Parameter-Efficient Fine-Tuning (PEFT)

Tanmoy Chakraborty
Associate Professor, IIT Delhi
https://tanmoychak.com/

Advances in Large Language Models

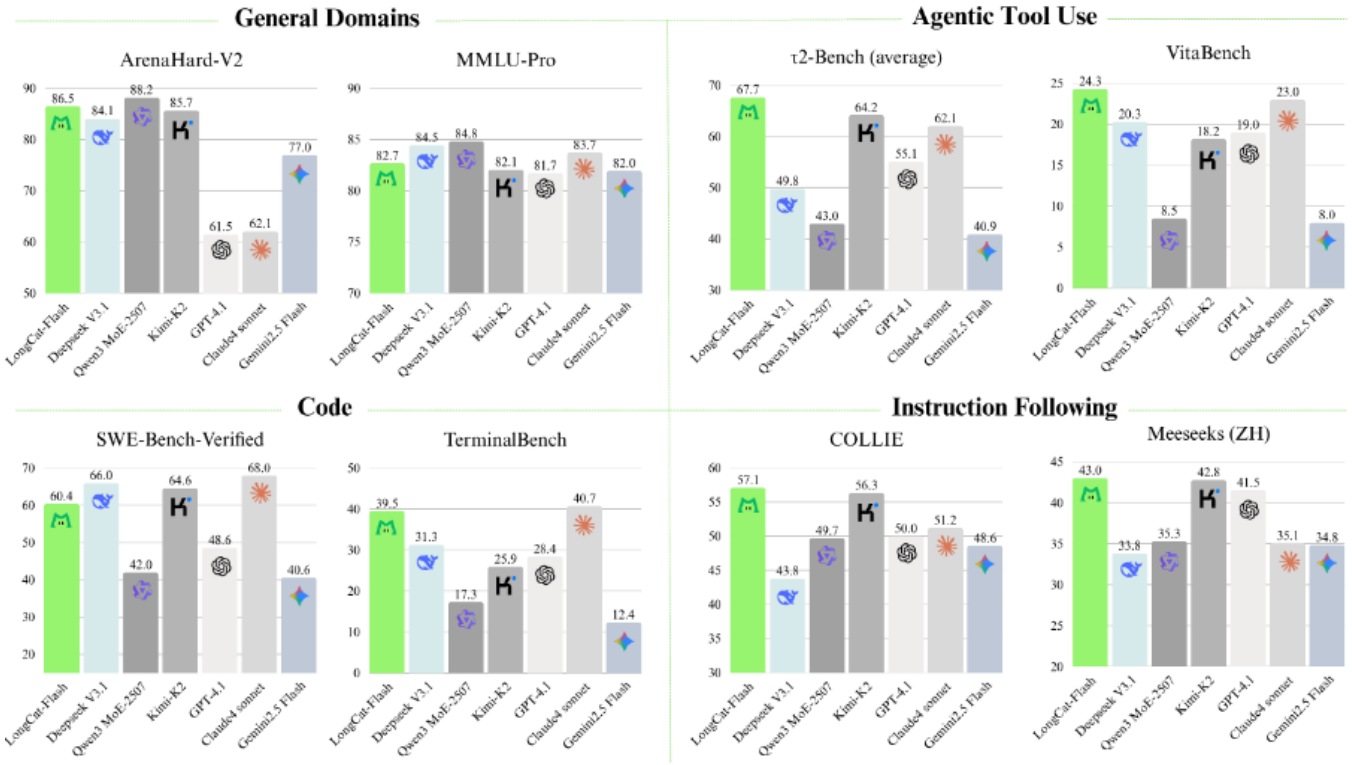
LongCat-Flash-Chat

Meituan, China's largest food delivery company released its open weight LLM

Announced on August 31, 2025

LongCat Blog

LongCat-Flash achieves advanced agentic capabilities via a multi-stage pipeline, focusing on reasoning, coding, and supporting a long context length of 128k. It uses a shortcut-connected design which allows faster inference even at very large scales



LongCat-Flash is an innovative MoE architecture with a total of 560B parameters, of which it dynamically activates only 18.6-31.3 billion parameters per token, optimizing both performance and efficiency.

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://lambdalabs.com/blog/demystifying-gpt-3/

The GPT-3 Story (Estimation)!!

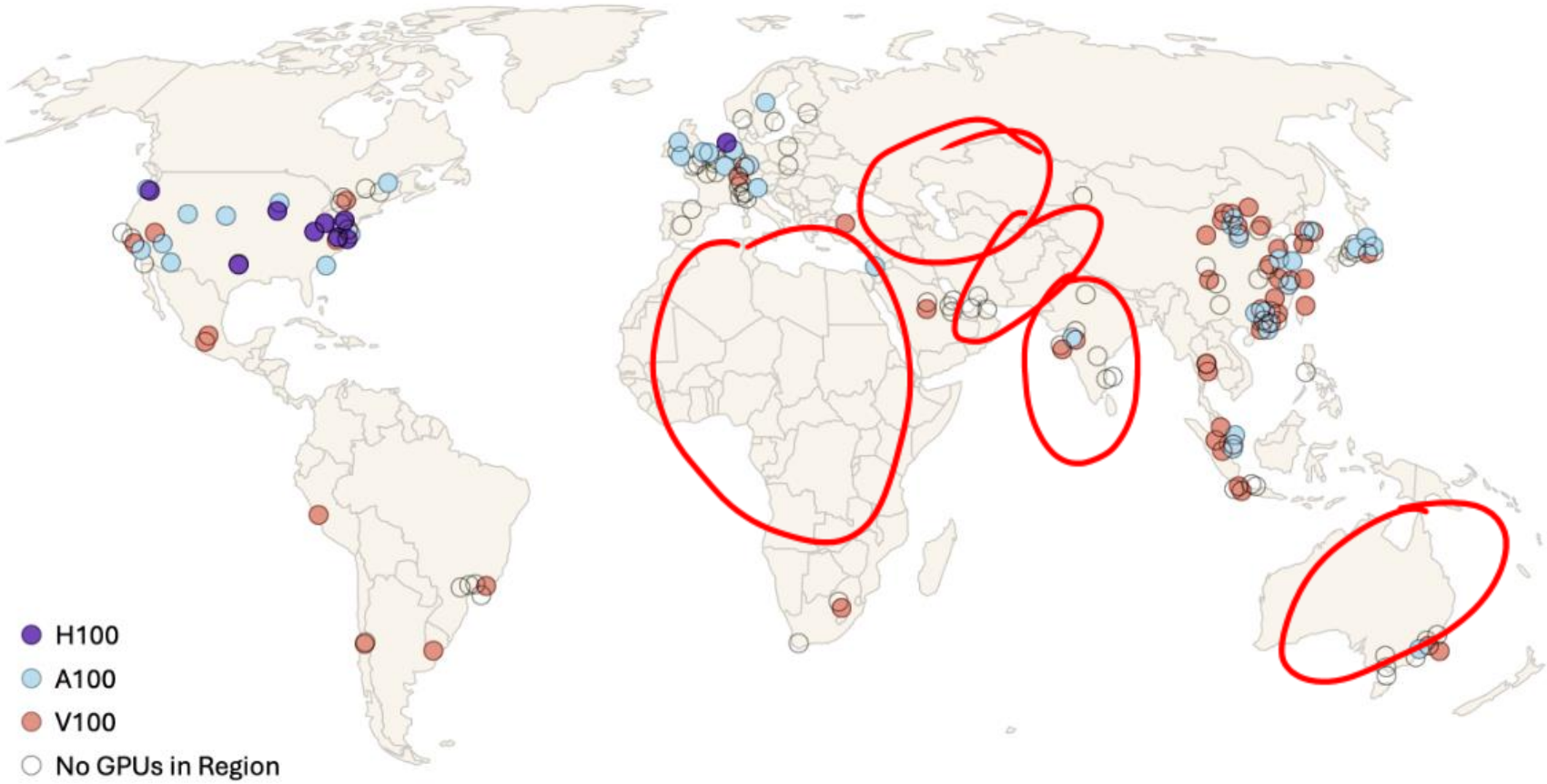
Water Consumption

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

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

https://arxiv.org/abs/2304.03271

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!!

Today's Agenda -- PEFT

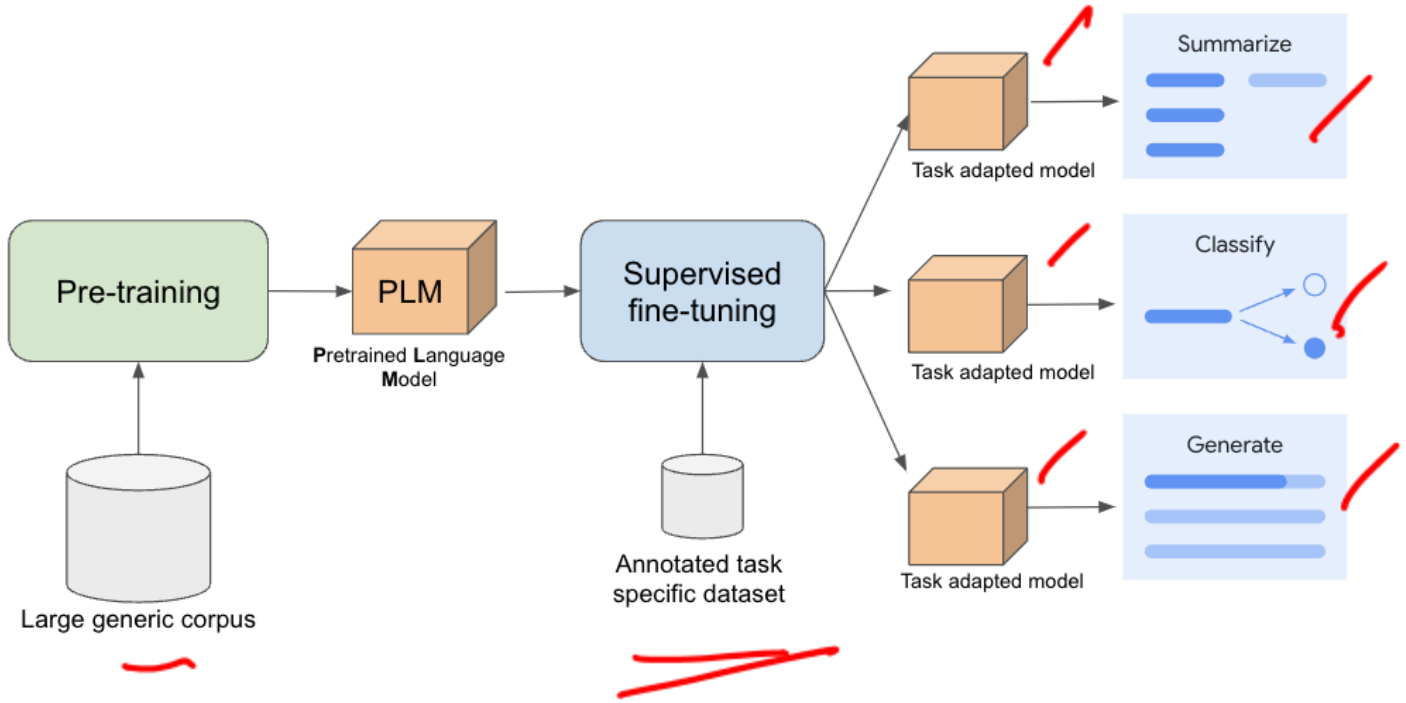
- Why Fine-Tune LLMs?
- Challenges of Full Fine-Tuning
- In-Context Learning
- Introduction to PEFT
- PEFT Method Categories
  - Additive: Adapters, Prefix-tuning, Prompt-tuning
  - Selective: BitFit, FISH Mask, PaFI
  - Re-parameterization: LoRA, AdaLoRA, DoRA
- Summary & Takeaways

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



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

In-Context Learning (ICL)

Idea - Feed it input-target examples (“shots”)

Example of a 4-shot input

Please unscramble the letters into a word, and write that word:

- asinoc=casino
- yfrogg=froggy
- plesim=simple
- iggestb=biggest
- astedro=?

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

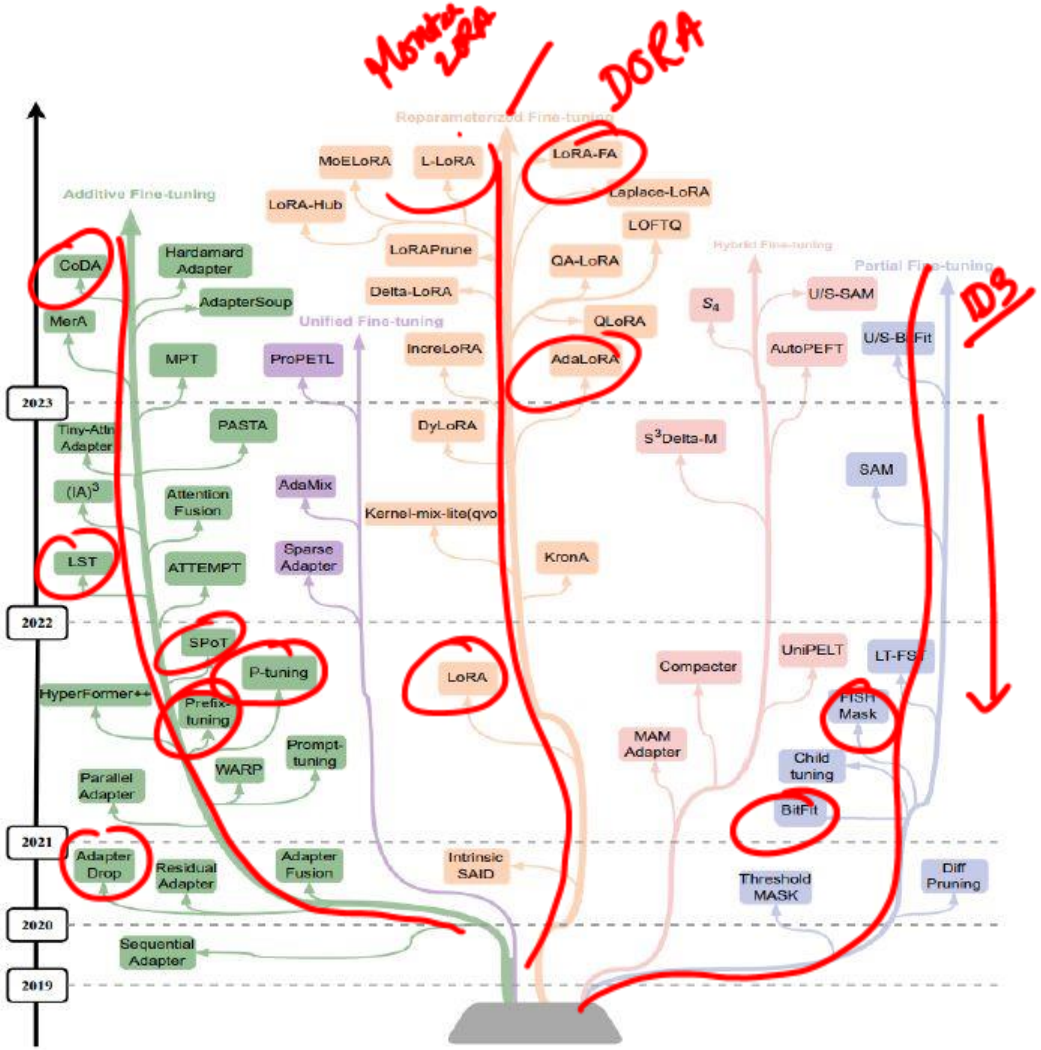
Motivations

- 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)



- Development of Additive and Selective PEFT methods started in early 2019
- 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

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

Additive Methods - Adapters

Core Idea

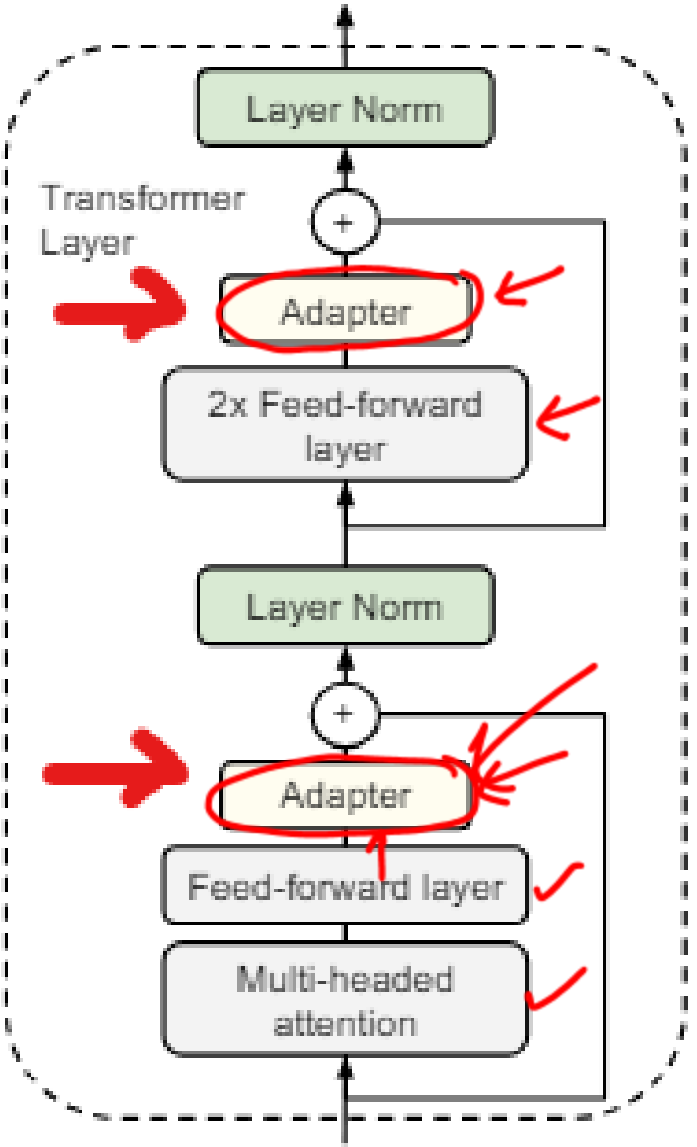
- Insert adapter layers in between the layers of transformer architecture
- Only adapters are updated; base model remains frozen

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



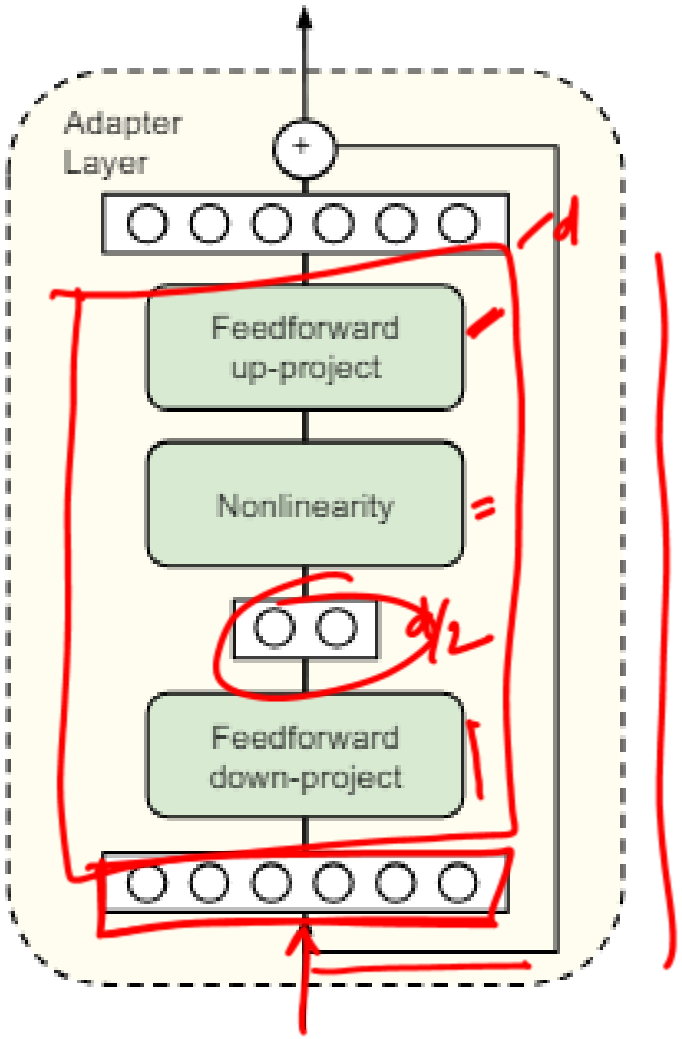
Additive Methods - Adapters

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***$_{\text{large}}$, 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**

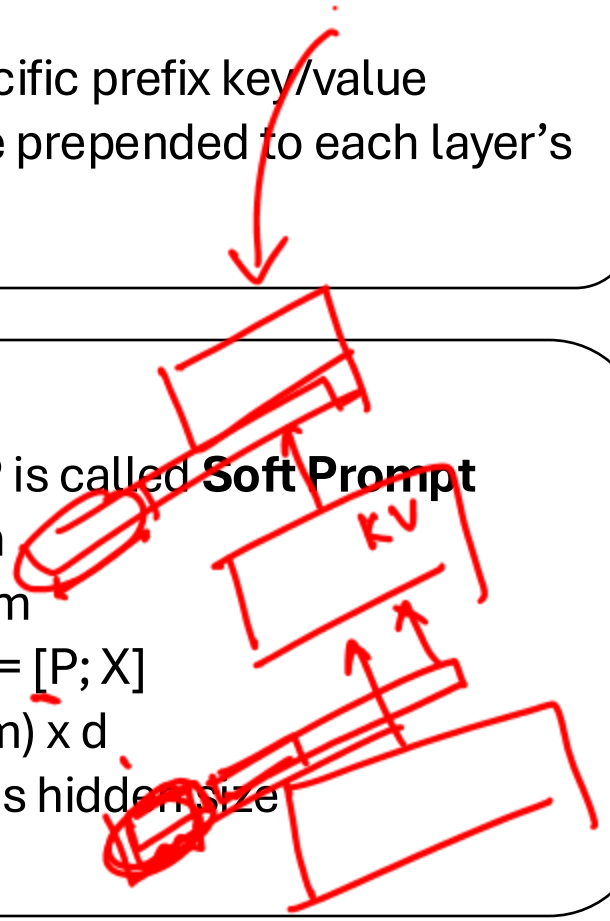
input X of length n

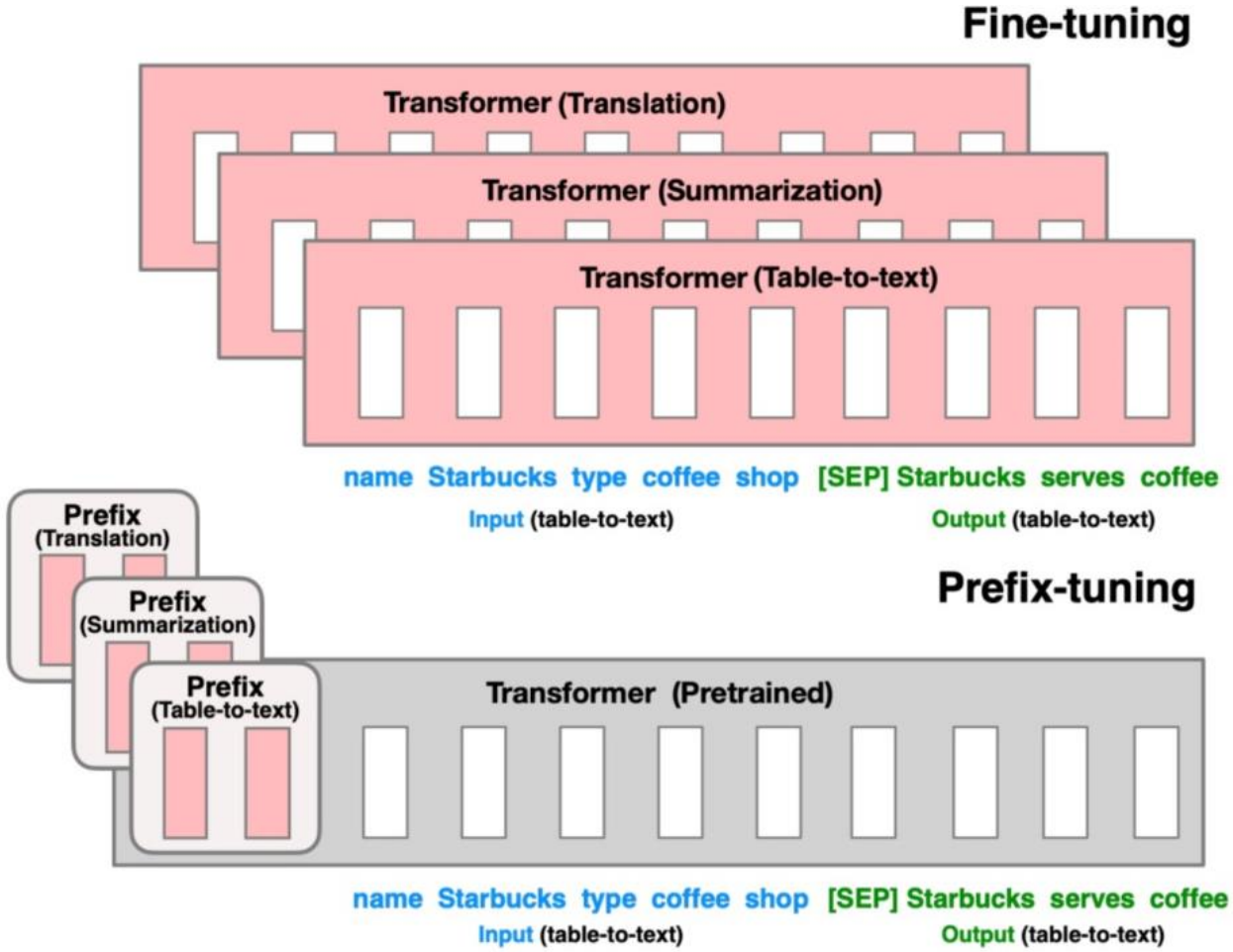
prefix P of length m

modified input X' = [P; X]

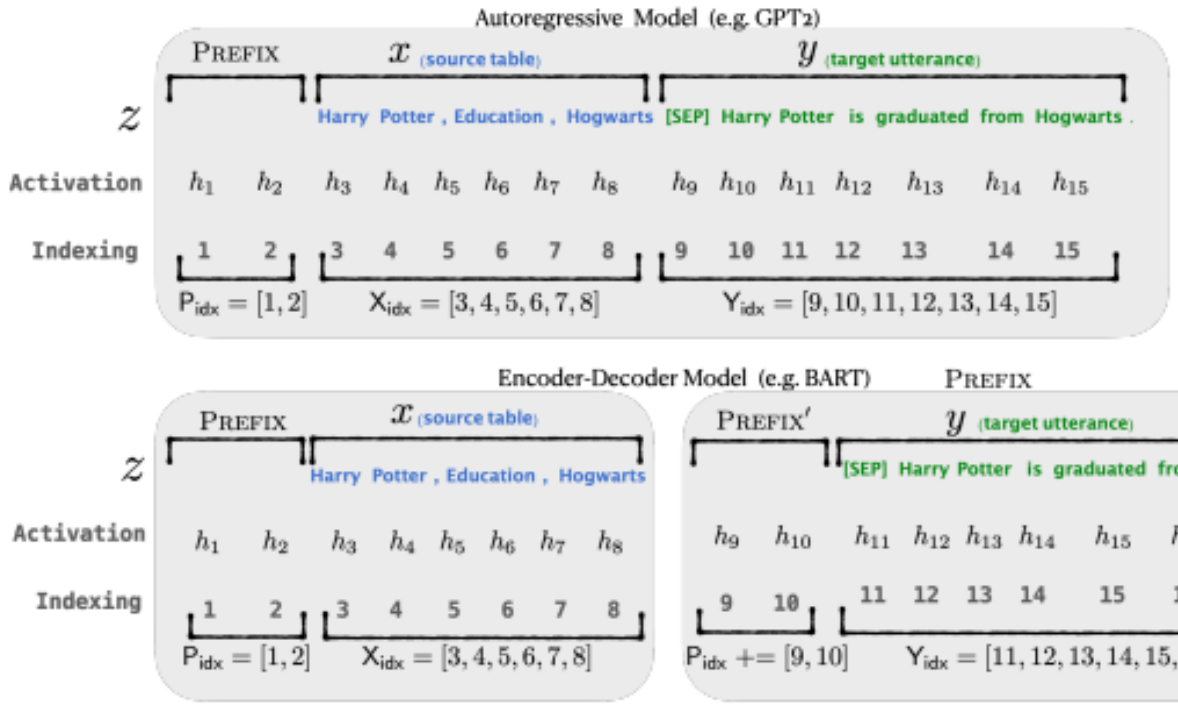
shape of X' = (n+m) x d

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

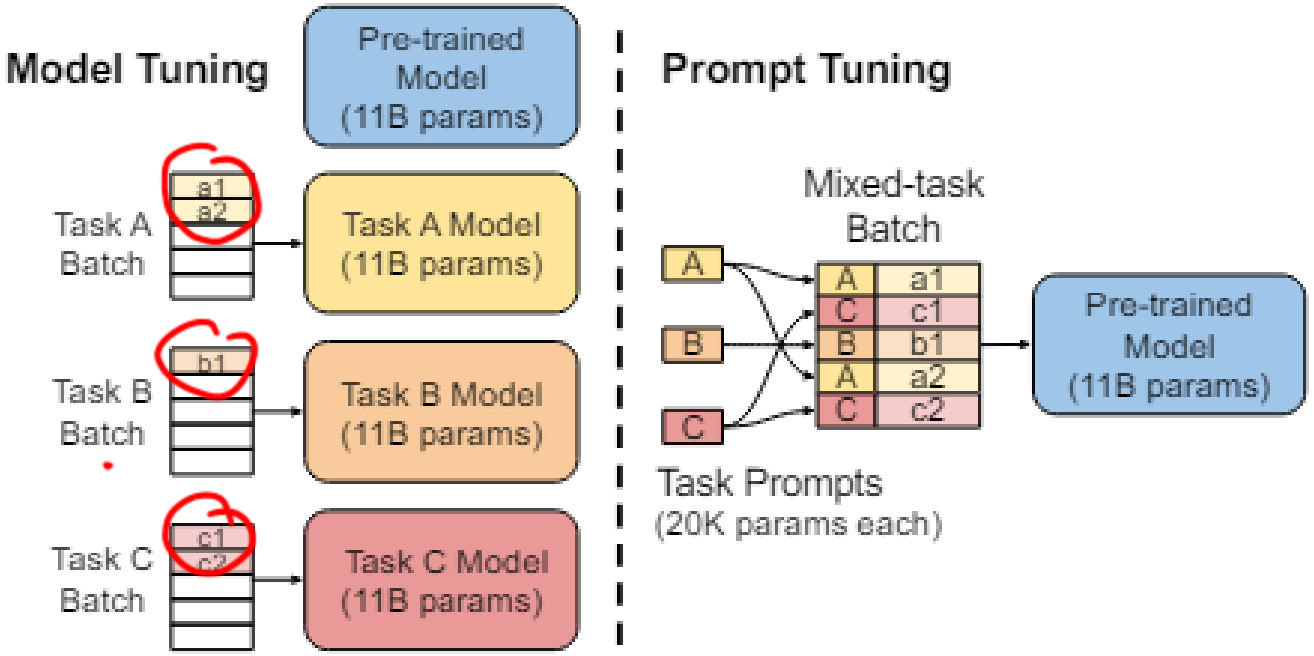
Advantages:

Multi-task input in a single batch (Training)

Very few parameters (< 0.1%)

Disadvantages:

Less effective than prefix-tuning in low-data regimes



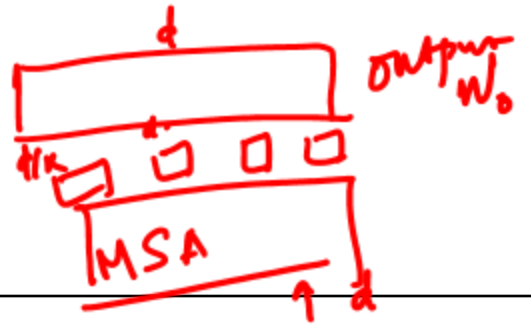
Selective Methods - Structured and Unstructured

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 - BitFit (Zaken et al.)

$BERT_{base}$ model

Which bias terms are most influential?

- All query vectors and 2nd MLP layer
- $b^{(l)}_q$ and b_{m2} rival performance of FFT
- $b^{(l)}_k$ has zero effect on output

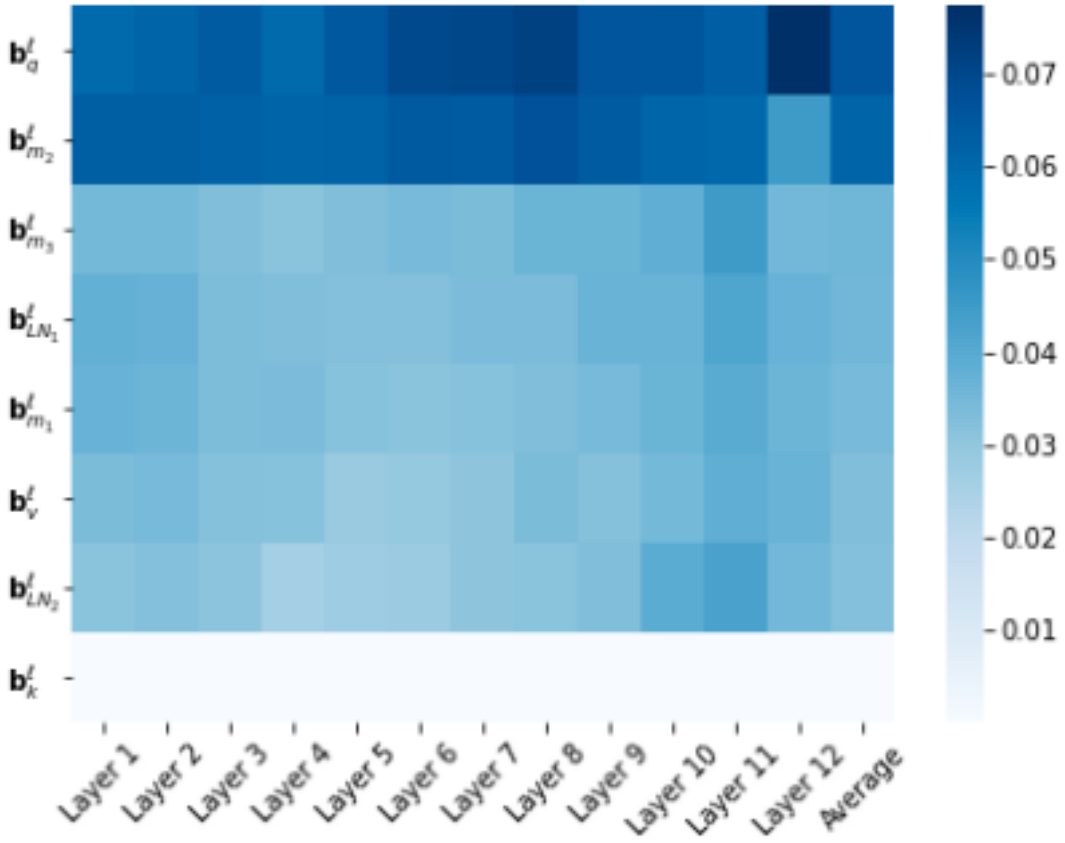


Figure 1: Change in bias components (RTE task).

Selective Methods - FISH Mask (Sung et al.)

Idea:

- Use Fisher Information to select which weights to update

One way to measure a parameter's importance, is to consider what extend the parameter will change the model's output.

We know that the output distribution is given by - $p_\theta(y|x)$

This can be computed using the KL-Divergence as: $D_{KL}(p_\theta(y|x)||p_{\theta+\delta}(y|x))$

Where δ is a small perturbation

It can be shown that as $\delta \to 0$ this is equivalent to the **Fisher information** matrix. Hence Fisher information matrix is widely used in modern ML as a measure of parameter importance

Selective Methods – Fisher Importance Score

Fisher importance for a parameter θ can be calculated as

$$F_\theta = \mathbb{E}_{x\sim p(x)}[\mathbb{E}_{y\sim p_\theta(y|x)}\nabla_\theta \log p_\theta(y|x) \nabla_\theta \log p_\theta(y|x)^{\mathrm{T}}],$$

Due to expectation over the entire input data distribution $p(x)$, actual Fisher importance cannot be calculated. We resort to approximated (or empirical) Fisher calculated on a subsample of data, defined as

$$\hat{F}_\theta = \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}_{y\sim p_\theta(y|x_i)}\Big[\nabla_\theta \log p_\theta(y|x_i) \odot \nabla_\theta \log p_\theta(y|x_i)\Big],$$

Here, $\nabla_\theta \log p_\theta(y|x_i)$ can be calculated by using the gradient of the loss function (loglikelihood) w.r.t the parameter.

Selective Methods - FISH Mask (Sung et al.)

Steps

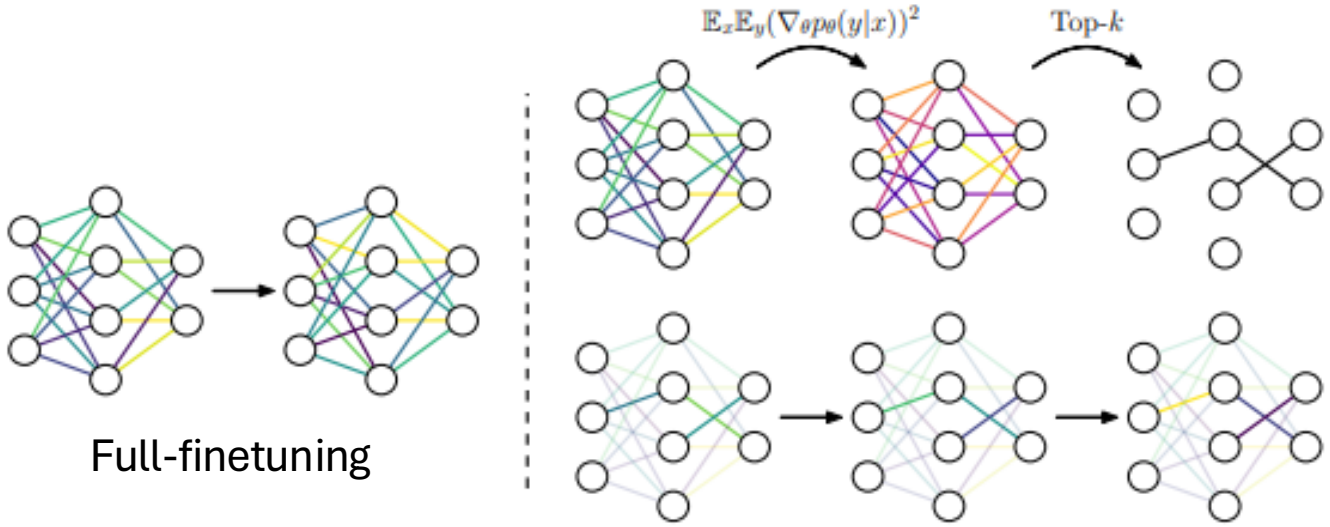
1. Compute Fisher score per weight matrix
2. Select top-K% as trainable
3. Keep rest frozen

Advantages

- Mask is determined initially using sample of the dataset and then kept fixed
- Fixed mask is very efficient during distributed training

Disadvantages

- Requires extra computation for Fisher estimation



Full-finetuning

FISH Masking

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

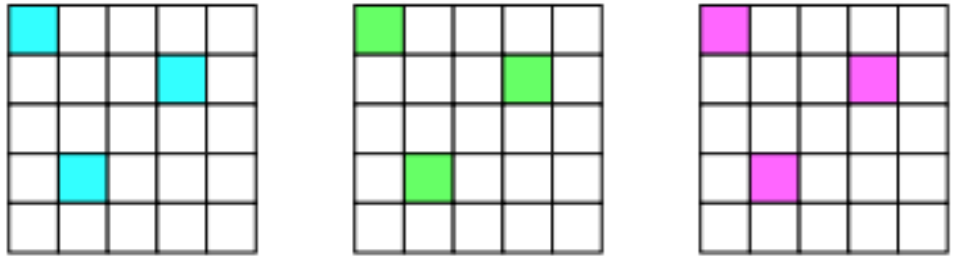
Advantages:
- Fixed masks
- Mask generation based on only magnitude (less computation cost)

Intuition
- 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.

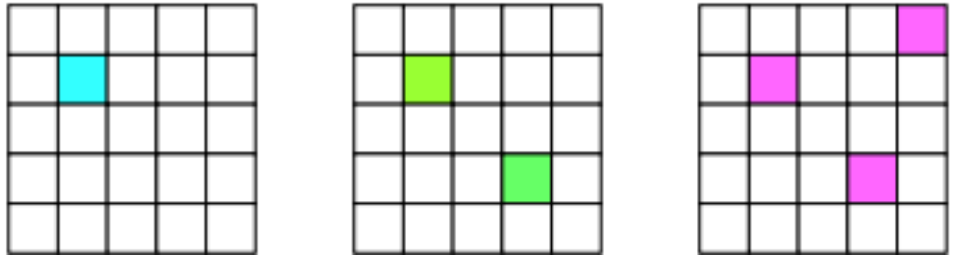
Selective Methods – ID3 (Agarwal et al.)

Idea:

- Incremental selection of parameters instead of selecting all parameters in beginning
- Dynamic gradient & magnitude-based heuristic for selection of parameters in each optimizer step



Same subset of parameters trained in all steps



Incrementally select new parameters in each step

Given parameters θ^i and gradients ∇_{θ^i}

We define parameter importance function (also referred to as heuristic function) as:

$$\mathcal{H}(\theta^i) = \frac{|\nabla_{\theta^i}|}{(|\theta^i| + \epsilon)^{exp}}$$

Where ε and *exp* are hyperparameters to control smoothing of function and effect of parameter magnitude on final importance

Selective Methods – ID3 (Agarwal et al.)

At each optimizer step:

1. Compute importance score for all tensor parameters
2. Select top K/T parameters
3. Add these parameters to the set of parameters that need to be updated

Advantages

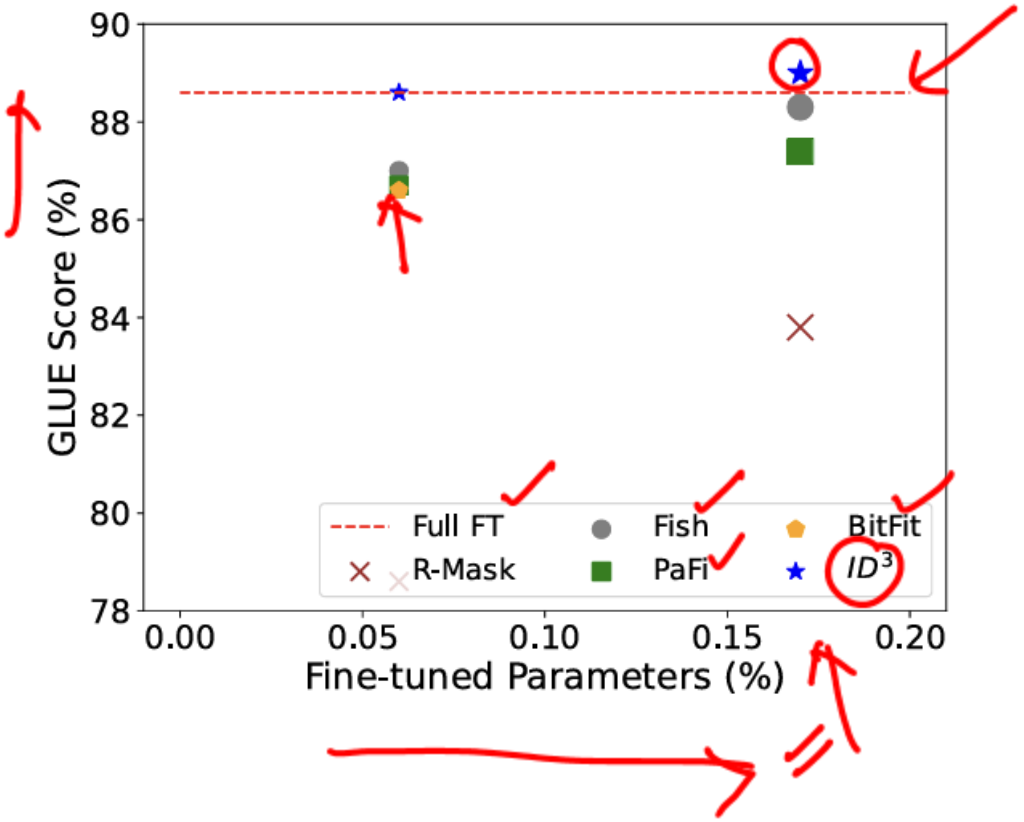
- Incremental selection captures change in parameter importance throughout training process
- Produces progressively improved models across increasing budget levels, allowing users to balance budget and performance effectively
- Compatible with both additive (Adapters) and reparameterization (LoRA) methods

Disadvantages

- Increased computation in optimizer step for calculating parameter importance

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	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	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	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	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 – LoRA (Hu et al.)

Idea:
Insert low-rank matrices into the model's weight update formula

Mechanism:
- Freeze the pretrained model weights
- Inject trainable matrices into attention (e.g., query/key/value) layers and MLP
- Only train A and B (much smaller than W)

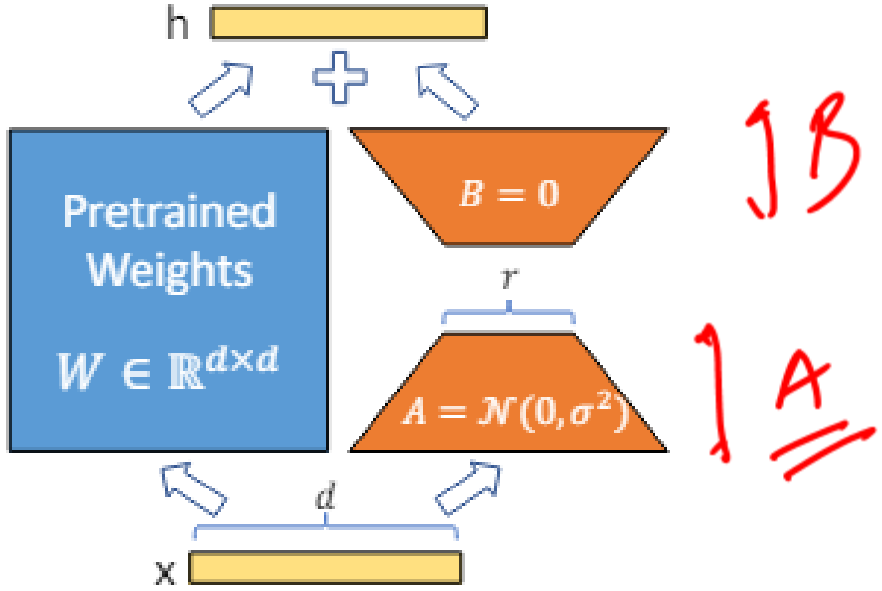


Figure 1: Our reparametrization. We only train A and B.

Re-parameterization Methods – LoRA (Hu et al.)

For a pretrained matrix $W_o \in \mathbb{R}^{d \times k}$, the gradient-based update Δ W is represented as:

$$\Delta W = BA$$
$$\Delta W \in \mathbb{R}^{d \times k}$$
$$B \in \mathbb{R}^{d \times r}$$
$$A \in \mathbb{R}^{r \times k}$$
$$r << min(d, k)$$

During inference time, $h = W_o x$ becomes $h = W_o x + \Delta W x = W_o x + BAx$

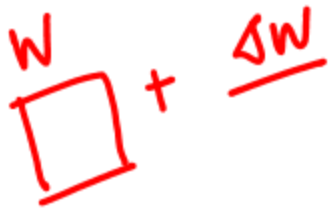
Re-parameterization Methods – LoRA (Hu et al.)

Advantages

- Single pre-trained model can be used to build multiple small LoRA modules for different tasks
- We can freeze the PLM and efficiently switch tasks by replacing the matrices A and B.
- LoRA is orthogonal to other prior PEFT methods and often can be combined
- Simple design with no computation heavy operations
- Compatible with very large models (e.g., GPT, BERT, LLaMA, Qwen, etc)

Limitations:

- Requires careful rank selection
- May need tuning of learning rate and regularization
- Sensitive to hyper-parameters



Re-parameterization Methods - AdaLoRA (Zhang et al.)

Idea

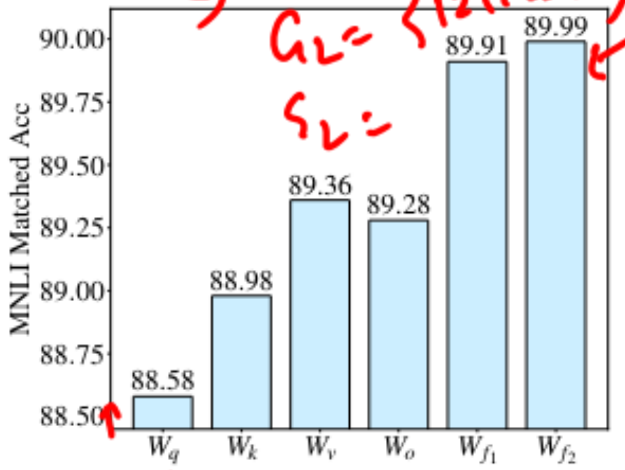
- Extension of LoRA that dynamically allocates low-rank budget across layers

Intuition

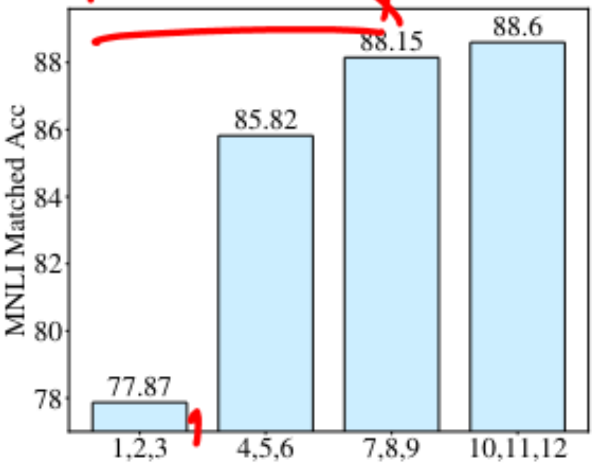
- Not all layers are important
- Adaptive rank leads to better use of limited trainable parameters

Steps

1. Initialize full-rank LoRA
2. Estimate per-layer importance
3. Reduce ranks of less useful layers and increase where needed
4. Continue fine-tuning with the adapted budget



(a) Selected weight matrix



(b) Selected layers

Re-parameterization Methods - AdaLoRA (Zhang et al.)

- The authors reparametrize the weight update (LoRA weights) in a form to approximate the SVD: $\Delta = P\Lambda Q$
- This incremental matrix Δ is divided into triplets where each triplet G_i contains the i-th singular value and the corresponding singular vector
- Each triplet is assigned an importance score
- Triplets with low importance scores are pruned, i.e., their singular values are zeroed, effectively reducing the rank and parameter budget.
- High-importance triplets are retained, allocating more capacity.
- A global schedule starts with a slightly higher parameter budget than the target and gradually reduces the budget during fine-tuning

Advantages

- More expressive and efficient than static LoRA
- Better performance under tight parameter constraints

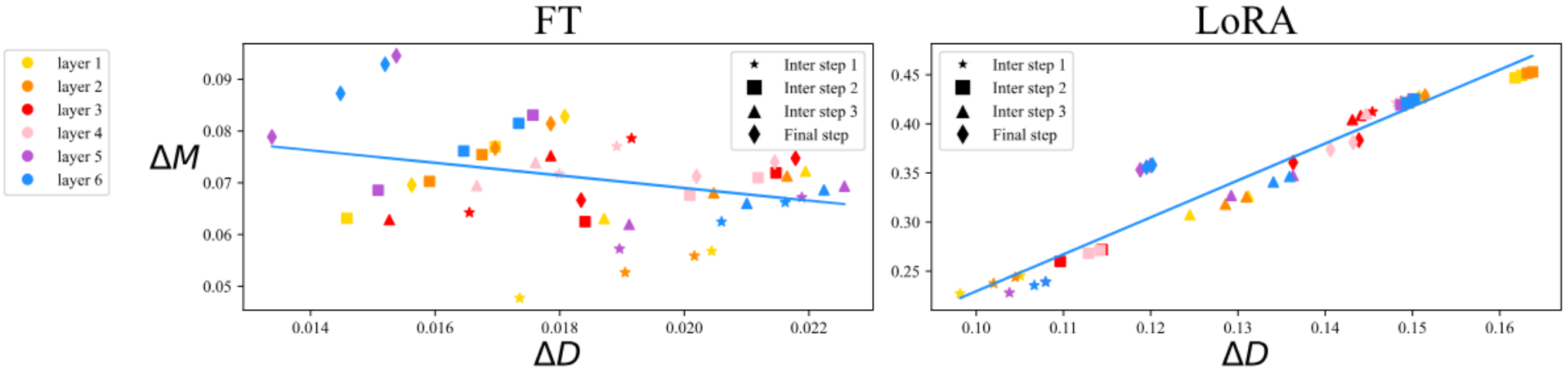
Disadvantages

- Slightly more complex to implement
- Sensitive to ranking heuristics

Re-parameterization Methods - DoRA (Liu et al.)

Idea:

- Decompose weight updates into **direction** and **magnitude** components



Changes in parameters during Full-Finetuning and LoRA fine-tuning

Intuition:

- Comparing FFT and LoRA by looking at the changes in direction and magnitude, shows that LoRA captures a monotonic relationship between magnitude and directional changes, failing to captured more nuanced patterns.

Re-parameterization Methods - DoRA (Liu et al.)

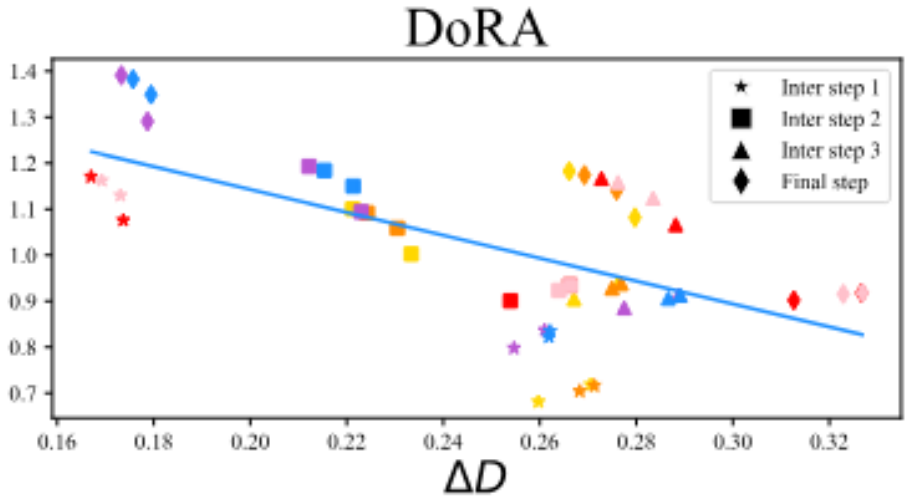
The weight matrix is decomposed as:

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

$$where\ W \in \mathbb{R}^{d\times k}\ is\ the\ weight\ matrix$$

$$m \in \mathbb{R}^{1\times k}\ is\ the\ magnitude\ vector$$

$$and\ V \in \mathbb{R}^{d\times k}\ is\ the\ directional\ matrix$$



Advantages:

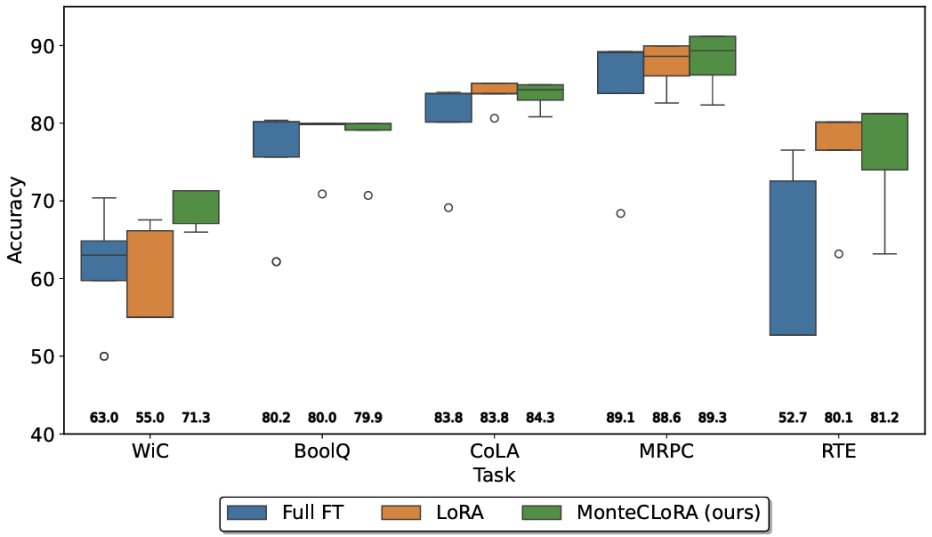
- Improves generalization
- Reduces overfitting in low-data settings
- Lower parameter sensitivity to initialization

Disadvantages:

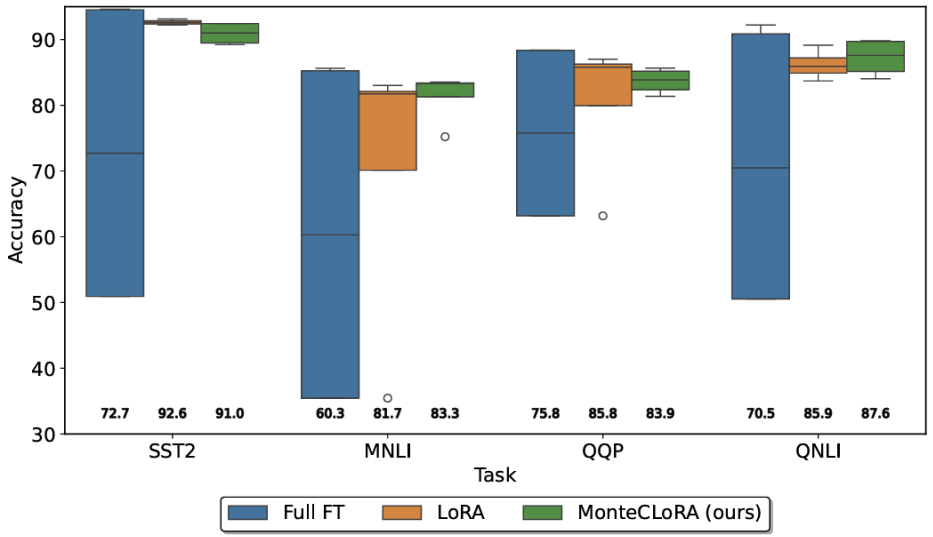
- May underperform when magnitude carries key task signal

Re-parameterization Methods - MonteCLoRA (Sengupta et al.)

LoRA-based re-parameterization methods are highlight sensitive to hyperparameters such as learning rate or batch size.



(a) GLUE Small Tasks

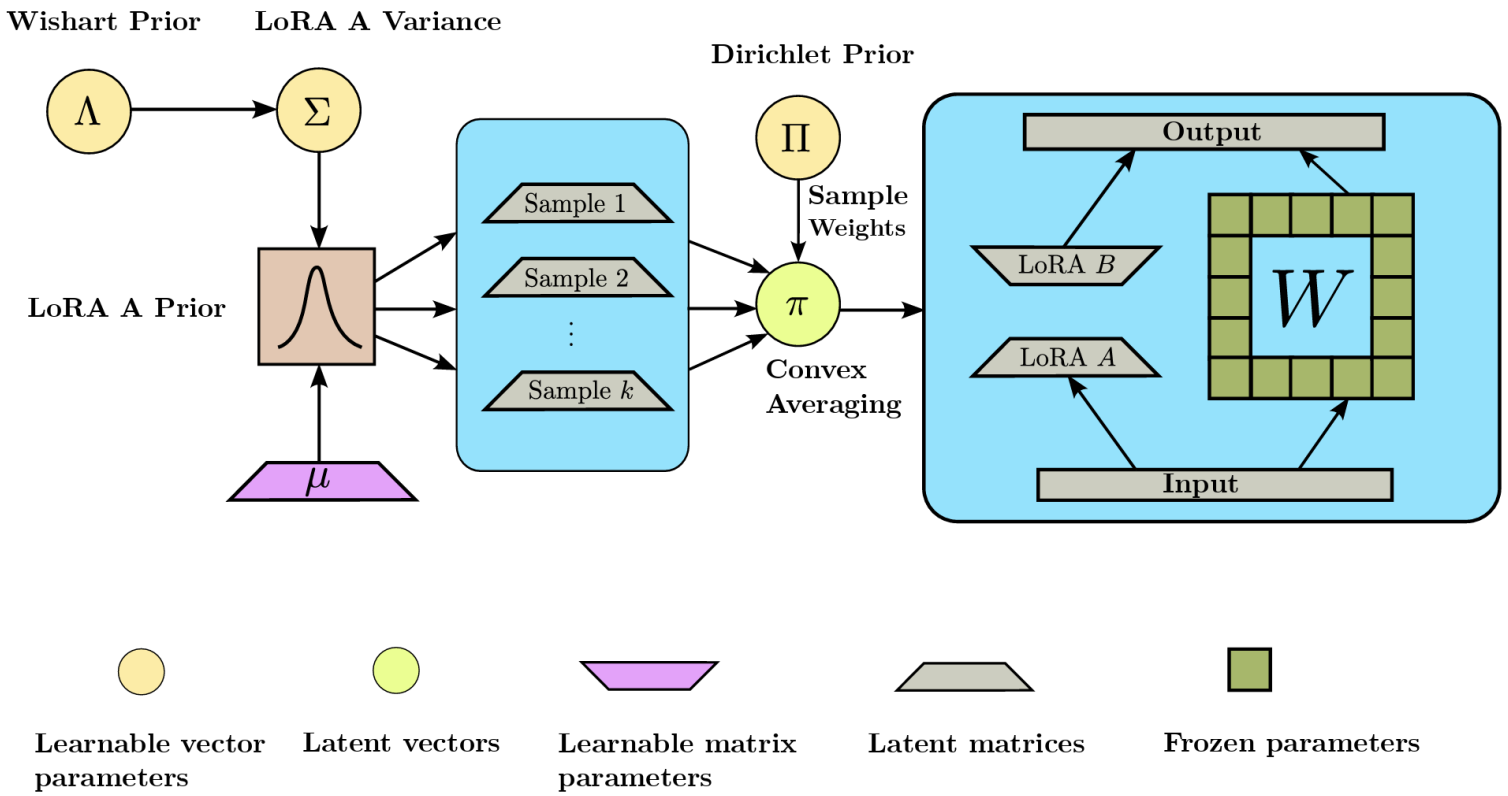


(b) GLUE Large Tasks

Negative outliers for LoRA and full fine-tuning results are noticeable for RoBERTa-base model on GLUE tasks.

Re-parameterization Methods - MonteCLoRA (Sengupta et al.)

Idea: Can we parameterize the low-rank learnable matrices with some probabilistic priors?



- Each LoRA A matrix can be treated as mixture of Gaussian with mean μ (learnable) and variance Σ (learnable)
- Variance is sampled from a Wishart distribution with prior Λ (learnable)
- Mixture probability is sampled from a Dirichlet distribution with prior Π (hyperparameter)

Re-parameterization Methods - MonteCLoRA (Sengupta et al.)

Some theory on why MonteCLoRA works?

- Output generated from MonteCLoRA layer is an unbiased estimator of an output generated from the underlying LoRA layer (therefore no additional expected loss)
- An added Gaussian noise to a matrix parameter θ reduces Lipschitz bound (largest eigenvalue) of the Hessian matrix (square of gradient matrix) – which reduces sensitivity to learning rates
- MonteCLoRA update is equivalent to adding a Gaussian noise to each LoRA parameter

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

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

References

References

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.