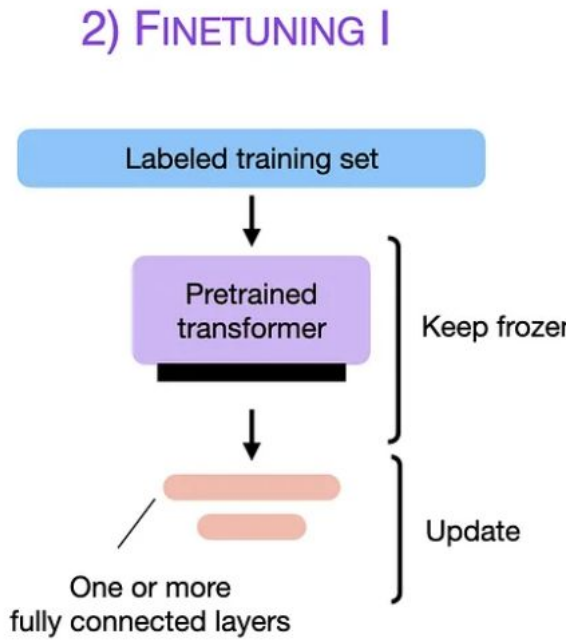


Fine-tuning Large Language Models
https://magazine.sebastianraschka.com/p/finetuning-large-language-models

Finetuning I – Updating The Output Layers

We keep the parameters of the pretrained LLM frozen.

We only train the newly **added output layers**, analogous to training a logistic regression classifier or small multilayer perceptron on the embedded features



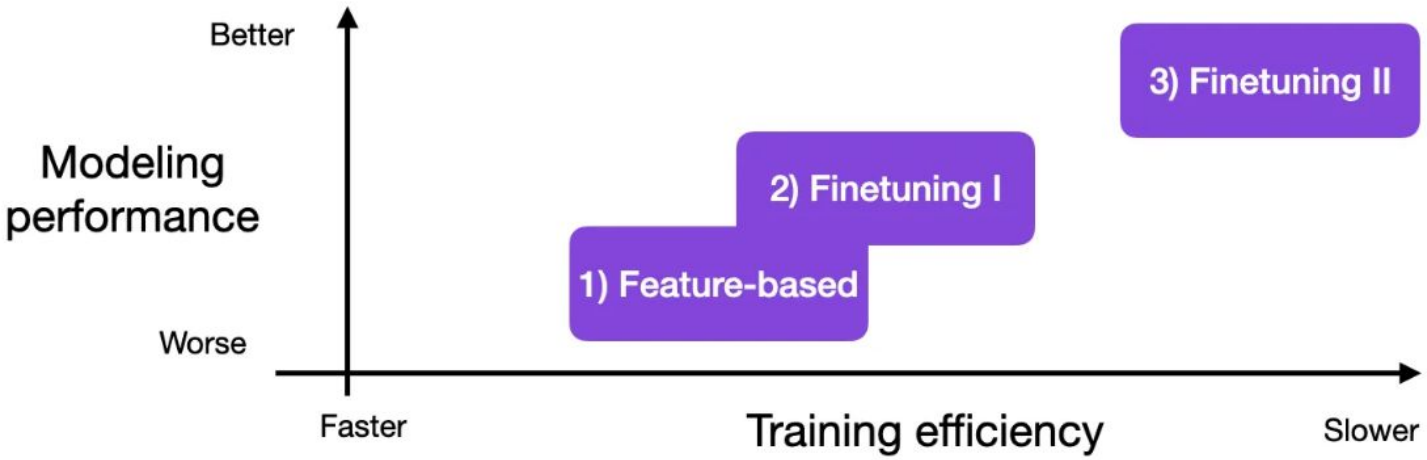
https://magazine.sebastianraschka.com/p/finetuning-large-language-models

Finetuning II – Updating All Layers

BERT paper showed that updating only the output layer can sometimes achieve performance close to full fine-tuning, this approach trains only a tiny fraction of the parameters

In practice, full fine-tuning almost always delivers better performance. For optimal results, the standard approach is to **update all model parameters**, rather than freezing the pretrained layers. Full fine-tuning is conceptually similar to partial fine-tuning, except that **no layers are frozen**.

Finetuning II – Updating All Layers



Code for fine-tuning in 3 different setups:
https://github.com/rasbt/LLM-finetuning-scripts/tree/main/conventional/distilbert-movie-review

Issue: Full Fine-Tuning

Update all the weights of the model on the new dataset.

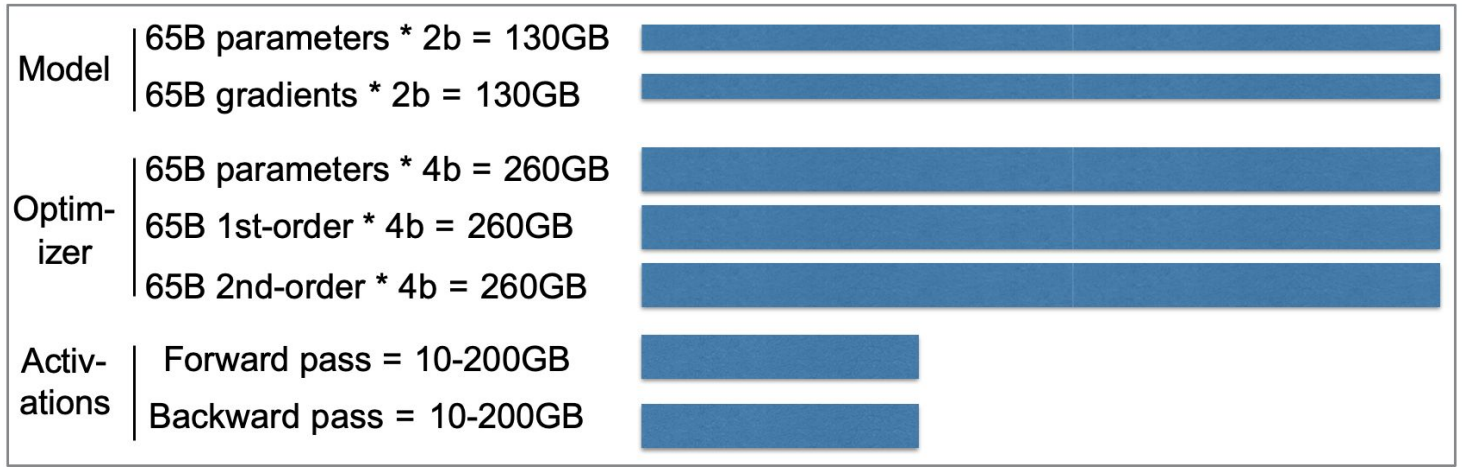
Best for:

- Large, task-specific datasets
- When you have significant computational resources

Full finetuning is ***time-consuming and expensive*** due to the computation and memory requirements.

Full Fine-tuning

- Simply continue training the LM on the output
- **Issue:** depending on optimizer, optimization method, can take lots of memory!
- **Example:** Training 65B parameter model with 16-bit mixed precision (FP16) (Rajbhandari et al. 2019)



1000-1400GB of GPU memory!

GPU Specs

GPU	Memory	Cost (2/2024)	(Cloud) Machines
T40 / K80	24GB	$150	Google Colab, AWS p2.*
V100	32GB	$2,500	Google Colab
A100	40GB or 80GB	$8,000/$16,000	Google Colab, AWS p3.*
H100	80GB	$44,000	AWS p4.*
6000 Ada, L40	48GB	$8000	N/A
Mac M*	Same as CPU	$2000	N/A

- Other hardware options:
  - AMD GPUs
  - Google TPUs
  - Special-purpose Cerebras, AWS Trainium, etc.

The GPT-3 Story (Estimation)!!

Power Consumption

GPT-3 consumed around **1,287 MWh**

 Running the **London Eye** continuously for **over 5 years**

https://arxiv.org/abs/2104.10350
https://arxiv.org/abs/2005.14165

The GPT-3 Story (Estimation)!!

GPU Consumption

GPT-3 training used **~355 GPU-years** on V100s

 Running 10,000 high-end gaming PCs at full load for 13 days straight

https://arxiv.org/abs/2104.10350
https://arxiv.org/abs/2005.14165

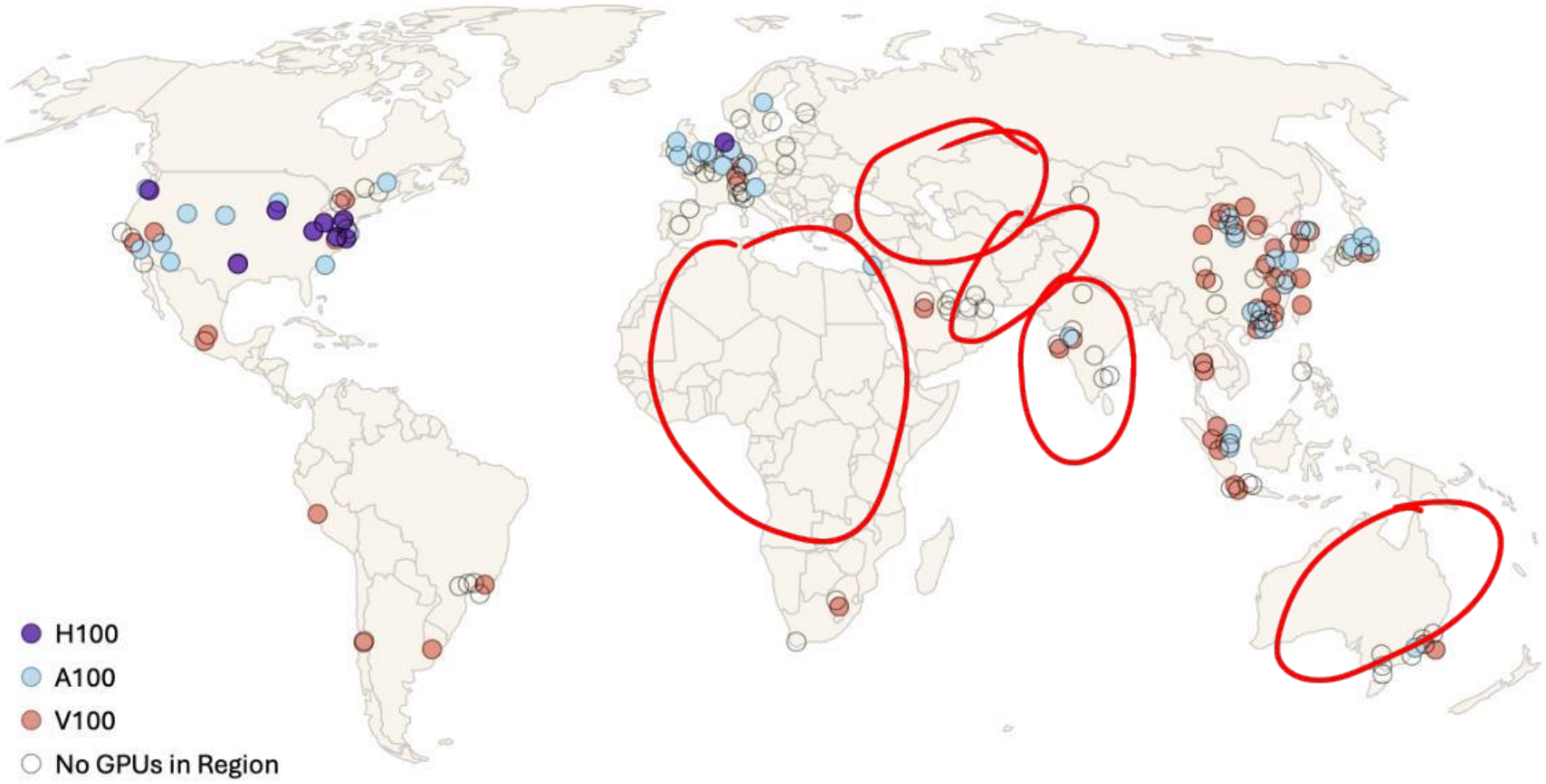
The GPT-3 Story (Estimation)!!

Water Consumption

GPT-3 required ~ **700K liters** of water

- Running a standard shower for 6 years non-stop
- Filling 1.5 million water bottles (500ml each)

AI is science for the rich, not the poor!



Approximate locations of public cloud regions and the most advanced GPU type available in each region

Image Source: Lehdonvirta, Wú, Hawkins. Compute North vs. Compute South: The Uneven Possibilities of Compute-based AI Governance Around the Globe. AIES 2024

Lets make modern LLMs **less hungry** for resource and **less thirsty** for water!!

Challenges of Full Fine-Tuning

Memory and Computation Challenges

- Billions of parameters → GPU memory bottlenecks
- Training time is extensive
- Requires storing a full model copy for every task

Deployment Challenges

- Inflexible: Cannot easily switch between tasks
- Increases cost and carbon footprint

Motivation for Alternatives

Need more:

- Efficiency
- Scalability
- Modularity

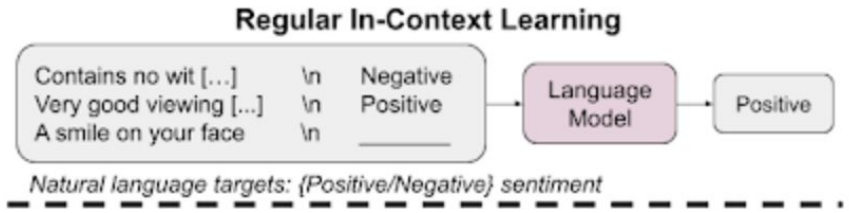
Parameter-Efficient Fine-Tuning (PEFT)

- In-Context Learning
- Introduction to PEFT
- PEFT Method Categories
  - Additive: Adapters, Prefix-tuning, Prompt-tuning
  - Selective: BitFit, FISH Mask, PaFI
  - Re-parameterization: LoRA, AdaLoRA, DoRA

In-Context Learning (ICL)

..is a technique where a large language model (LLM) performs a task by learning from examples provided directly within the prompt, without any updates to the model's underlying parameters.

- **zero examples** (zero-shot),
- **one example** (one-shot),
- **to multiple examples** (few-sho



A Survey on In-context Learning

Qingxiu Dong[1], Lei Li[1], Damai Dai[1], Ce Zheng[1], Jingyuan Ma[1], Rui Li[1], Heming Xia[2], Jingjing Xu[3], Zhiyong Wu[4], Tianyu Liu[5], Baobao Chang[1], Xu Sun[1], Lei Li[6] and Zhifang Sui[1]
[1] Peking University [2] The Hong Kong Polytechnic University
[3] ByteDance [4] Shanghai AI Lab [5] Alibaba Group [6] Carnegie Mellon University
dqx@stu.pku.edu.cn, szf@pku.edu.cn

Abstract

With the increasing capabilities of large language models (LLMs), in-context learning (ICL) has emerged as a new paradigm for natural language processing (NLP), where LLMs make predictions based on contexts augmented with a few examples. It has been a significant trend to explore ICL to evaluate and extrapolate the ability of LLMs. In this paper, we aim to survey and summarize the progress and challenges of ICL. We first present a formal definition of ICL and clarify its correlation to related studies. Then, we organize and discuss advanced techniques, including training strategies, prompt designing strategies, and related analysis. Additionally, we explore various ICL application scenarios, such as data engineering and knowledge updating. Finally, we address the challenges of ICL and suggest potential directions for further research. We hope that our work can encourage more research on uncovering how ICL works and improving ICL.

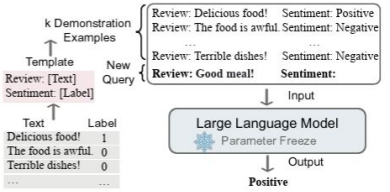


Figure 1: Illustration of in-context learning. ICL requires a prompt context containing a few demonstration examples written in natural language templates. Taking this prompt and a query as the input, large language models are responsible for making predictions.

are usually written in natural language templates. Then, ICL concatenates a query question and the piece of prompt context together to form the input, which is then fed into the language model for prediction. Different from supervised learning, which requires a training stage that uses backward gradients to update model parameters, ICL does not

In-Context Learning (ICL)

Idea - Feed it input-target examples ("shots")

Advantages

- No training required
- Fast and flexible

Disadvantages

- Prompt tokens consume context window
- Does not generalize or adapt across sessions
- Cannot learn from errors or improve performance

Parameter-Efficient Fine-Tuning (PEFT)

What is PEFT?

- A set of strategies to adapt large models by training only a small subset of parameters
- Core model remains frozen; auxiliary or selective parameters are updated

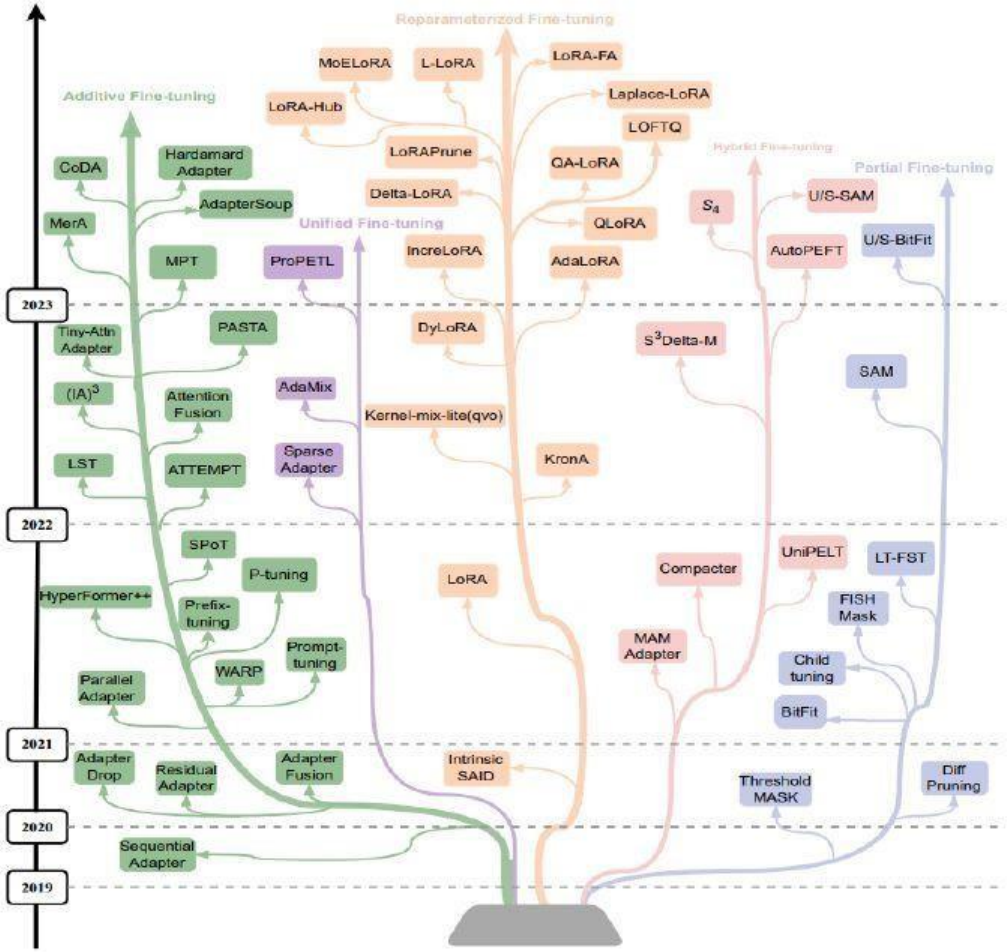
Motivations

- Reduce compute and memory footprint
- Allow reusability and modular adaptation
- Enable personalization and domain-specific training on edge devices

Three Main Categories

- **Additive Methods:** Add new modules (e.g., adapters, prompts)
- **Selective Methods:** Selectively update a sub-set of existing parameters
- **Re-parameterization Methods:** Re-express parameters to enable efficient tuning

Parameter-Efficient Fine-Tuning (PEFT)



PARAMETER-EFFICIENT FINE-TUNING METHODS FOR PRETRAINED LANGUAGE MODELS: A CRITICAL REVIEW AND ASSESSMENT

- Development of Additive and Selective PEFT methods started in early 2016
- Re-parameterization-based method garnered interest from 2022
- Over the past few years, re-parameterization-based methods have got wider-spread popularity, due to their flexibility with LLMs

Additive Methods - Adapters

Core Idea

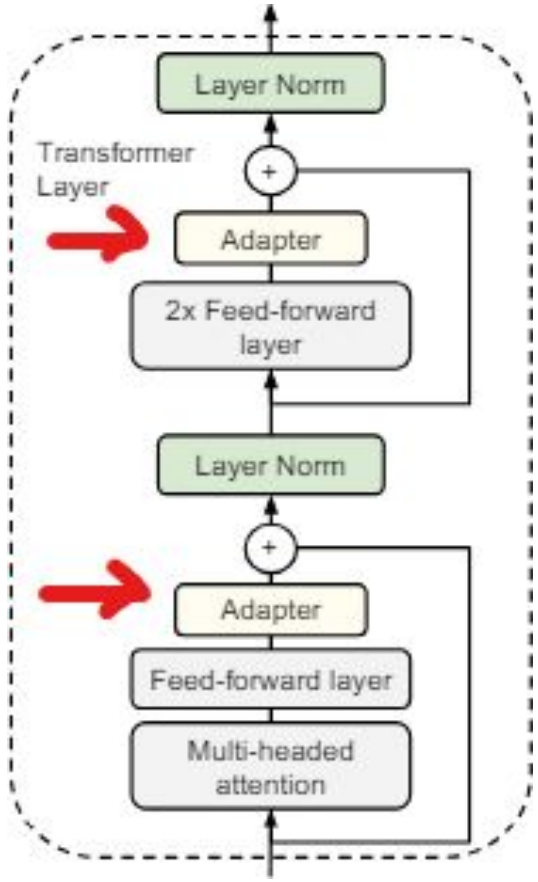
- Insert adapter layers in between the layers of transformer architecture
- Only adapters are updated; base model remains frozen

Advantages

- Now we can train and plug-in task specific adapters
- Small memory needed to save many task specific adapters

Disadvantages

- Increase in inference time due to additional layers
- Needs code changes to integrate into model architecture



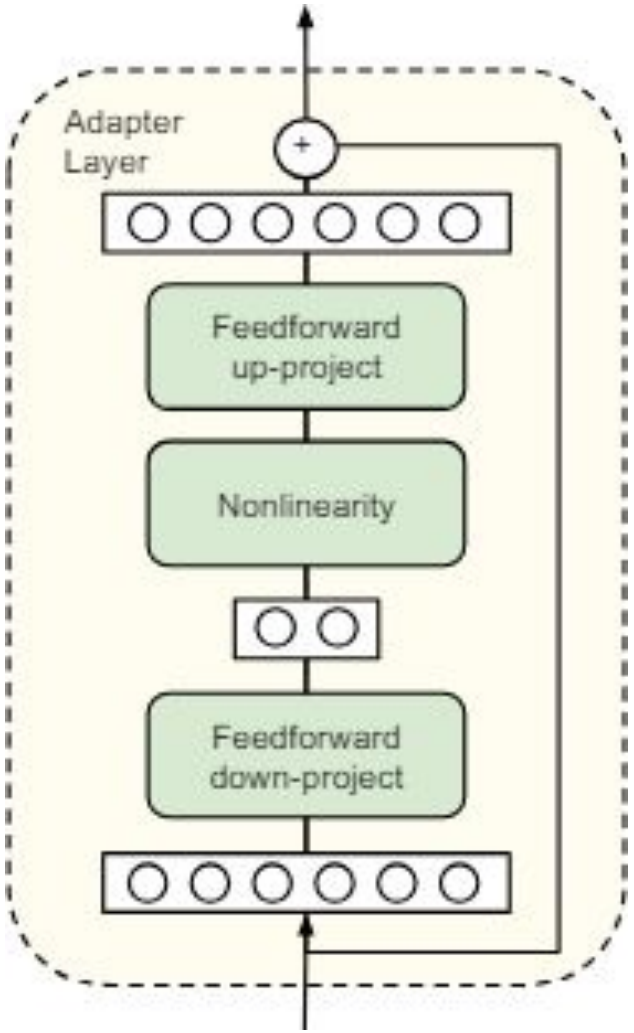
Additive Methods - Adapters

The Adapter Layer - Bottleneck structure

- Each adapter layer consists of one downward projection matrix, followed by a non-linear activation function and finally an upward projection
- There is also a residual connection

Results

- In $BERT_{large}$, with 3.6% additional parameters (adapters), able to achieve **99.6%** performance of full-finetuning



Additive Methods – Prefix-Tuning

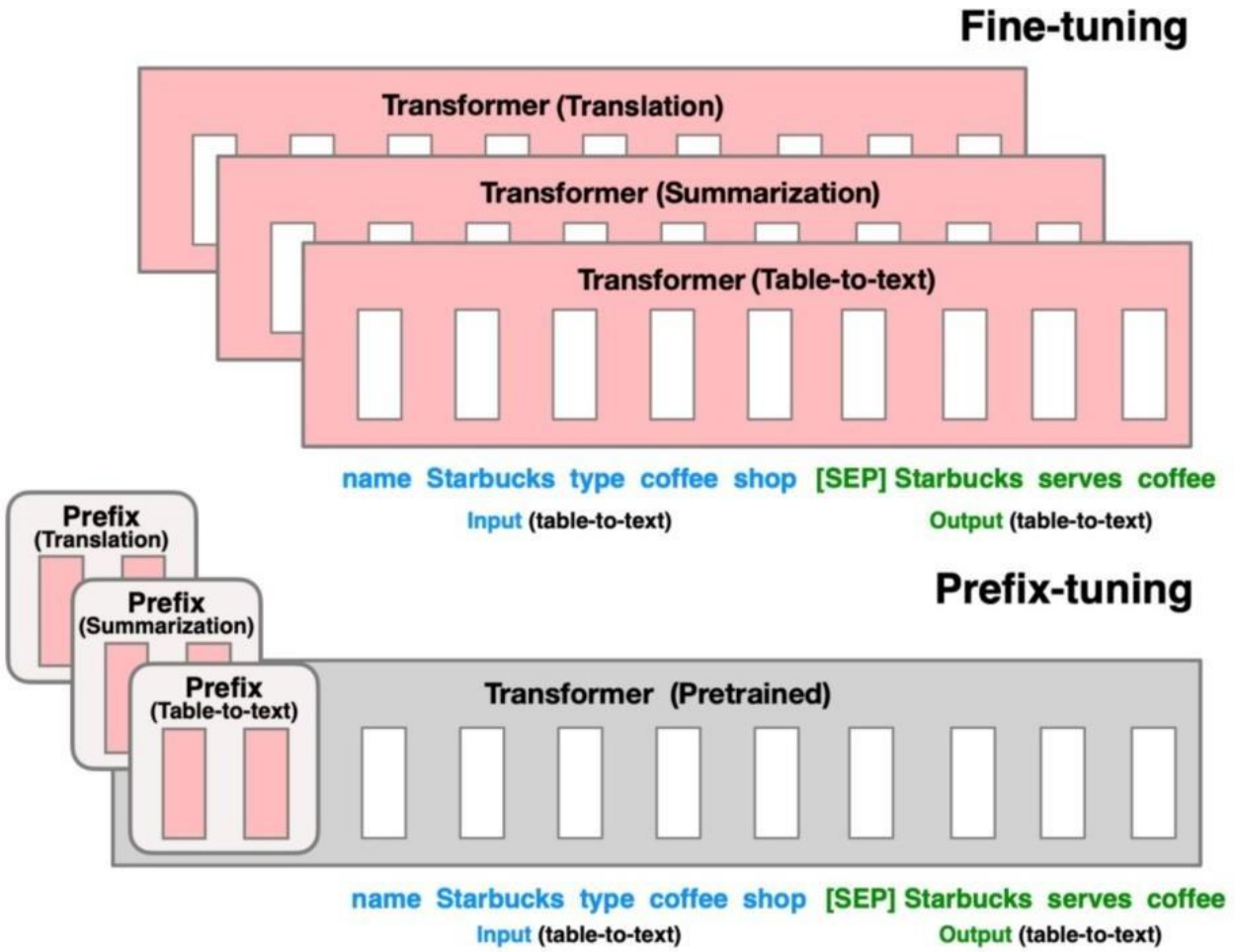
Core Idea:

- Learn task-specific prefix key/value vectors that are prepended to each layer's self-attention

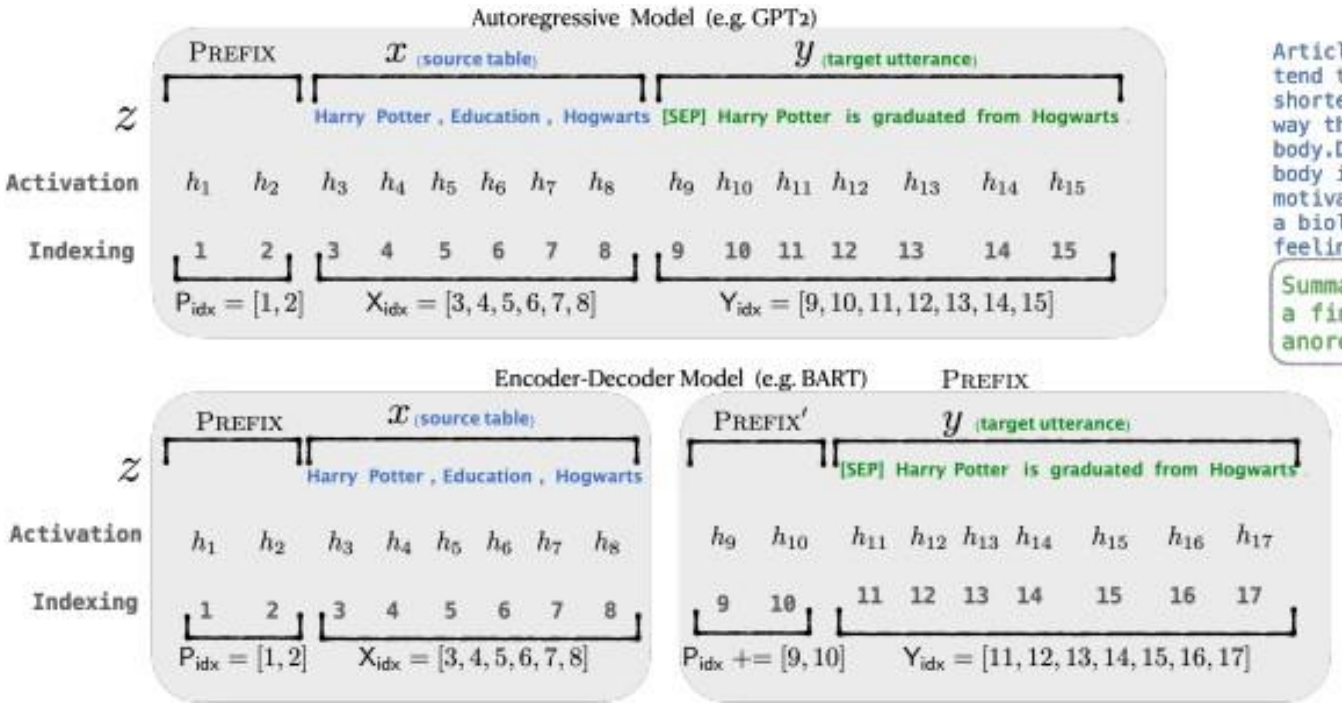
Method:

Learnable vector P is called **Soft Prompt**

input X of length n
prefix P of length m
modified input X' = [P; X]
shape of X' = (n+m) x d
where d is model's hidden size



Additive Methods – Prefix-Tuning



Summarization Example

Article: Scientists at University College London discovered people tend to think that their hands are wider and their fingers are shorter than they truly are.They say the confusion may lie in the way the brain receives information from different parts of the body.Distorted perception may dominate in some people, leading to body image problems ... [ignoring 308 words] could be very motivating for people with eating disorders to know that there was a biological explanation for their experiences, rather than feeling it was their fault."

Summary: The brain naturally distorts body image – a finding which could explain eating disorders like anorexia, say experts.

Table-to-text Example

Table: name[Clowns] customer-rating[1 out of 5] eatType[coffee shop] food[Chinese] area[riverside] near[Clare Hall]

Textual Description: Clowns is a coffee shop in the riverside area near Clare Hall that has a rating 1 out of 5 . They serve Chinese food .

- During fine-tuning, the parameters of P are optimised for the specific task
- Prefix vectors are added to key and value but omitted from the query vector

Additive Methods – Prompt-Tuning

Core Idea:

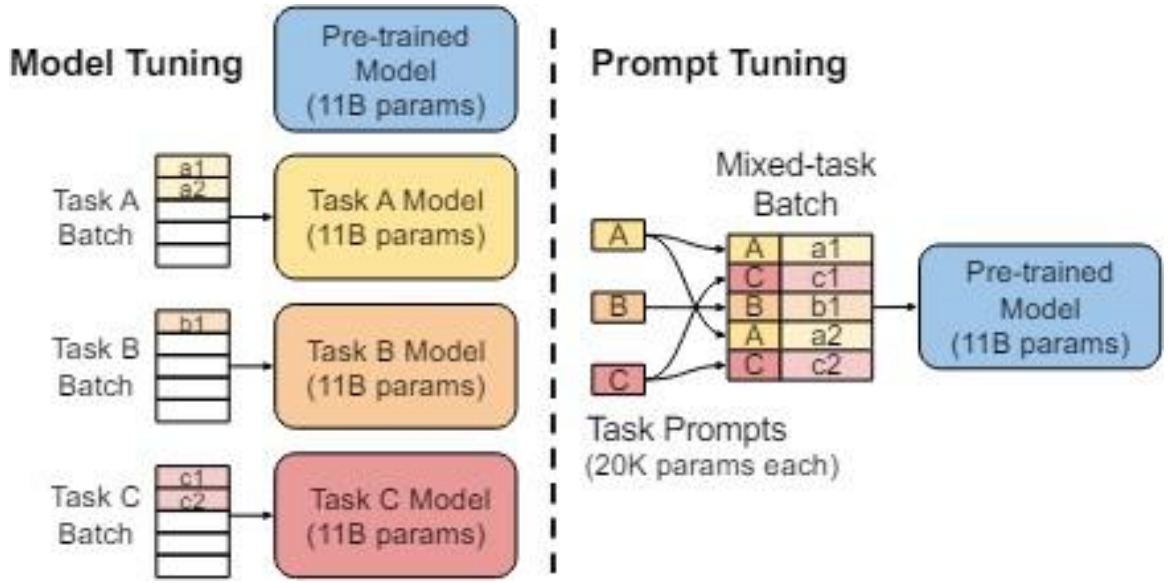
- Learn a continuous prompt embedding prepended to the input

Method:

Input is: [Prompt Embeddings] + [Original Tokens] Only prompt embeddings are updated

Advantages: Multi-task input in a single batch (Training) Very few parameters (< 0.1%)

Disadvantages: Less effective than prefix-tuning in low-data regimes



Selective Methods - Structured and Unstructured

Selective Methods:
- Selectively update a sub-set of existing parameters

Structured:
- Update specific components (e.g., attention weights only, FFN weights)
- Basically, update the entire tensor parameter (coarse update)

Unstructured:
- Update arbitrary subset of parameters (often selected by heuristics)
- A sub-set of scalar parameters in a tensor parameter (finer granularity)

Selective Methods - BitFit (Zaken et al.)

Idea:

- Only update bias terms of transformer layers

$$\mathbf{Q}^{m,\ell}(\mathbf{x}) = \mathbf{W}_q^{m,\ell}\mathbf{x} + \mathbf{b}_q^{m,\ell}$$
$$\mathbf{K}^{m,\ell}(\mathbf{x}) = \mathbf{W}_k^{m,\ell}\mathbf{x} + \mathbf{b}_k^{m,\ell}$$
$$\mathbf{V}^{m,\ell}(\mathbf{x}) = \mathbf{W}_v^{m,\ell}\mathbf{x} + \mathbf{b}_v^{m,\ell}$$

BERT encoder model

$$\mathbf{h}_2^\ell = \text{Dropout}\left(\mathbf{W}_{m_1}^\ell \cdot \mathbf{h}_1^\ell + \mathbf{b}_{m_1}^\ell\right) \quad (1)$$
$$\mathbf{h}_3^\ell = \mathbf{g}_{LN_1}^\ell \odot \frac{(\mathbf{h}_2^\ell + \mathbf{x}) - \mu}{\sigma} + \mathbf{b}_{LN_1}^\ell \quad (2)$$
$$\mathbf{h}_4^\ell = \text{GELU}\left(\mathbf{W}_{m_2}^\ell \cdot \mathbf{h}_3^\ell + \mathbf{b}_{m_2}^\ell\right) \quad (3)$$
$$\mathbf{h}_5^\ell = \text{Dropout}\left(\mathbf{W}_{m_3}^\ell \cdot \mathbf{h}_4^\ell + \mathbf{b}_{m_3}^\ell\right) \quad (4)$$
$$\text{out}^\ell = \mathbf{g}_{LN_2}^\ell \odot \frac{(\mathbf{h}_5^\ell + \mathbf{h}_3^\ell) - \mu}{\sigma} + \mathbf{b}_{LN_2}^\ell \quad (5)$$

- Bias terms are small in number but influential
- Extremely light-weight and easy to implement
- But it is architecture dependent (certain model architectures, e.g., Llama, do not have bias terms)

Selective Methods - PaFI (Liao et al.)

Idea:
- Select parameters based on magnitude

Steps:
1. Rank parameters based on magnitude
2. Select smallest - K% as trainable
3. Keep rest frozen

Advantages:
- Fixed masks
- Mask generation based on only magnitude (less computation cost)

Intuitio
n
- Important parameters of a PLM learn a more generic representation (params with high magnitude)
- These params are important for downstream task as well
- Hence, keep these fixed and update low-magnitude params in fine-tuning
- We can argue that the important parameters of a PLM are also important for downstream task and needs to be updated. But authors empirically show that this is not the case.

Selective Methods – ID3 (Agarwal et al.)

ID3-PEFT refers to a parameter-efficient fine-tuning (PEFT) approach that applies the intuition of the ID3 decision-tree algorithm to automatically select which parameters of an LLM should be tuned.

ID3-PEFT uses **information-gain–based selection** to identify *which* layers or parameters of a large language model should be fine-tuned, instead of tuning all possible PEFT modules.

Selective Methods – ID3 (Agarwal et al.)

Core Idea

Instead of manually deciding:

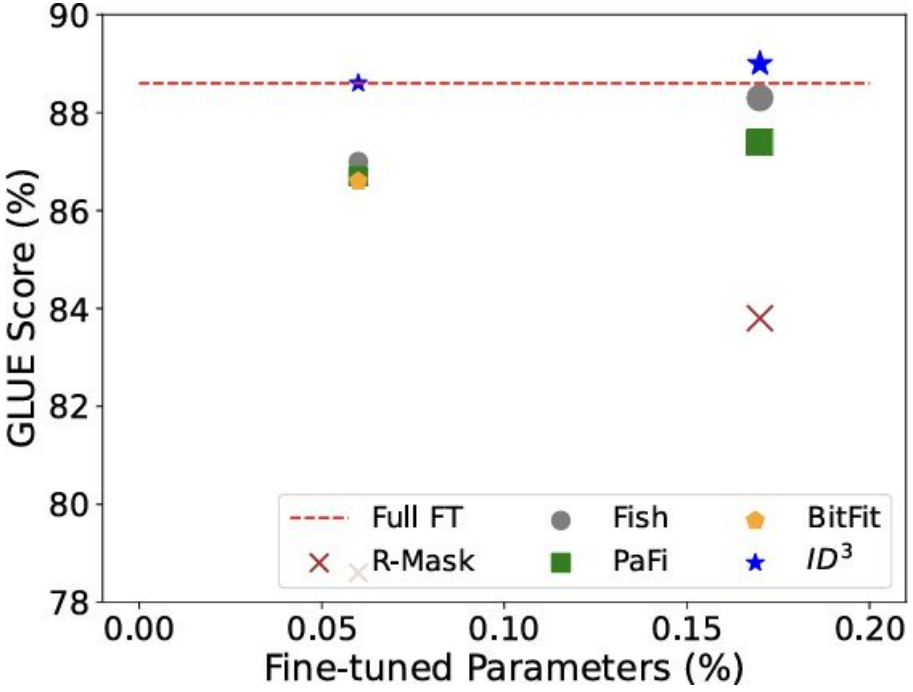
- which layers get LoRA,
- where to add adapters,
- what rank to use,

ID3-PEFT uses an information-gain criterion to select the *most impactful* parts of the model for fine-tuning.

It automatically chooses the layers that contribute the most to reducing loss on the new target task.

Selective Methods – Performance

Performance on GLUE with DeBERTa-V3 model



- BitFit tends to perform relatively poor as compared to full fine-tuning or methods like PaFi or Fish mask
- As higher budget (K), PaFi and Fish mask can match the performance of FFT, even after fine-tuning only 0.15% of all parameters
- ID3 performs significantly better, even at lower budget, often outperforming full fine- tuning and other selective baselines

Selective Methods – Performance

Performance on math reasoning tasks

Model	Budget	Method	AddSub	MultiArith	SingleEq	GSM8K	AQuA	SVAMP	Avg.
LLaMA-7B	56M	LoRA (r=32)	81.3	95.5	81.7	34.1	17.7	46.7	59.5
	3.5M	LoRA (r=2)	78.2	**96.7**	76.6	**35.3**	**16.9**	44.9	58.1
	3.5M	PaFi + LoRA (r=32)	78.7	92.3	76.8	33.9	**16.9**	43.2	57.0
	3.5M	$\texttt{ID}^3$ + LoRA (r=32)	**80.7**	95.8	**79.3**	34.3	15.7	**45.7**	**58.6**
Qwen-7B	54M	LoRA (r=32)	94.4	98.2	97.6	76.9	34.6	85.8	81.3
	3.4M	LoRA (r=2)	**93.9**	98.3	96.4	76.4	31.9	86.8	80.6
	3.4M	PaFi + LoRA (r=32)	91.1	**99.0**	**97.0**	**78.5**	**37.8**	85.8	**81.5**
	3.4M	$\texttt{ID}^3$ + LoRA (r=32)	93.6	98.5	95.1	77.9	37.0	**87.1**	**81.5**
Qwen-3B	40M	LoRA (r=32)	92.1	98.5	95.9	71.9	34.2	81.5	79.0
	2.5M	LoRA (r=2)	**92.9**	97.5	94.9	70.8	34.6	**85.1**	79.3
	2.5M	PaFi + LoRA (r=32)	90.9	97.8	**96.2**	70.6	36.2	83.9	79.3
	2.5M	$\texttt{ID}^3$ + LoRA (r=32)	92.6	**98.2**	95.9	**71.5**	**37.4**	83.9	**79.9**
Qwen-1.5B	25M	LoRA (r=32)	90.4	98.2	96.6	65.8	36.6	75.3	77.2
	1.5M	LoRA (r=2)	**91.9**	**98.2**	95.5	62.8	31.1	80.9	76.7
	1.5M	PaFi + LoRA (r=32)	89.4	96.7	**95.9**	**64.5**	32.3	78.4	76.2
	1.5M	$\texttt{ID}^3$ + LoRA (r=32)	91.6	97.8	93.7	62.6	**34.6**	**81.0**	**76.9**

- Due to dependence on model architecture, BitFit is less adaptive with LLMs and adapters like LoRA (more in the next slides).
- PaFi and ID3 can be seamlessly integrated with LLMs, with/without adapters.

Re-parameterization Methods

Idea
- Modify the form of parameters to allow efficient learning

Common trick
- Low-rank approximation: Replace weight W with W + AB

Popular Methods
- LoRA
- AdaLoRA
- DoRA
- QLoRA
- etc

Re-parameterization Methods - Results

Method	Metric	MRPC	CoLA	RTE	WiC	BoolQ	SST2	QNLI	QQP	MNLI	Average
Full FT	Accuracy ↑	89.2	84.0	76.5	70.4	80.4	**94.6**	**92.2**	**88.4**	**85.6**	**84.6**
LoRA		89.9	**85.1**	80.1	67.5	80.0	93.1	89.1	87.0	83.0	83.9
AdaLoRA		88.7	84.5	**81.2**	67.7	**80.6**	92.4	89.7	86.6	84.2	84.0
MonteCLoRA		**91.2**	84.9	**81.2**	**71.3**	79.9	92.4	89.8	85.7	83.5	84.4
Full FT	NLL ↓	0.34	**0.40**	0.60	0.68	0.51	0.26	0.28	**0.28**	**0.39**	0.42
LoRA		0.32	**0.40**	0.54	0.62	**0.49**	**0.20**	0.27	0.31	0.44	0.40
AdaLoRA		0.42	0.54	0.62	0.83	0.55	0.21	0.28	0.30	0.42	0.46
MonteCLoRA		**0.31**	0.41	**0.46**	**0.60**	0.50	0.21	**0.26**	0.32	0.44	**0.39**

- MonteCLoRA > AdaLoRA > LoRA with RoBERTa-base on GLUE tasks.
- In terms of accuracy, full fine-tuning does better than most other reparameterization-based PEFT methods, at the expense of higher training compute.

Re-parameterization Methods – Results (Llama)

Method	PiQA (↑)	Social(↑)	WinoGrande(↑)	ARC-e(↑)	ARC-c(↑)	OpenbookQA(↑)	Average(↑)
Adapter	73.0	68.3	64.0	72.4	57.1	67.4	67.0
LoRA	77.7	**71.5**	**67.7**	79.2	62.8	75.2	**72.3**
DoRA	72.6	69.7	64.9	78.0	58.8	73.4	69.6
MonteCLoRA	**78.0**	71.0	65.8	**79.9**	**62.9**	**76.0**	**72.3**

PEFT	SM (max)↑	FE (max)↑	SM (ext. robust.)↑	FE (ext. robust.)↑
LoRA	0.70	0.71	0.65	0.65
DoRA	0.75	0.75	0.67	0.67
LoRA+	0.76	0.76	0.71	0.71
IA^3	0.76	0.77	0.74	0.75
Prompt Tuning	0.71	0.71	0.68	0.68
MonteCLoRA	**0.78**	**0.78**	**0.75**	**0.76**

PEFT	pass@2 (max)↑	pass@4 (max)↑	pass@2 (ext. robust.)↑	pass@4 (ext. robust.)↑
LoRA	0.52	0.58	0.50	0.55
DoRA	0.55	0.60	0.51	0.54
LoRA+	0.56	0.60	0.53	0.58
IA^3	0.56	0.63	**0.56**	**0.62**
Prompt Tuning	0.45	0.54	0.45	0.54
MonteCLoRA	**0.58**	**0.65**	0.55	0.61

- MonteCLoRA performs significantly better with LLMs (here LLaMA series models) than LoRA and DoRA.
- Commonsense reasoning, GSM8k (math reasoning) and HumanEval (code generation) tasks were evaluated

PEFT Summary

PEFT Landscape:

- **Additive**: Adapters, Prompt-tuning, Prefix-tuning
- **Selective**: BitFit, FISH Mask, PaFI, ID3
- **Re-parameterization**: LoRA, AdaLoRA, DoRA, MonteCLoRA, QLoRA

Key Takeaways:

- PEFT enables efficient, modular fine-tuning for LLMs
- Different methods offer trade-offs in parameter count, complexity, and performance
- Choosing the right PEFT method depends on task requirements, compute budget, and deployment constraints

PEFT Reading

1. Houlsby et al., Adapters: Parameter-Efficient Transfer Learning for NLP
2. Li et al., Prefix-Tuning: Optimizing Continuous Prompts for Generation
3. Lester et al., Prompt-Tuning: The Power of Scale for Parameter-Efficient Prompt Tuning
4. Ben Zaken et al., BitFit: Simple Parameter-efficient Fine-tuning for Transformers
5. Sung et al., Training Neural Networks with Fixed Sparse Masks
6. Liao et al., Parameter-Efficient Fine-Tuning without Introducing New Latency
7. Hu et al., LoRA: Low-Rank Adaptation Of Large Language Models
8. Zhang et al., ADALORA: Adaptive Budget Allocation for Parameter-Efficient Fine-Tuning
9. Liu et al., DoRA: Weight-Decomposed Low-Rank Adaptation
10. Dettmers et al., QLORA: Efficient Finetuning of Quantized LLMs
11. Sengupta et al., Robust and Efficient Fine-tuning of LLMs with Bayesian Reparameterization of Low-Rank Adaptation, TMLR, 2025.
12. Agarwal et al., Step-by-Step Unmasking for Parameter-Efficient Fine-tuning of Large Language Models, TACL, 2025.