



CMPT 413/713: Natural Language Processing

Parameter efficient fine-tuning

Spring 2025
2025-03-17

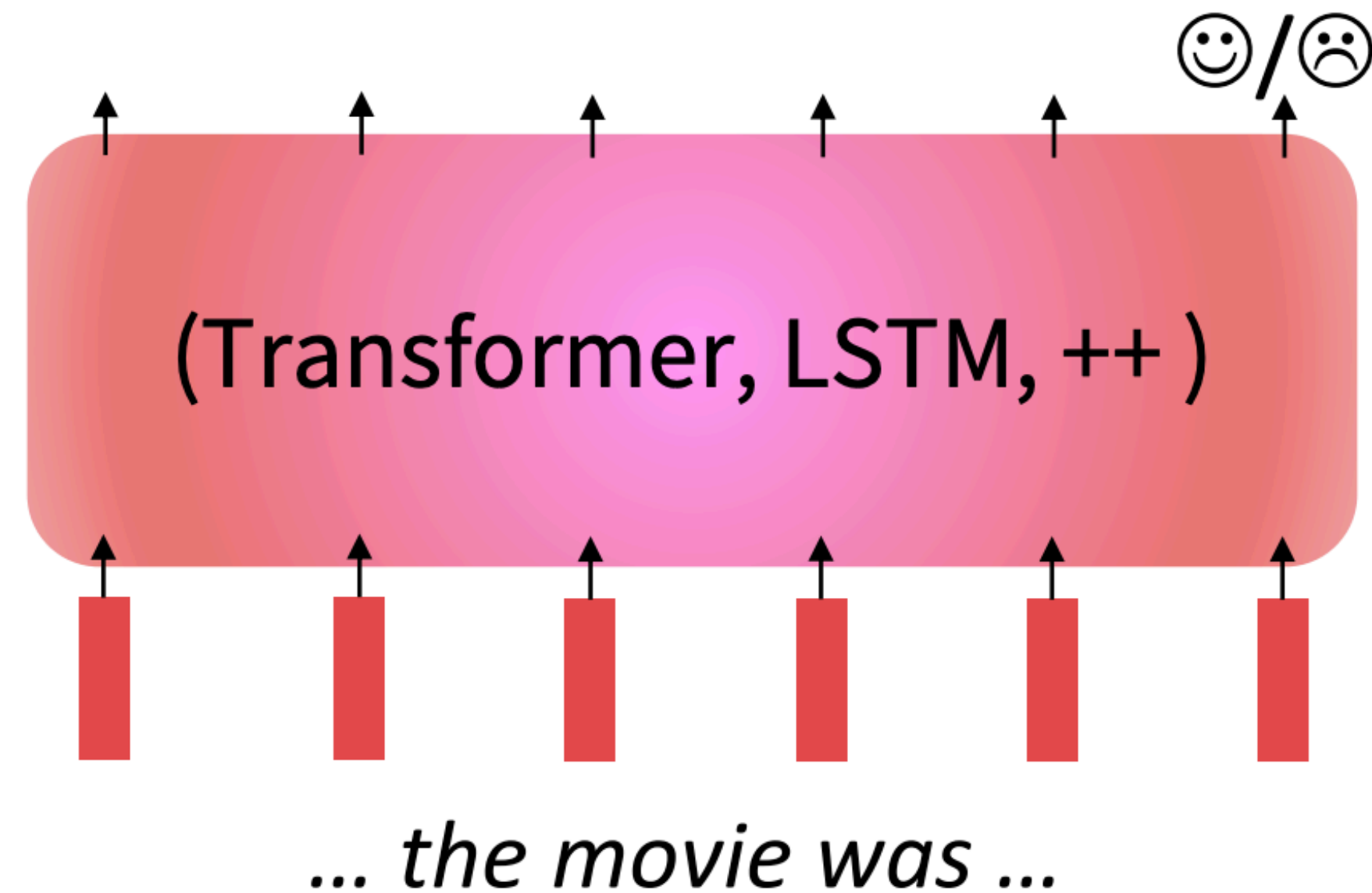
Slides adapted from Anoop Sarkar

Full finetuning vs parameter efficient fine-tuning

- Finetuning every parameter in a pretrained model works well, but is memory-intensive.
- **Lightweight** finetuning methods adapt pretrained models in a constrained way.
- Leads to **less overfitting** and/or **more efficient finetuning and inference**.

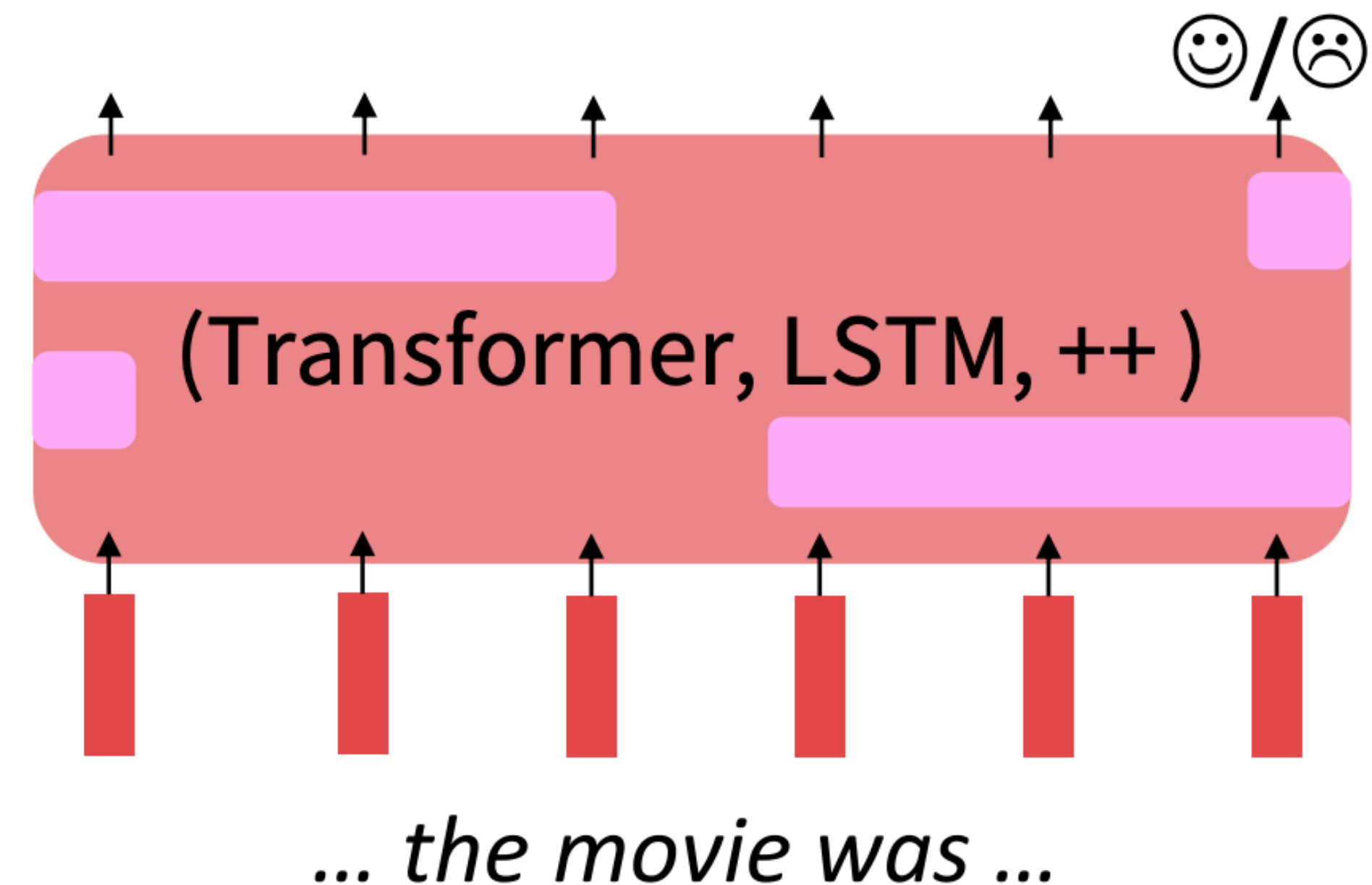
Full Finetuning

Adapt all parameters



Lightweight Finetuning

Train a few existing or new parameters

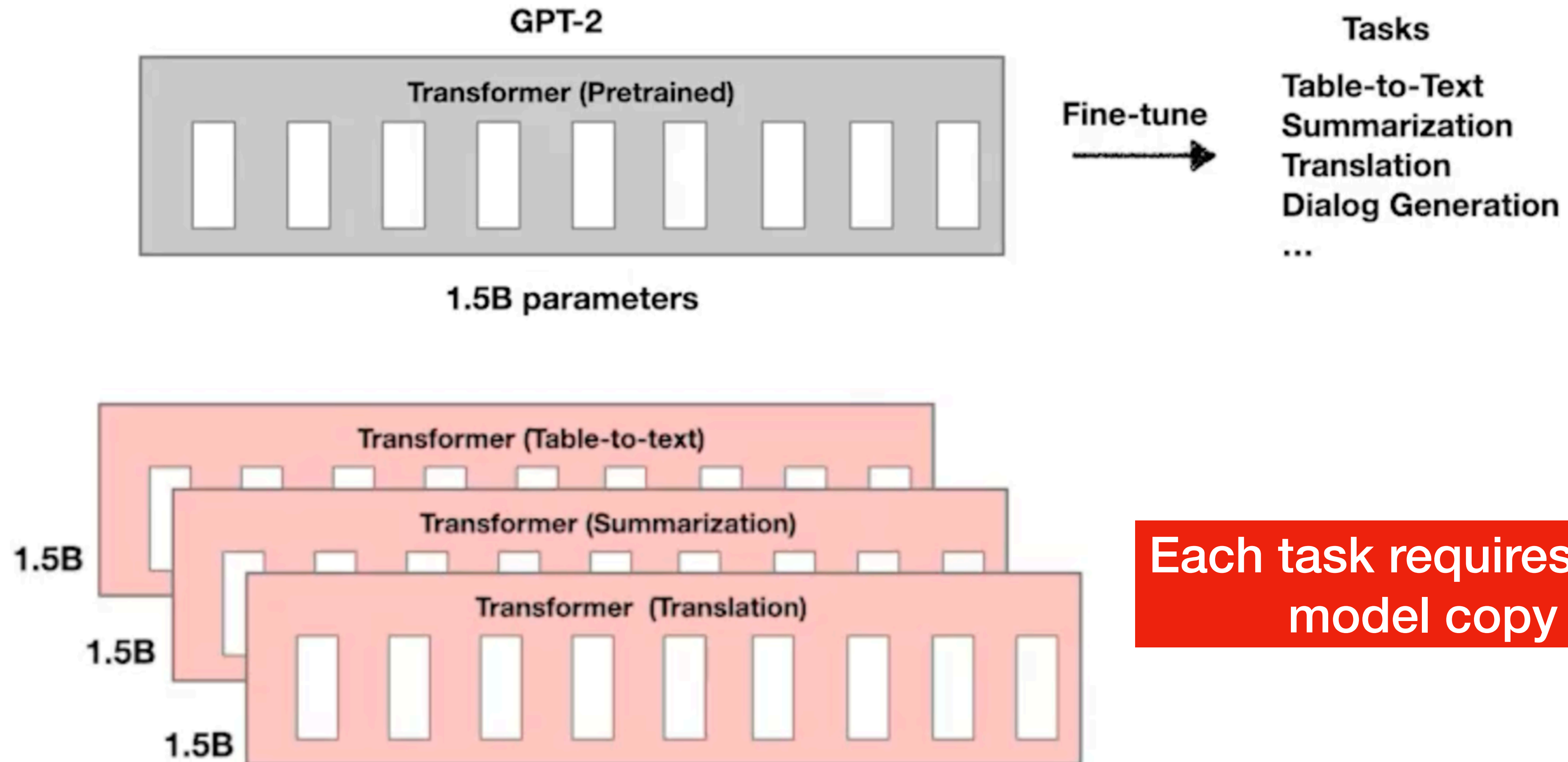


Prefix Tuning

Li and Liang, ACL 2021

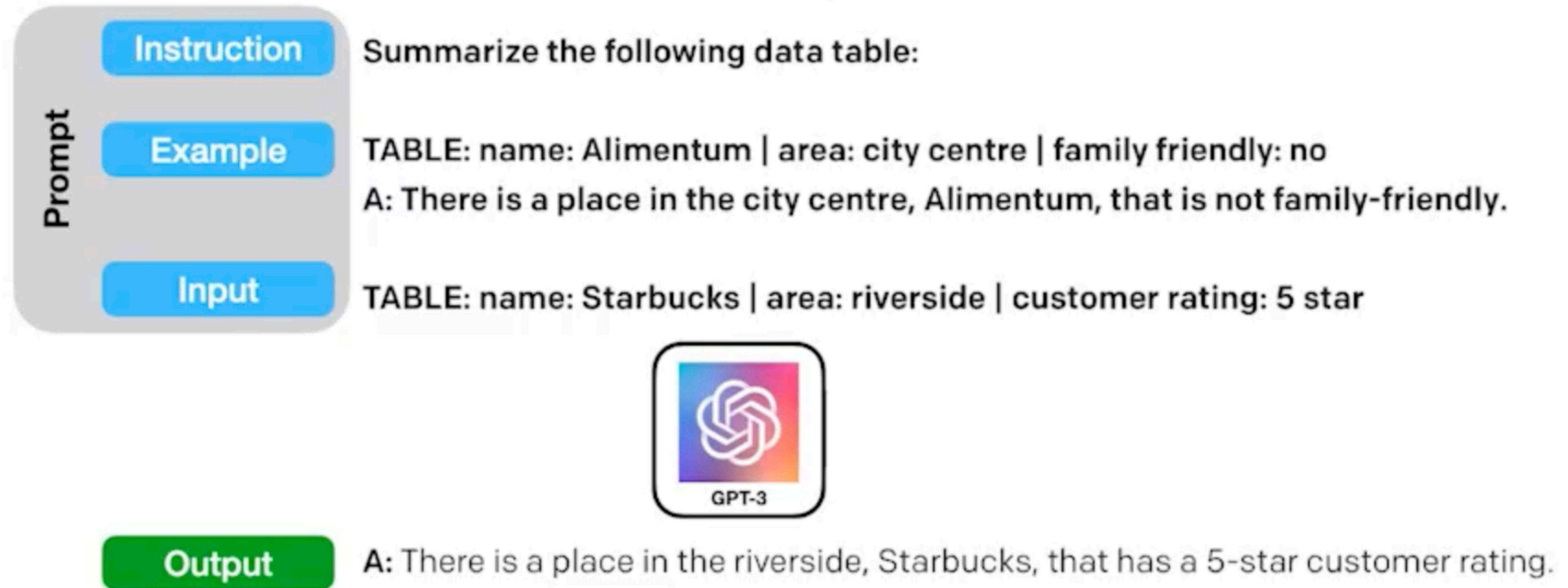
<https://aclanthology.org/2021.acl-long.353>

Why not just use fine-tuning



Each task requires a full model copy

In-context learning using prompts



- No task specific fine-tuning
- Preserves the LM

- Cannot use large training set
- Manual prompts can be suboptimal
- Cannot be used with smaller LMs like GPT-2

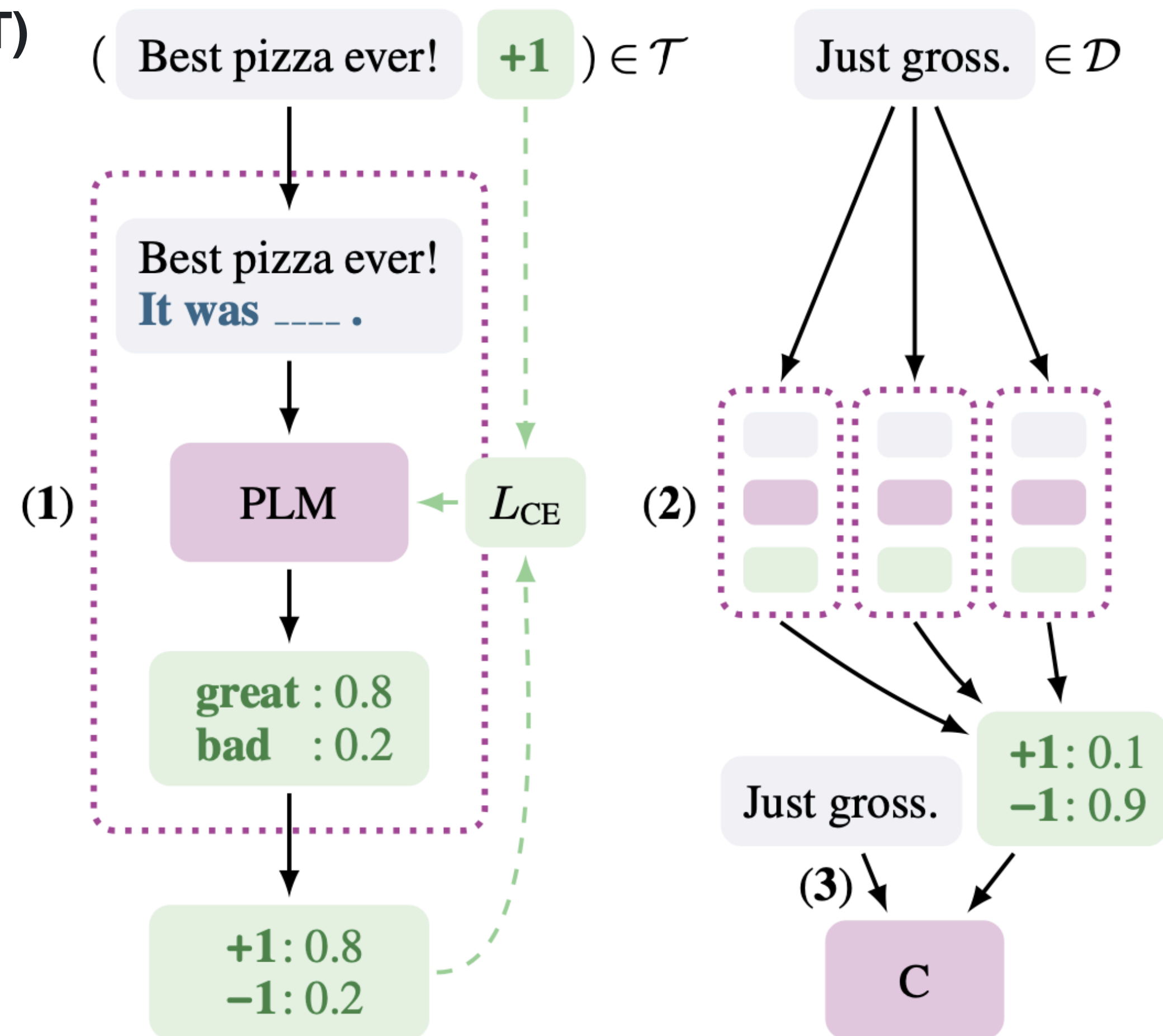
Prompt tuning: enabling smaller LMs

iPet: better prompts for each task improves accuracy for small LMs

Pattern-Exploiting Training (PET)
for sentiment analysis

(1) **Patterns encoding task description** used to convert training examples to cloze questions

Pretrained LM is fine-tuned for each pattern



(2) **Ensemble of trained models** used to annotate unlabeled data

(3) **Classifier trained on resulting soft-labeled dataset**

Prompt tuning: enabling smaller LMs

iPet: better prompts for each task improves accuracy for small LMs

iPET: iterative PET - iteratively repeat to generate larger dataset

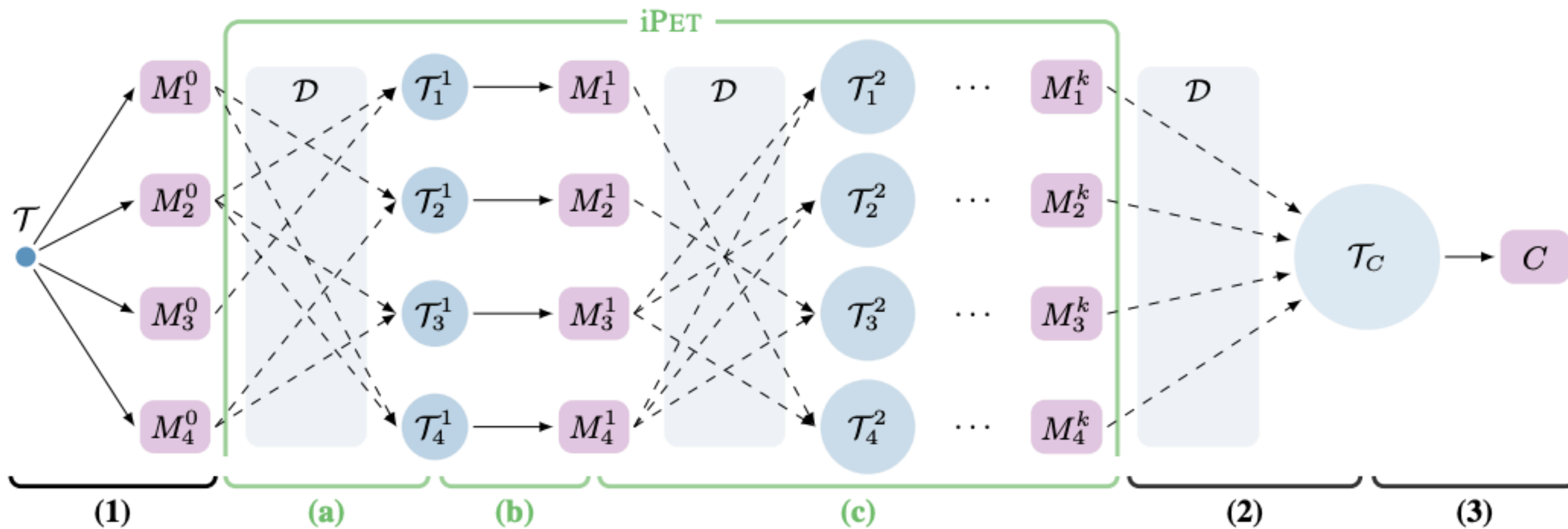


Figure 2: Schematic representation of PET (1-3) and iPET (a-c). (1) The initial training set is used to finetune an ensemble of PLMs. (a) For each model, a random subset of other models generates a new training set by labeling examples from \mathcal{D} . (b) A new set of PET models is trained using the larger, model-specific datasets. (c) The previous two steps are repeated k times, each time increasing the size of the generated training sets by a factor of d . (2) The final set of models is used to create a soft-labeled dataset \mathcal{T}_C . (3) A classifier C is trained on this dataset.

Prompt tuning: enabling smaller LMs

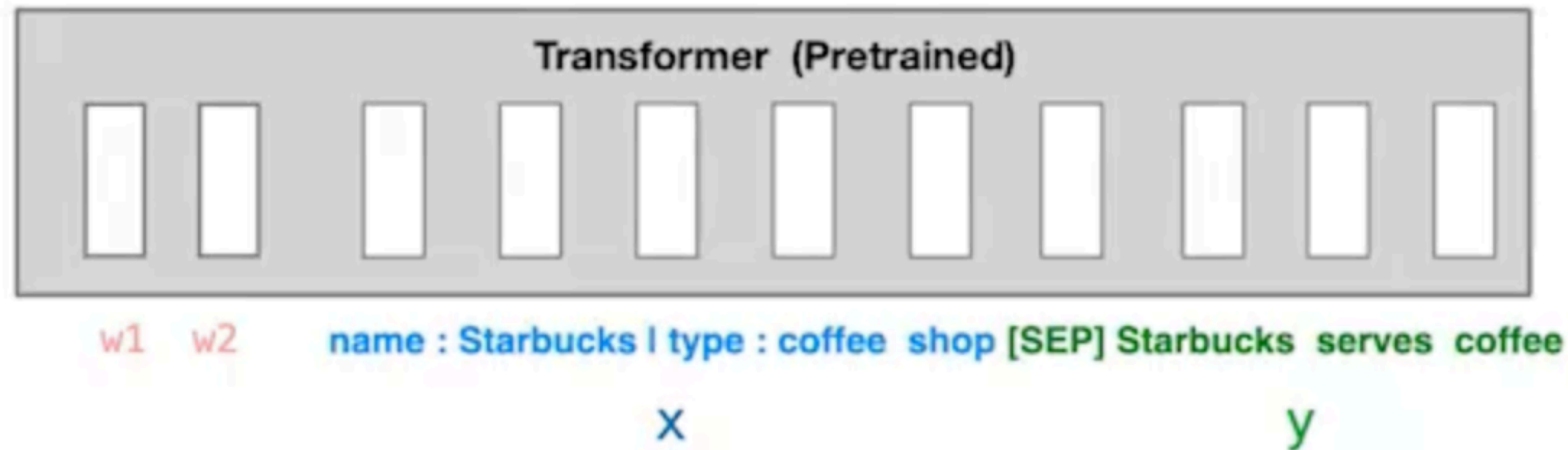
Line	Examples	Method	Yelp	AG's	Yahoo	MNLI (m/mm)
1	$ \mathcal{T} = 0$	unsupervised (avg)	33.8 \pm 9.6	69.5 \pm 7.2	44.0 \pm 9.1	39.1 \pm 4.3 / 39.8 \pm 5.1
2		unsupervised (max)	40.8 \pm 0.0	79.4 \pm 0.0	56.4 \pm 0.0	43.8 \pm 0.0 / 45.0 \pm 0.0
3		iPET	56.7 \pm 0.2	87.5 \pm 0.1	70.7 \pm 0.1	53.6 \pm 0.1 / 54.2 \pm 0.1
4	$ \mathcal{T} = 10$	supervised	21.1 \pm 1.6	25.0 \pm 0.1	10.1 \pm 0.1	34.2 \pm 2.1 / 34.1 \pm 2.0
5		PET	52.9 \pm 0.1	87.5 \pm 0.0	63.8 \pm 0.2	41.8 \pm 0.1 / 41.5 \pm 0.2
6		iPET	57.6 \pm 0.0	89.3 \pm 0.1	70.7 \pm 0.1	43.2 \pm 0.0 / 45.7 \pm 0.1
7	$ \mathcal{T} = 50$	supervised	44.8 \pm 2.7	82.1 \pm 2.5	52.5 \pm 3.1	45.6 \pm 1.8 / 47.6 \pm 2.4
8		PET	60.0 \pm 0.1	86.3 \pm 0.0	66.2 \pm 0.1	63.9 \pm 0.0 / 64.2 \pm 0.0
9		iPET	60.7 \pm 0.1	88.4 \pm 0.1	69.7 \pm 0.0	67.4 \pm 0.3 / 68.3 \pm 0.3
10	$ \mathcal{T} = 100$	supervised	53.0 \pm 3.1	86.0 \pm 0.7	62.9 \pm 0.9	47.9 \pm 2.8 / 51.2 \pm 2.6
11		PET	61.9 \pm 0.0	88.3 \pm 0.1	69.2 \pm 0.0	74.7 \pm 0.3 / 75.9 \pm 0.4
12		iPET	62.9 \pm 0.0	89.6 \pm 0.1	71.2 \pm 0.1	78.4 \pm 0.7 / 78.6 \pm 0.5
13	$ \mathcal{T} = 1000$	supervised	63.0 \pm 0.5	86.9 \pm 0.4	70.5 \pm 0.3	73.1 \pm 0.2 / 74.8 \pm 0.3
14		PET	64.8 \pm 0.1	86.9 \pm 0.2	72.7 \pm 0.0	85.3 \pm 0.2 / 85.5 \pm 0.4

Table 1: Average accuracy and standard deviation for RoBERTa (large) on Yelp, AG’s News, Yahoo and MNLI (m:matched/mm:mismatched) for five training set sizes $|\mathcal{T}|$.

Prefix Tuning

Intuition

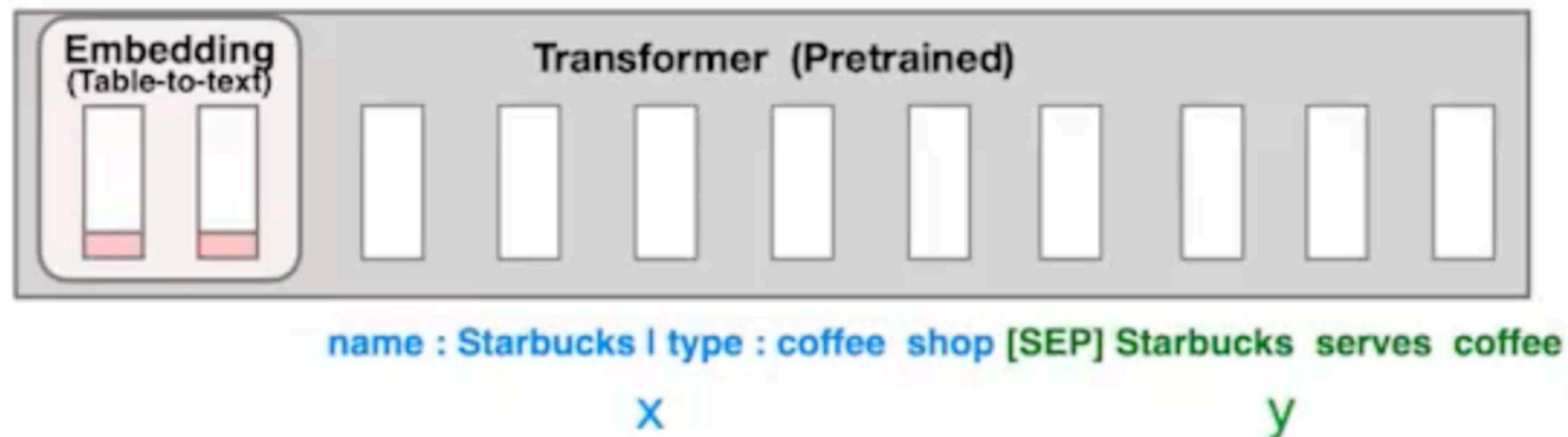
- Learn a good instruction that can steer the LM to produce the right output
- Optimize finding actual words
- Involves discrete optimization which is challenging and not expressive



Prefix Tuning

Intuition

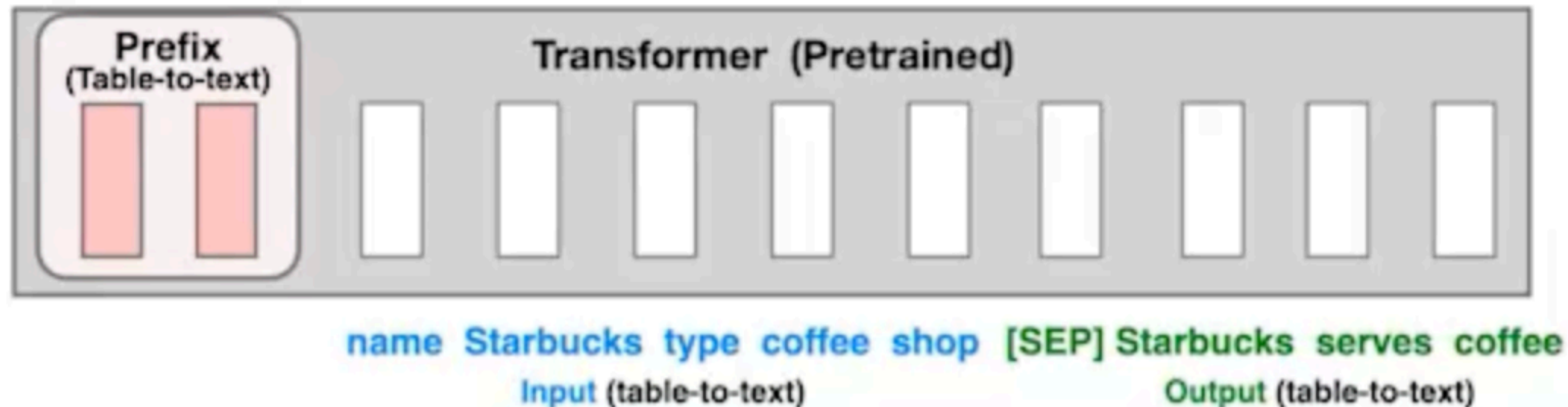
- Optimize the instruction as **continuous word embeddings**
- More expressive
- Limits the scope of the prompt to a input embeddings



Prefix Tuning

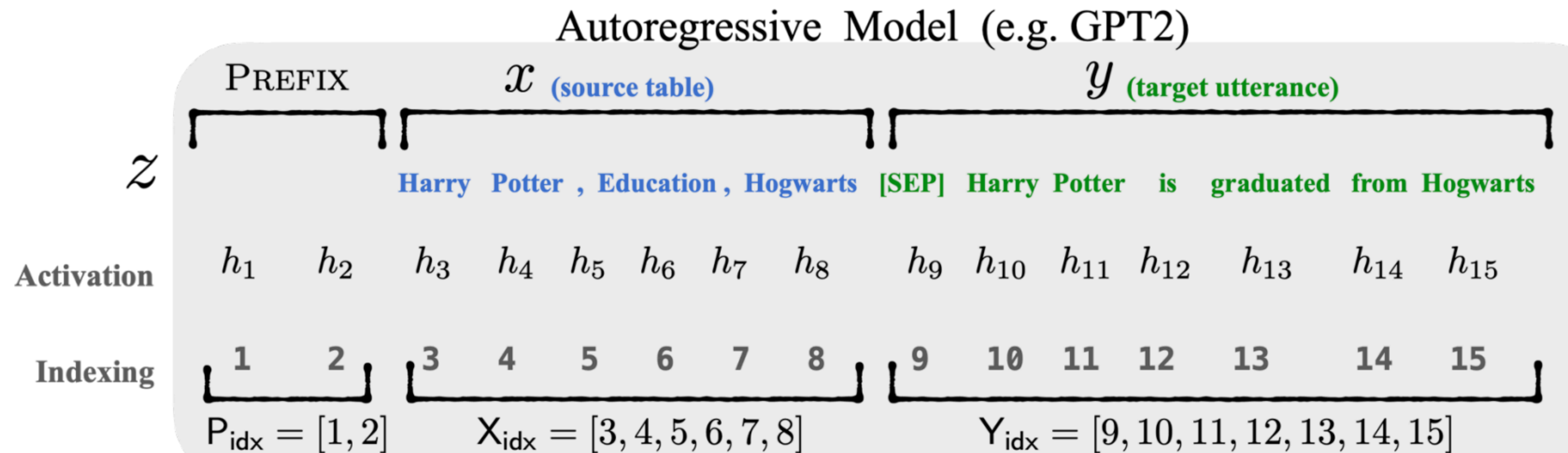
Intuition

- Optimize the instruction as prefix activation for all layers in the instruction
- Very expressive
- [All the layers of the prefix can be tuned](#) to create the most expressive prompt



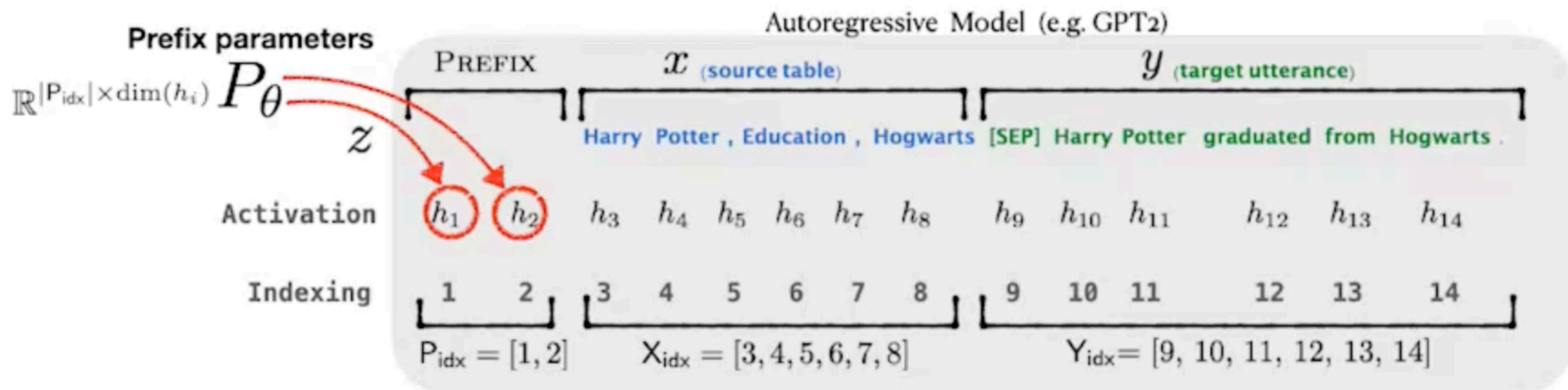
Prefix Tuning

Autoregressive Modelling



Prefix Tuning

$$h_i = \begin{cases} P_\theta[i, :], & \text{if } i \in P_{\text{idx}}, \\ \text{LM}_\phi(z_i, h_{<i}), & \text{otherwise.} \end{cases}$$



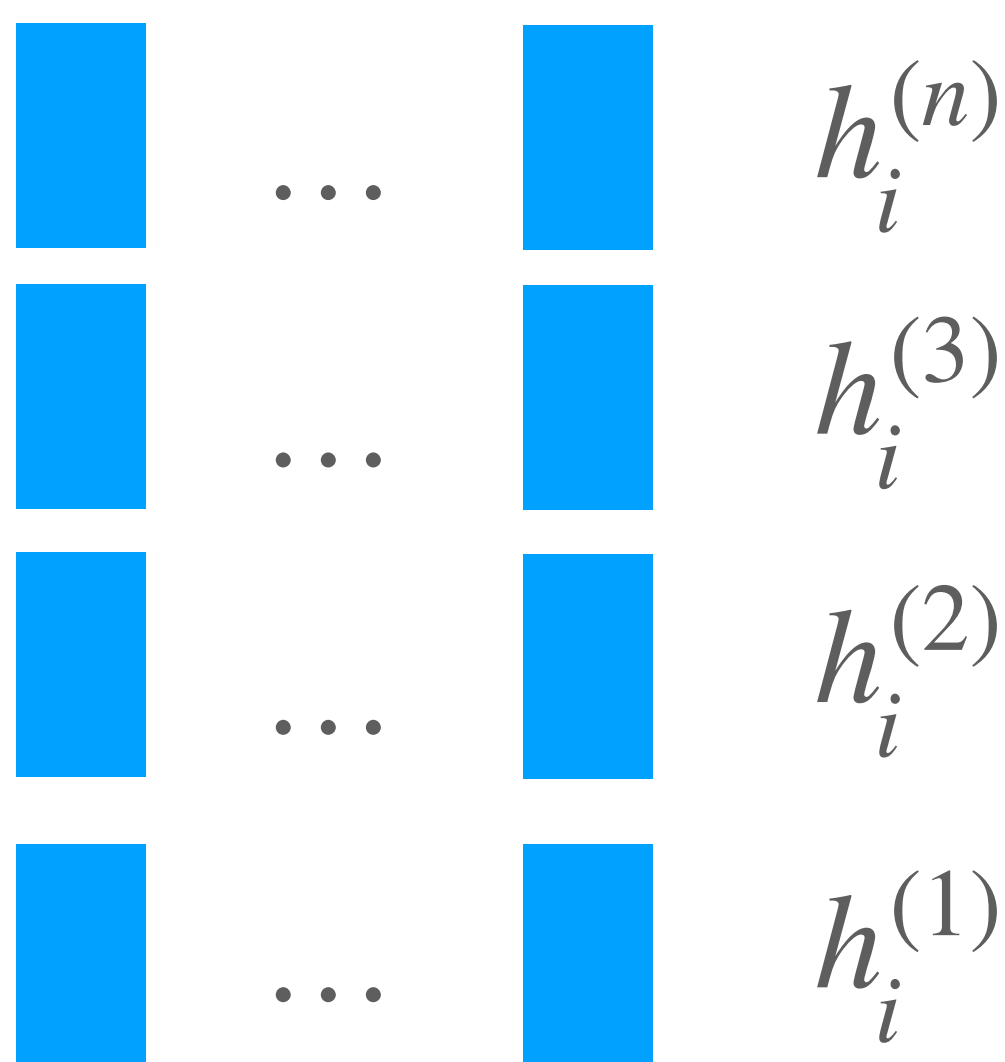
$$\max_{\theta} \log p_{\phi, \theta}(y \mid x) = \sum_{i \in Y_{\text{idx}}} \log p_{\phi, \theta}(z_i \mid h_{<i})$$

freeze LM parameters ϕ
update prefix parameters θ

Prefix Re-parametrization

$$h_i = \begin{cases} P_\theta[i, :], & \text{if } i \in P_{idx}, \\ \text{LM}_\phi(z_i, h_{<i}), & \text{otherwise.} \end{cases}$$

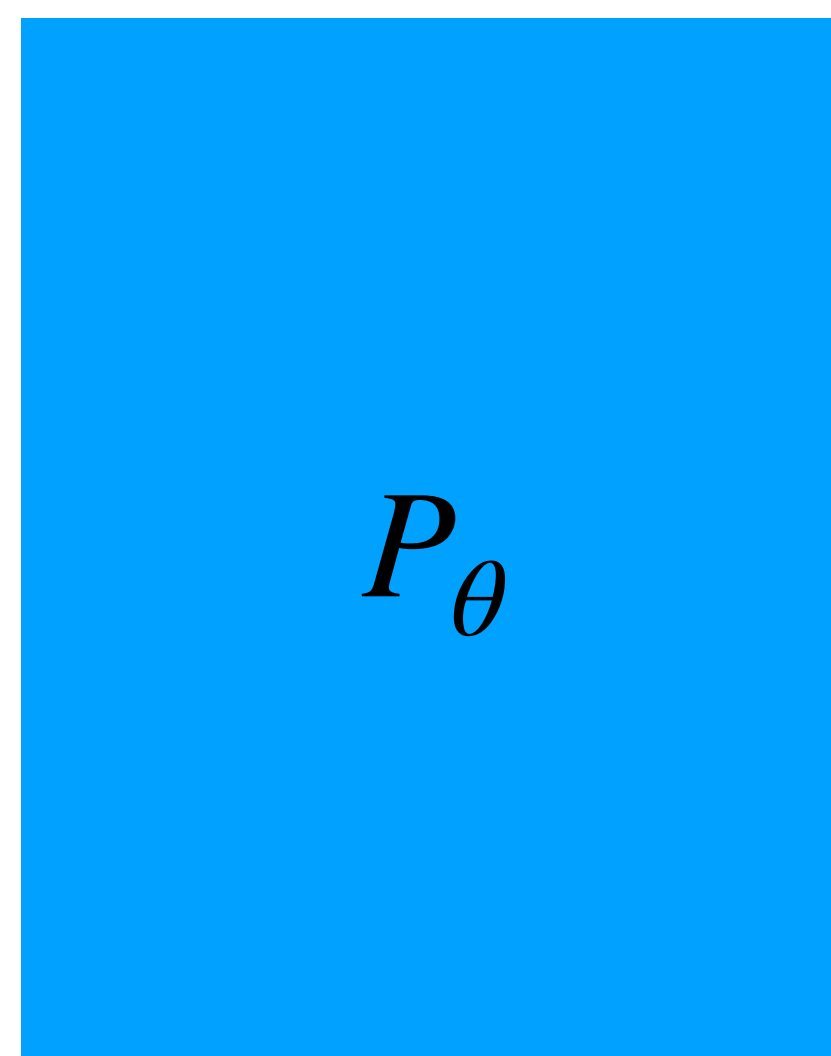
- Directly updating parameters P_θ is unstable



$h_1 \quad \dots \quad h_{|P_{idx}|}$

$P_\theta[i, :]$

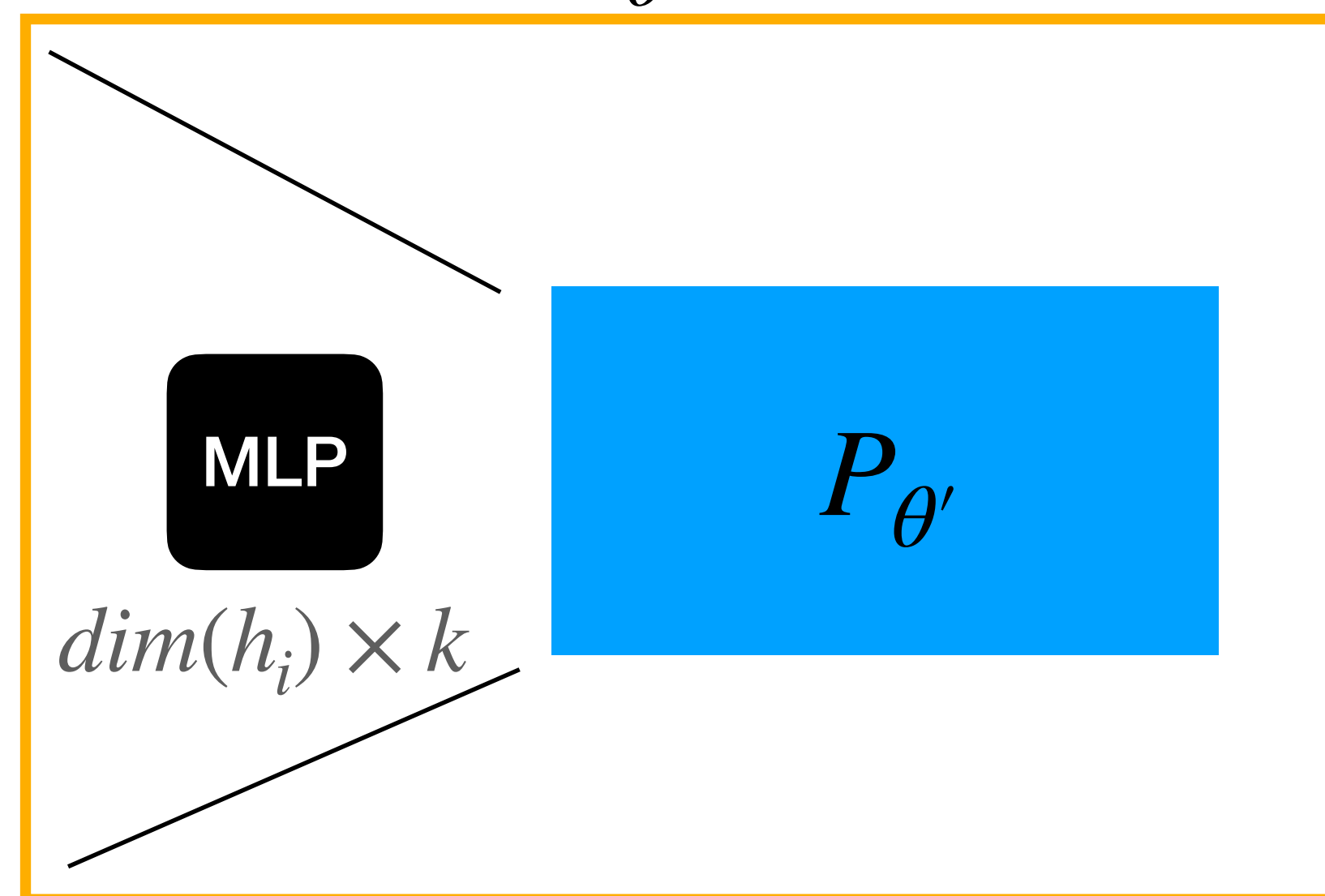
$|P_{idx}| \times \dim(h_i)$



$|P_{idx}| \times \dim(h_i)$

Once training is complete
we store only P_θ (throw
away the MLP)

Train $P_{\theta'}$ and MLP

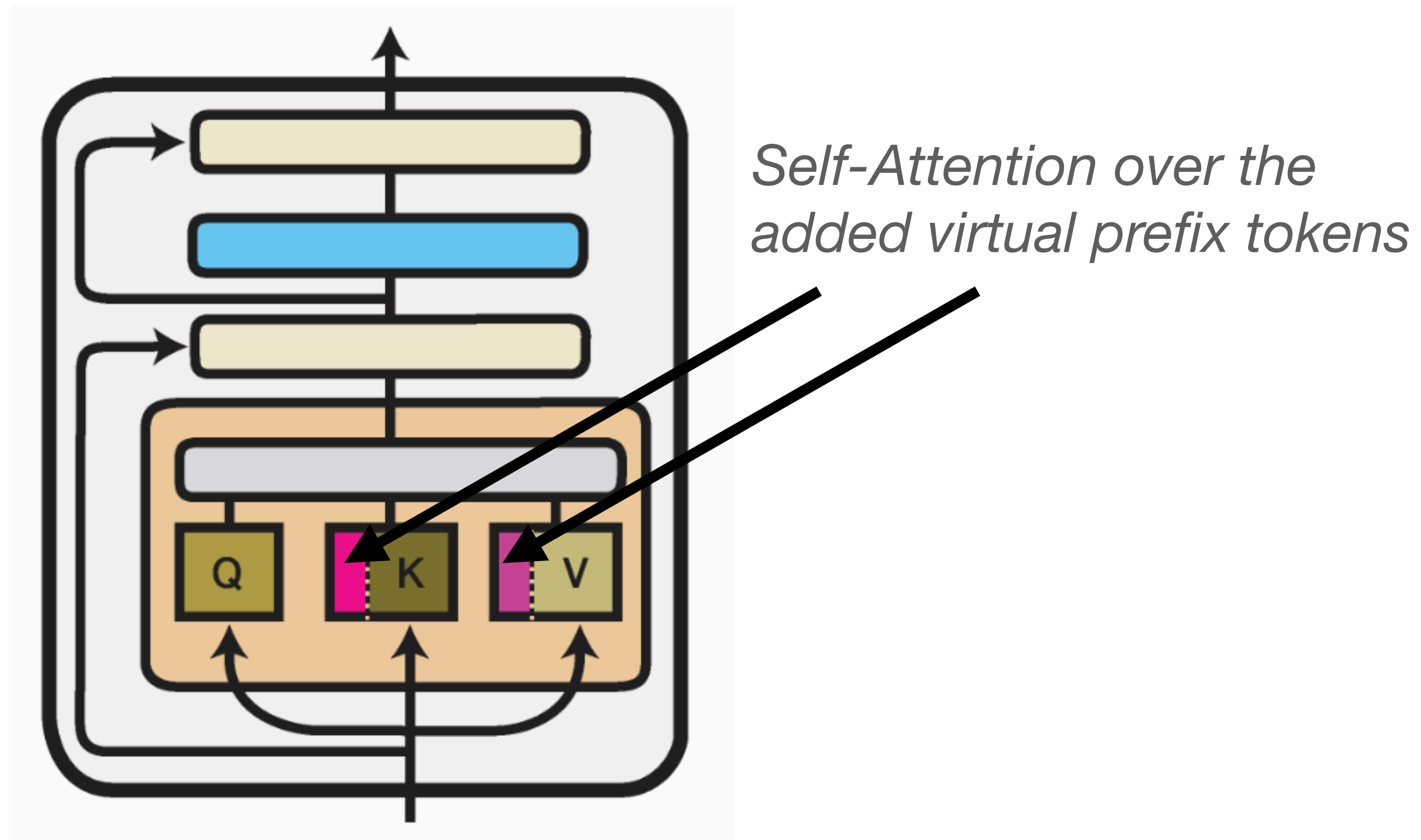


$\dim(h_i) \times k$

$|P_{idx}| \times k$

k is 512 for table-to-text
and 800 for summarization

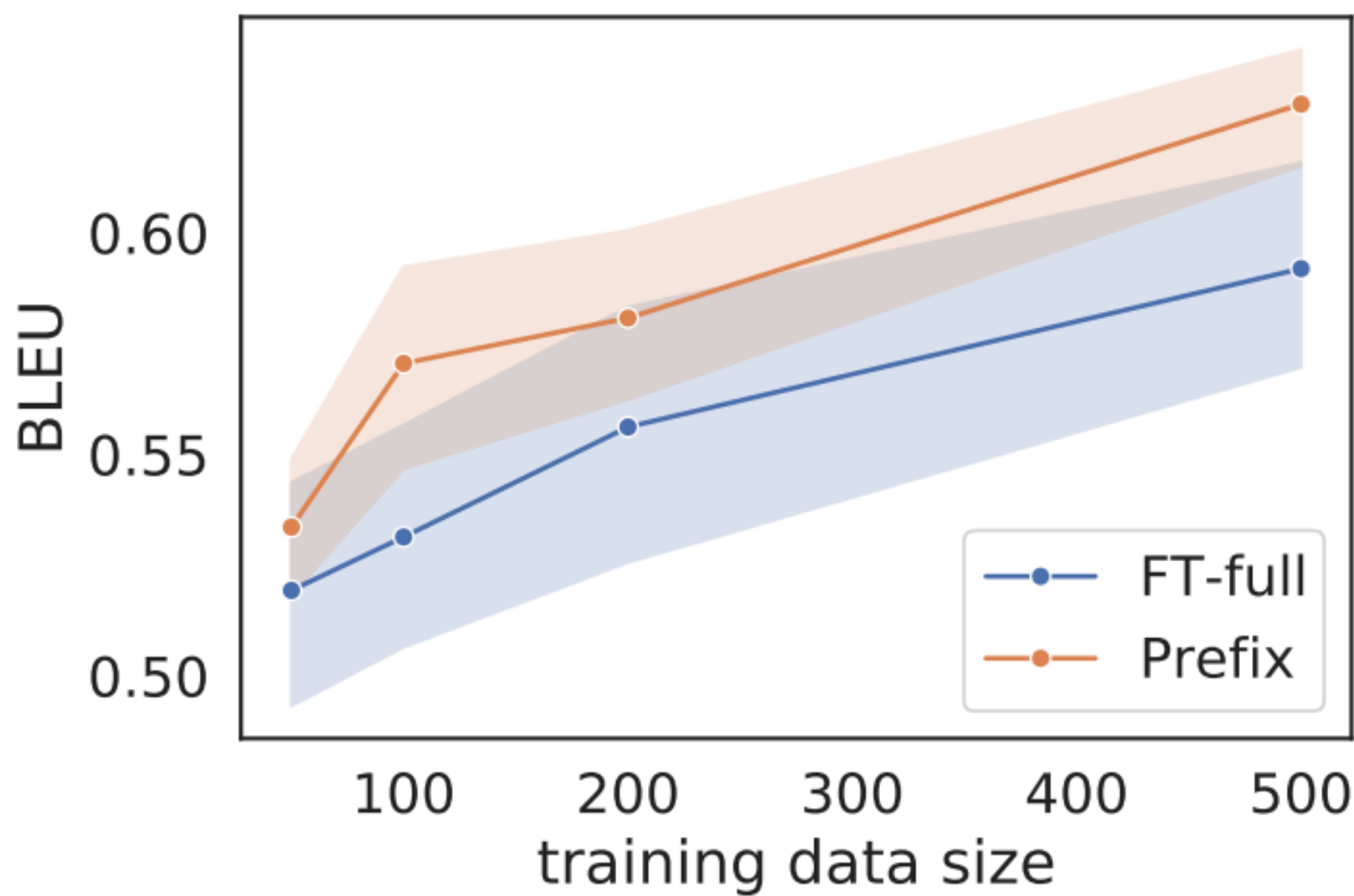
Effect of Prefix Tuning



Prefix Tuning

Vs. Finetuning

- Effective for small amount of training data, requires less parameters than full fine-tuning
- Slightly better (more faithful) outputs than full fine tuning



Source	name : The Eagle type : coffee shop food : Chinese price : cheap customer rating : average area : riverside family friendly : no near : Burger King
Prefix (50)	The Eagle is a cheap Chinese coffee shop located near Burger King.
Prefix (100)	The Eagle is a cheap coffee shop located in the riverside near Burger King. It has average customer ratings.
Prefix (200)	The Eagle is a cheap Chinese coffee shop located in the riverside area near Burger King. It has average customer ratings.
Prefix (500)	The Eagle is a coffee shop that serves Chinese food. It is located in the riverside area near Burger King. It has an average customer rating and is not family friendly.
FT (50)	The Eagle coffee shop is located in the riverside area near Burger King.
FT (100)	The Eagle is a cheap coffee shop near Burger King in the riverside area. It has a low customer rating and is not family friendly.
FT (200)	The Eagle is a cheap Chinese coffee shop with a low customer rating. It is located near Burger King in the riverside area.
FT (500)	The Eagle is a cheap Chinese coffee shop with average customer ratings. It is located in the riverside area near Burger King.

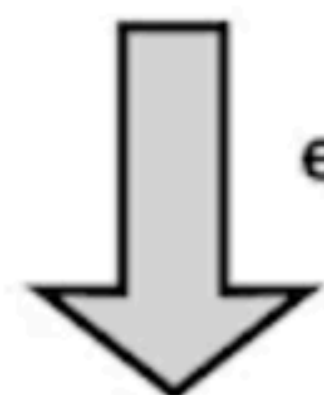
* The number in the parenthesis refers to the training size.

Prefix Tuning

Extrapolation to unseen categories

Trained on 9 categories

Astronaut, University, Monument, **Building**,
ComicsCharacter, Food, Airport,
SportsTeam, City, and WrittenWork



extrapolates

Test on 5 unseen categories

Athlete, **Artist**, MeanOfTransportation,
CelestialBody, Politician

x: [103_Colmore_Row | architect | John_Madin]
[John_Madin | birthPlace | Birmingham]
[Birmingham | leaderName | Andrew_Mitchell]

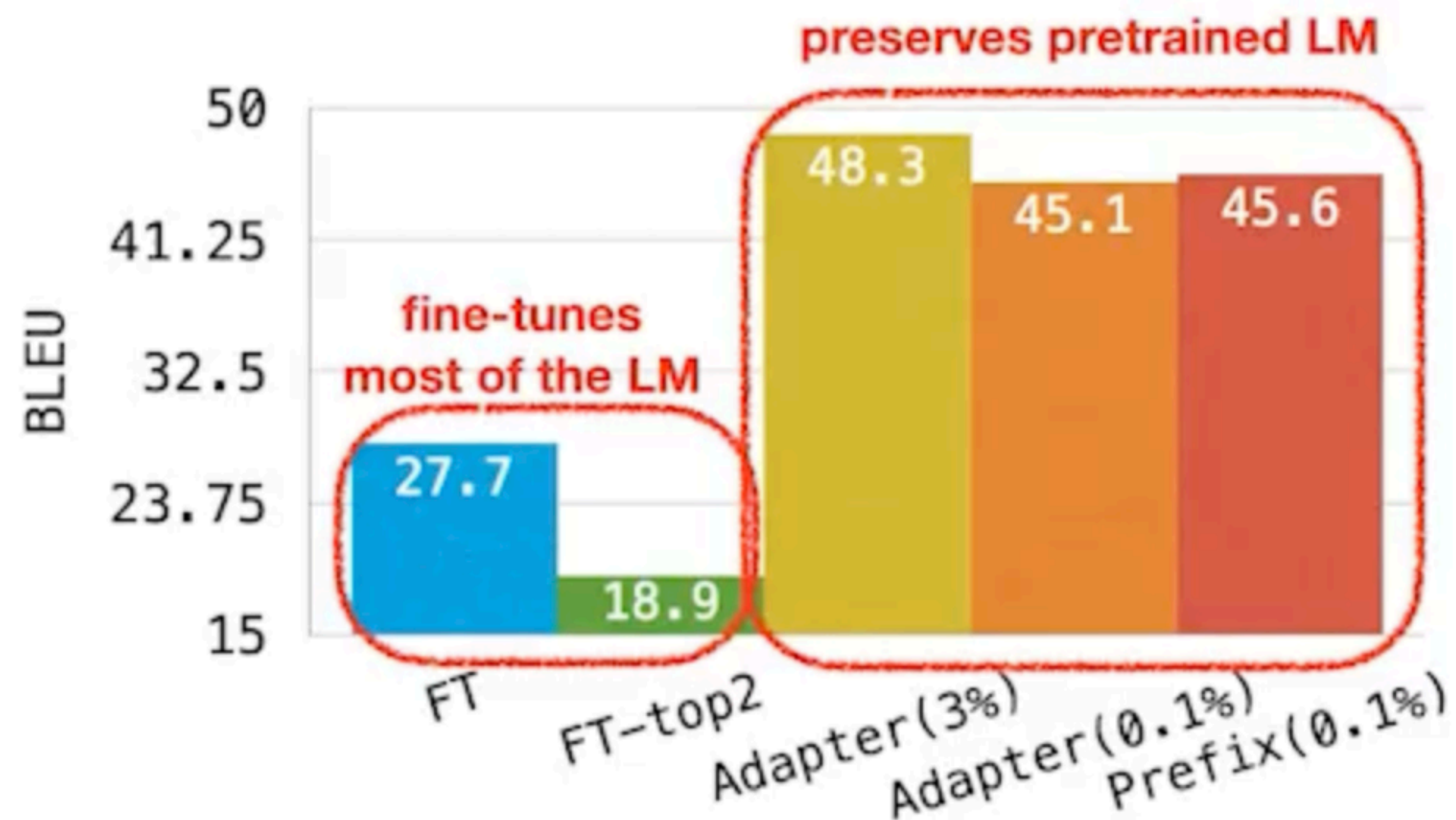
y: John Madin was born in Birmingham (with
Andrew Mitchell as a key leader) and became
an architect, designing 103 Colmore Row.

x: [Albennie_Jones | genre | Rhythm_and_blues]
[Albennie_Jones | birthPlace | Errata,_Mississippi]
[Rhythm_and_blues | derivative | Disco]

y: Albennie Jones, born in Errata, Mississippi, is
a performer of rhythm and blues, of which
disco is a derivative.

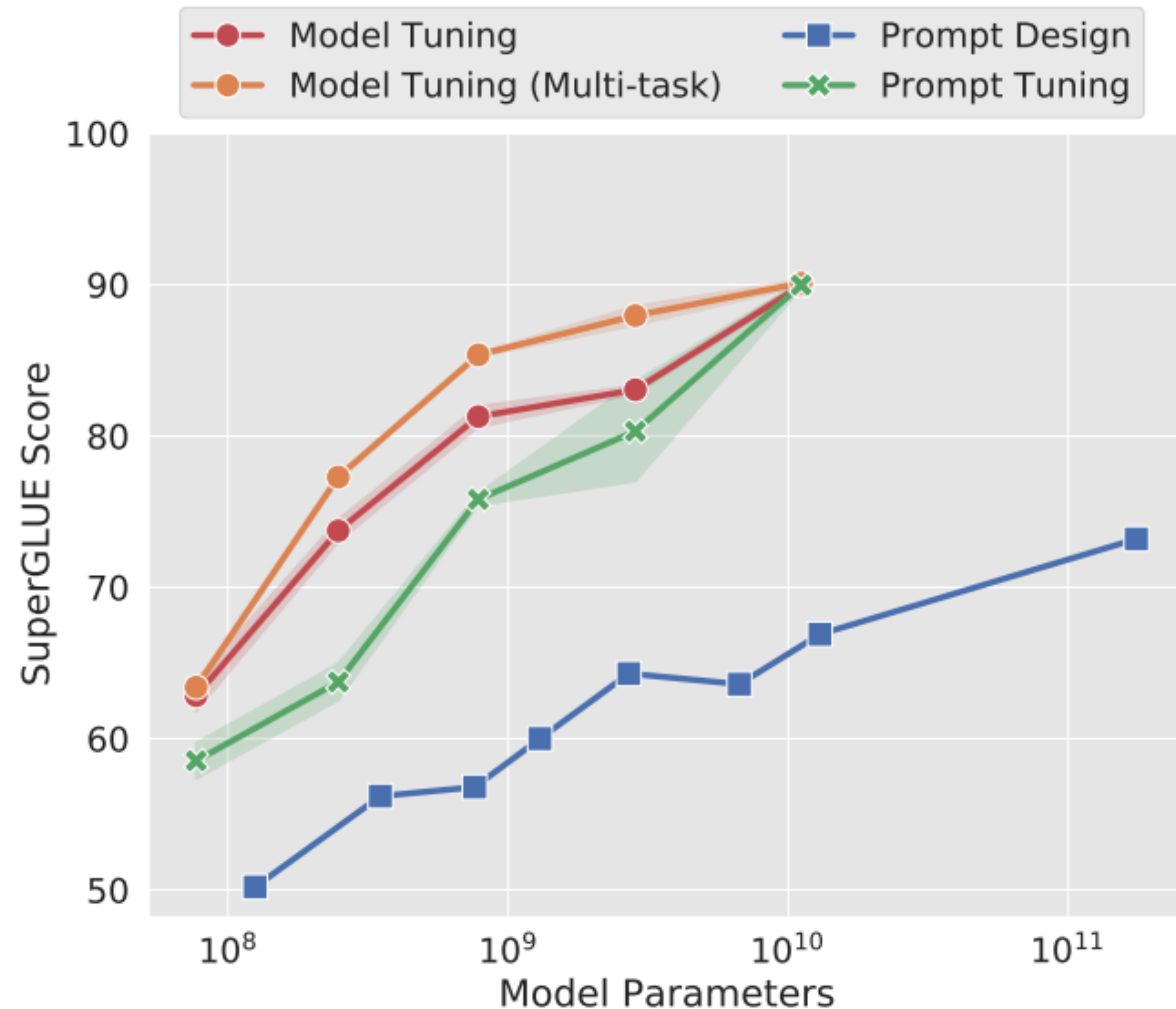
Prefix Tuning

Extrapolation to unseen categories



Prompt tuning works well at scale

- Only using trainable parameters at the input layer limits capacity for adaptation
- Prompt tuning performs poorly at smaller model sizes and on harder tasks



aka PaSTA

Parameter-Efficient Tuning with Special Token Adaptation

Xiaocong Yang^{†*}, James Y. Huang[‡], Wenxuan Zhou[‡] and Muhao Chen[‡]

[†]Tsinghua University; [‡]University of Southern California

`yangxc.18@sem.tsinghua.edu.cn;`

`{huangjam, zhouwenx, muhaoche}@usc.edu`

<https://aclanthology.org/2023.eacl-main.60.pdf>

Special tokens

typically capture
information from
global text

Attention is
focused on these
special tokens

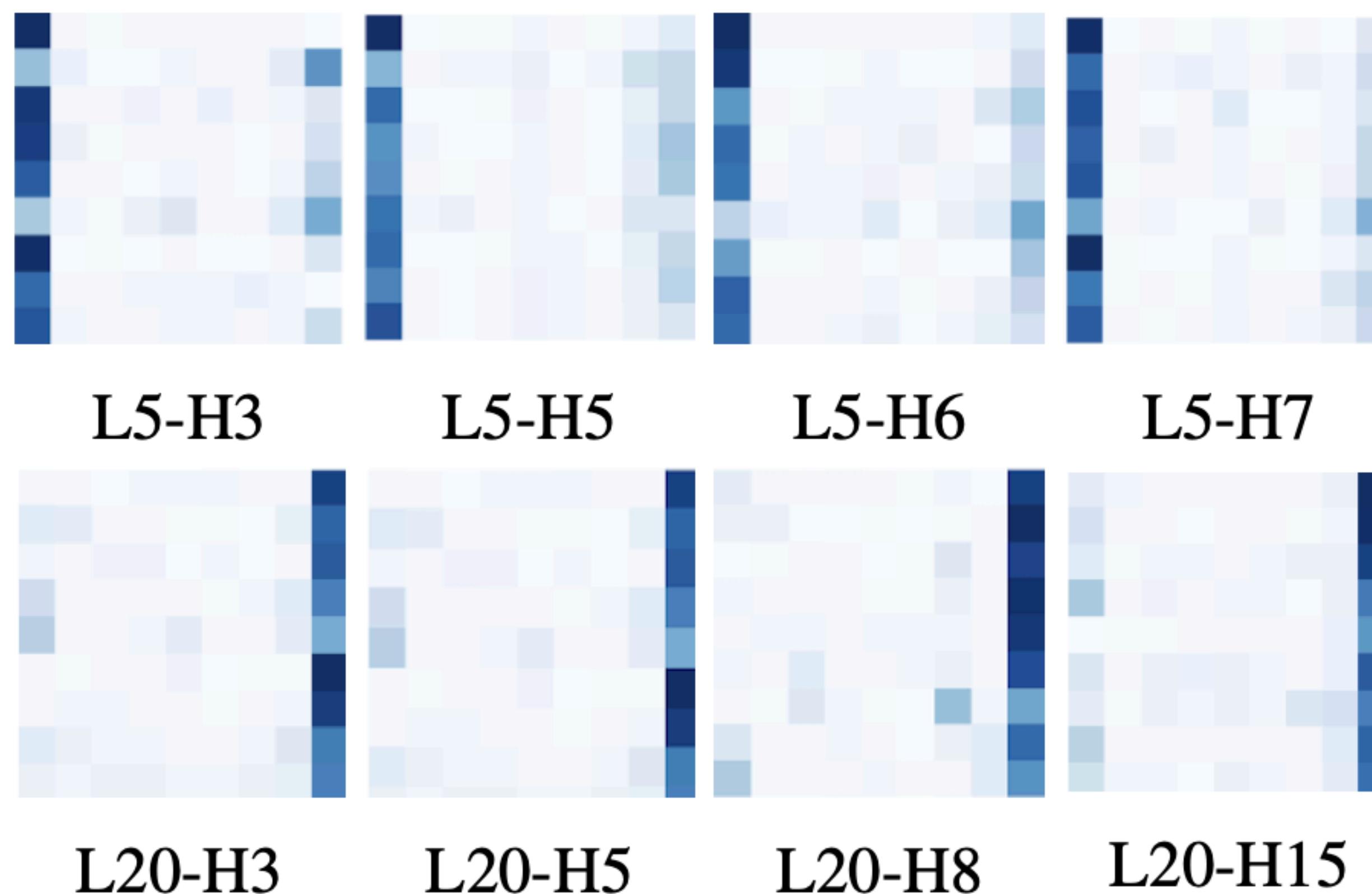


Figure 1: Examples of vertical attention heads in the 5-th and 20-th layer of BERT-large with a random sample from CoLA (Warstadt et al., 2019) as input. Heads in the first row and second row assign most of maximal attention weights to [CLS] and [SEP] respectively.

train a **hidden vector** for every special token

Self-attention allows for information to be spread to other tokens

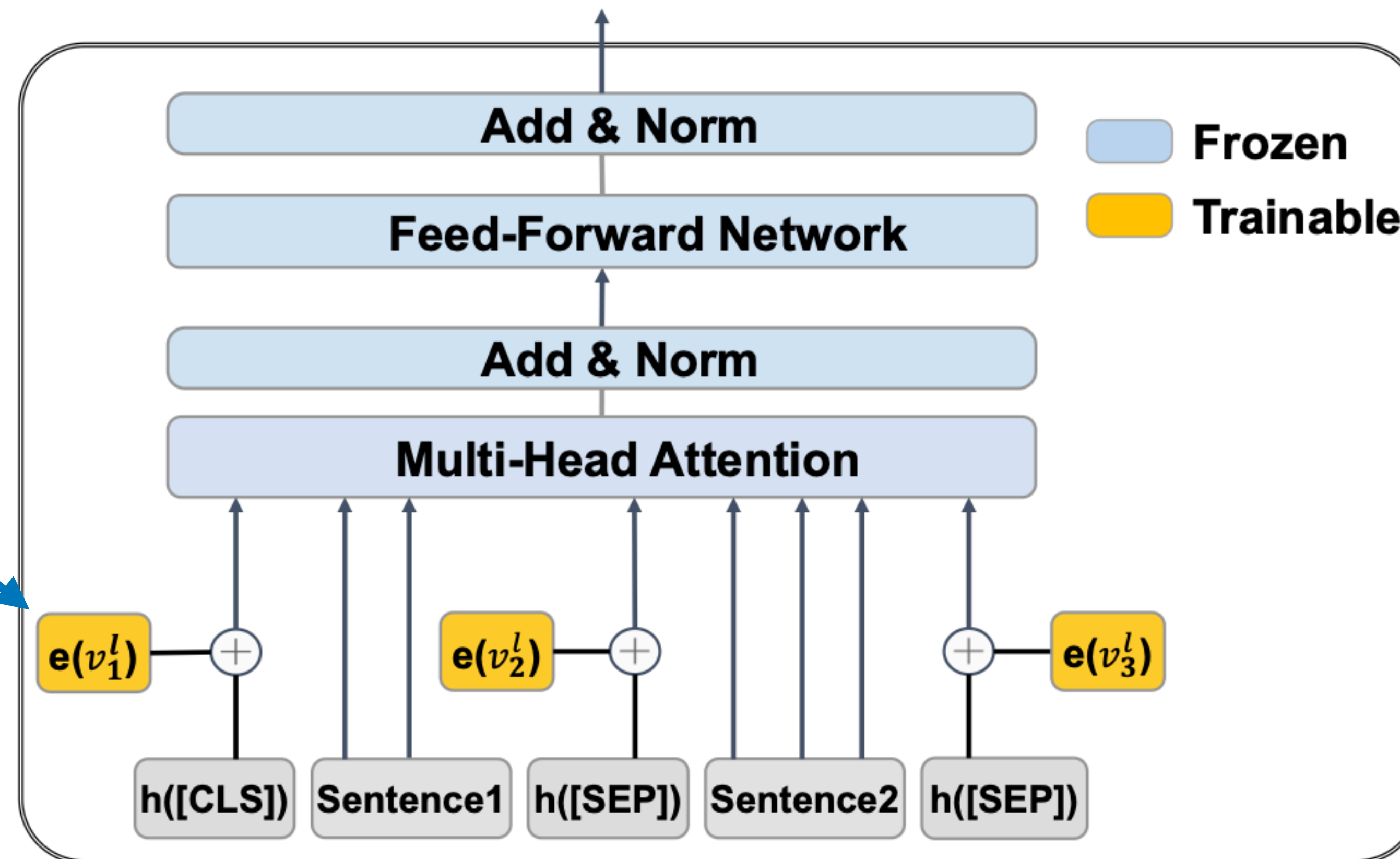


Figure 2: Architecture of PASTA layer in Transformer. Skip-connections in Transformers are not shown for brevity. At layer l we add a trainable vector $e(\mathbf{v}_p^l) \in \mathbb{R}^d$ to the hidden representation of the p -th special token in the input sequence, and freeze the weights of the PLM.

Results on GLUE with BERT-large

	%Param	RTE acc.	CoLA mcc.	STS-B Spearman	MRPC F1	SST-2 acc.	QNLI acc.	MNLI(m/mm) acc.	QQP F1	Avg.
Full Finetuning*	100%	70.1	60.5	86.5	89.3	94.9	92.7	86.7/85.9	72.1	81.6
Adapter**	3.6%	71.5	59.5	86.9	89.5	94.0	90.7	84.9/85.1	71.8	81.1
Diff-Prune [†]	0.5%	70.6	61.1	86.0	89.7	94.1	93.3	86.4/86.0	71.1	81.5
P-tuning v2	0.29%	70.1	60.1	86.8	88.0	94.6	92.3	85.3/84.9	70.6	81.0
BitFit [‡]	0.08%	72.0	59.7	85.5	88.9	94.2	92.0	84.5/84.8	70.5	80.9
PASTA	0.015%-0.022%	70.8	62.3	86.6	87.9	94.4	92.8	83.4/83.4	68.6	80.9

Ablation study on GLUE and CoNLL-2003

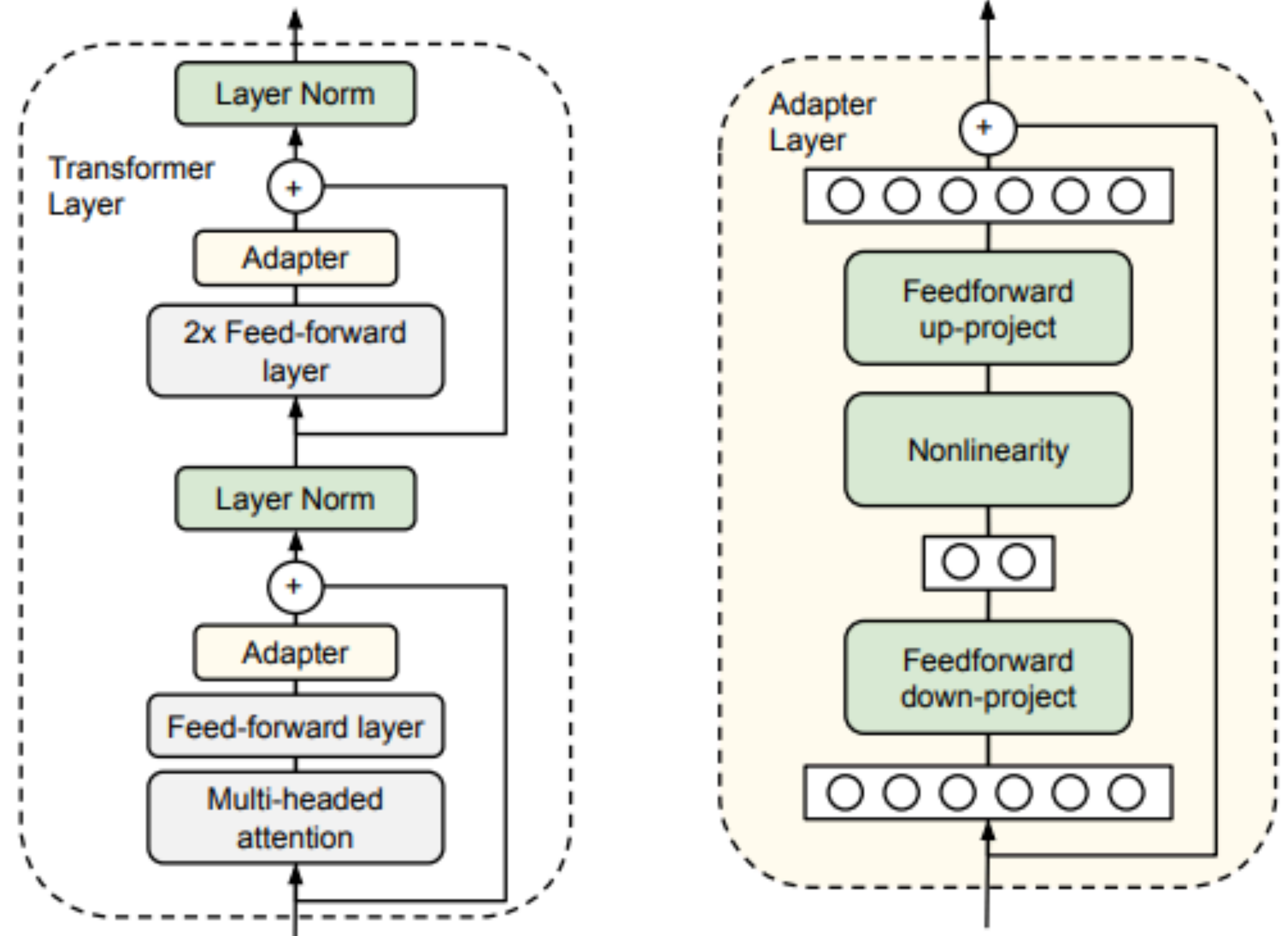
	CoLA	RTE	MRPC	STS-B	CoNLL2003
PASTA	65.4	76.2	89.7	90.8	94.0
- w/o [CLS]	58.8	72.6	91.4	90.2	93.7
- w/o [SEP]	64.5	71.1	91.9	90.3	93.7
- shared vector	64.7	74.7	92.1	90.0	93.9
- classifier only	36.5	54.2	81.5	64.9	77.4

Table 4: Performance of ablation study with BERT-large on GLUE and CoNLL2003 development sets.

Adapters

Adapters

- Insert function into model blocks to adapt it to a downstream task
- Adapter typically placed after the multi-head attention and/or after the feed-forward layer

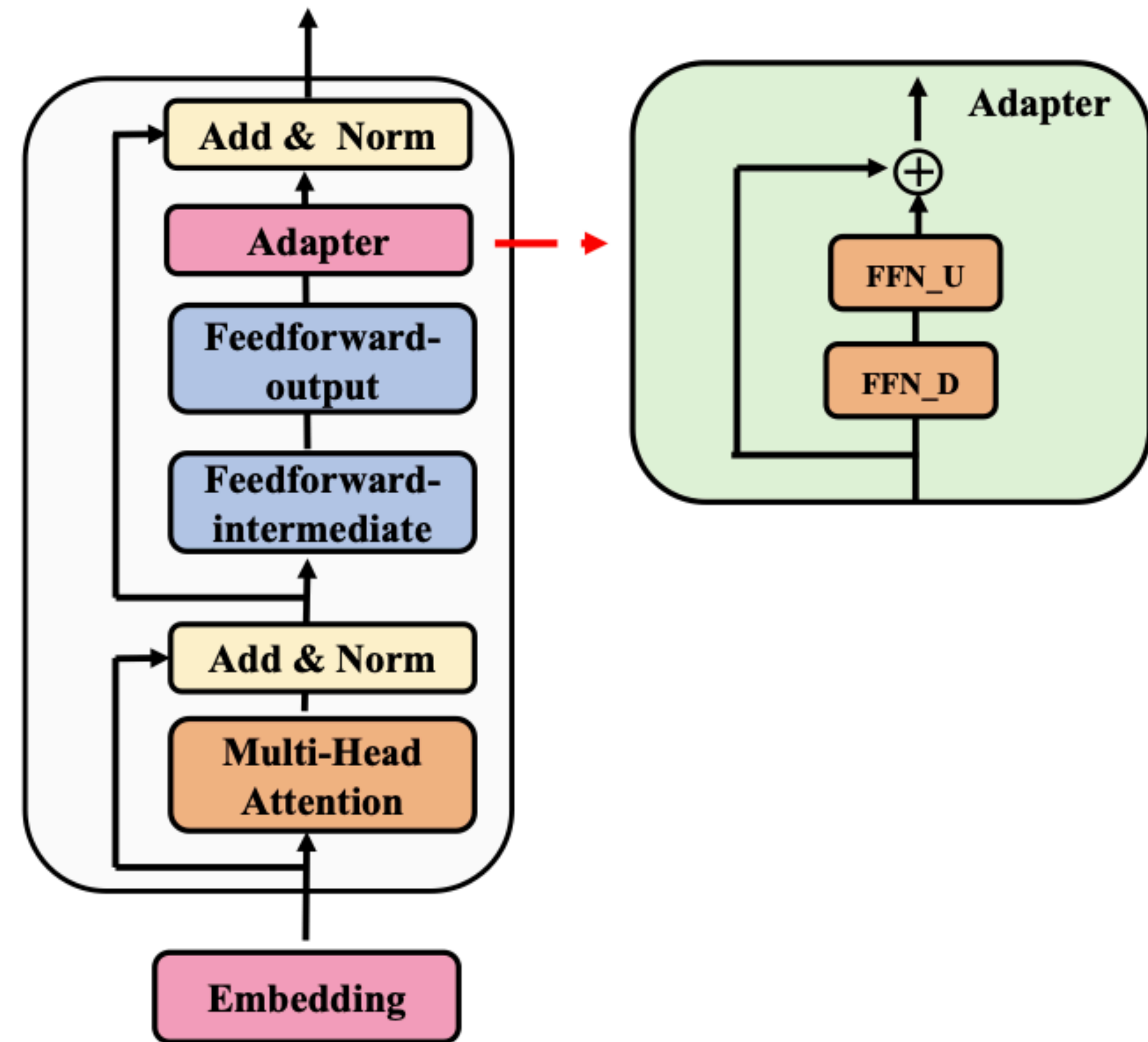


Parameter-Efficient Transfer Learning for NLP
[Houlsby et al., 2019]

<https://arxiv.org/abs/1902.00751>

Bottleneck Adapters

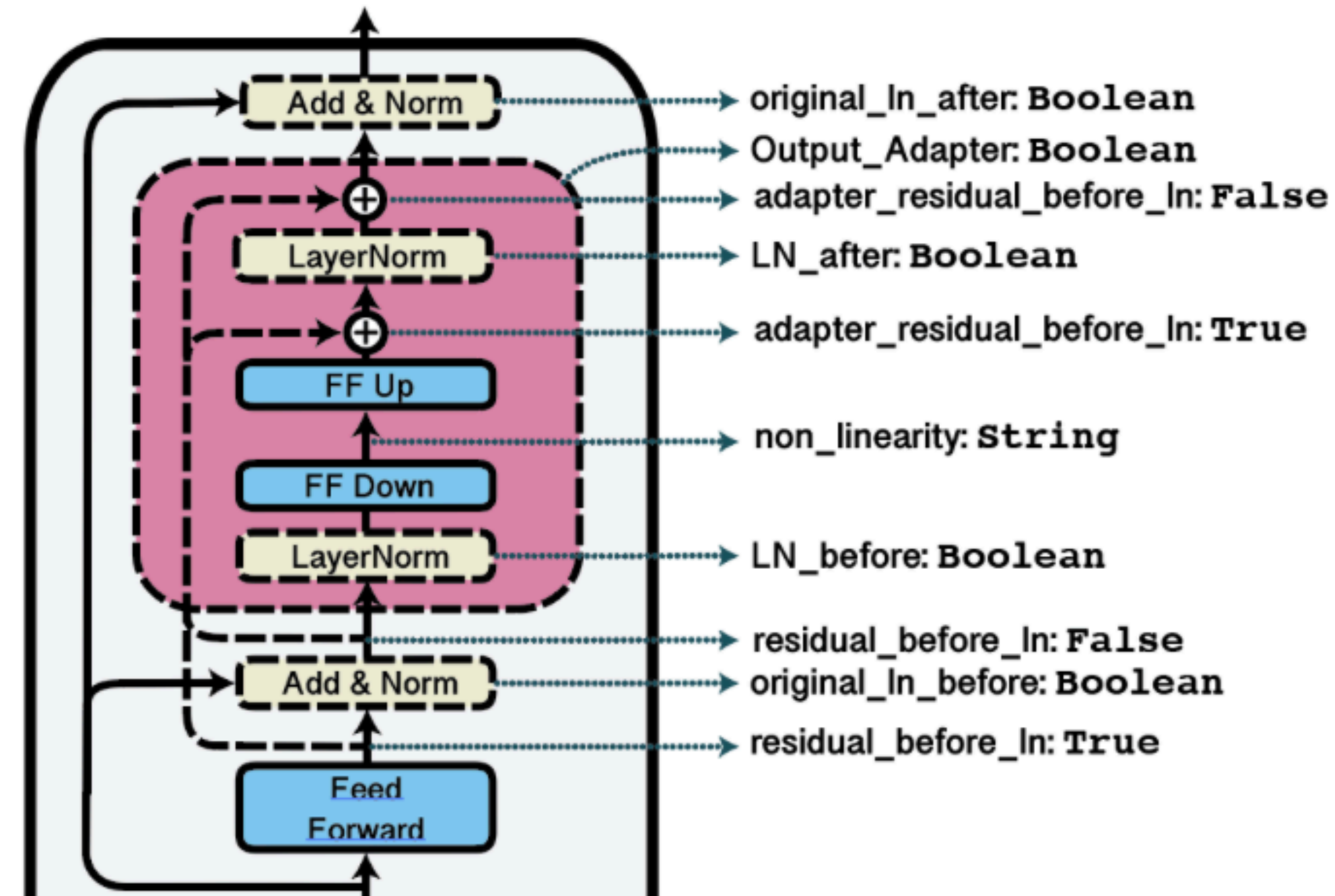
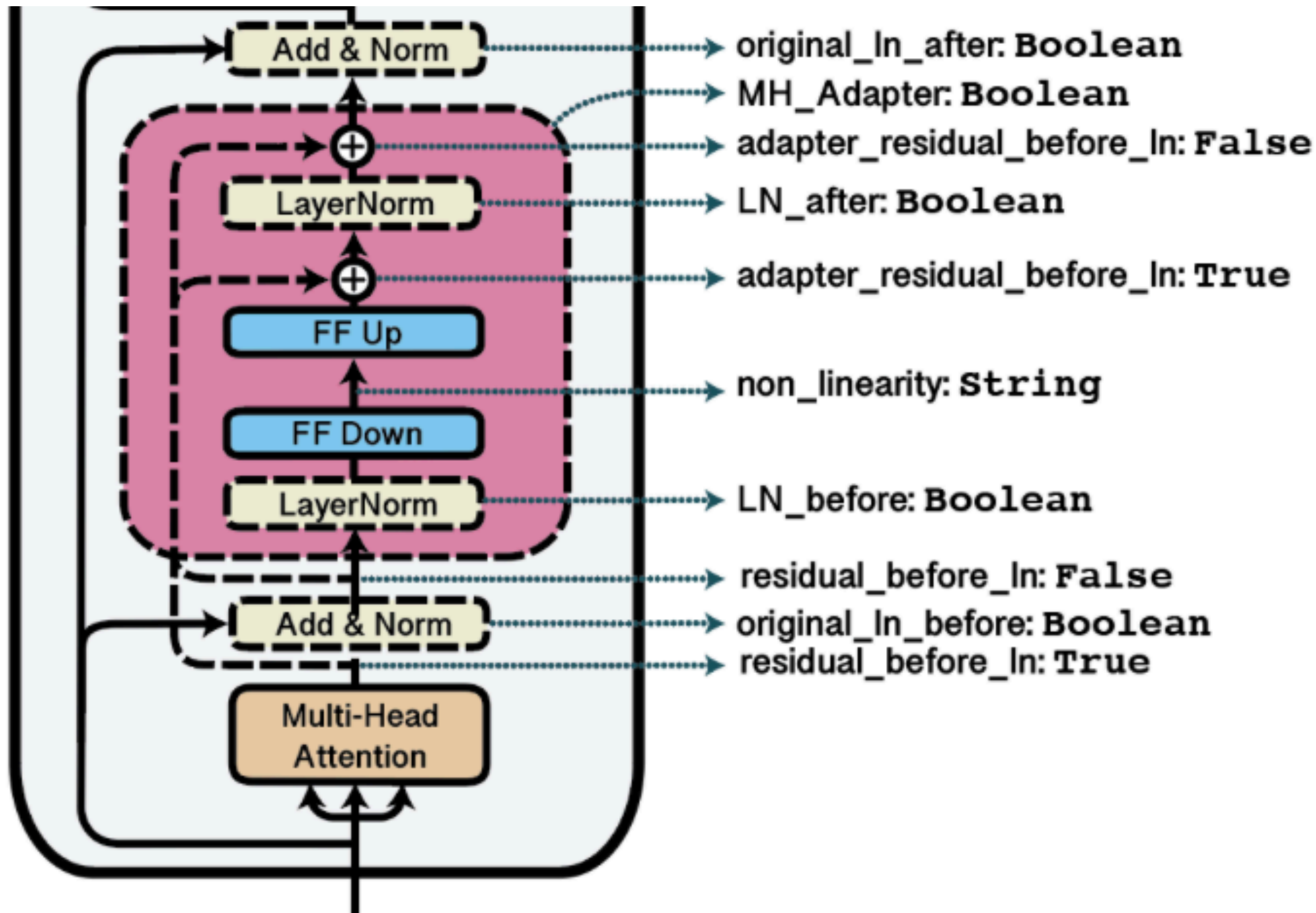
- Given a hidden layer h^ℓ for layer ℓ in a Transformer layer (before Add & Norm)
- $h^\ell \leftarrow h^\ell + f(h^\ell \cdot W_{down}) \cdot W_{up}$
- W_{down} lowers the dimensionality from $\dim(h^\ell)$ down to k where $k \ll \dim(h^\ell)$
- W_{up} raises the dimensionality from k back up to $\dim(h^\ell)$
- f is a non-linear function (GeLU)
- $h^{\ell+1} = \text{Add+LN}(h^\ell)$



<https://arxiv.org/abs/2205.12410>

Also see: <https://www.cs.huji.ac.il/labs/learning/Papers/allerton.pdf>

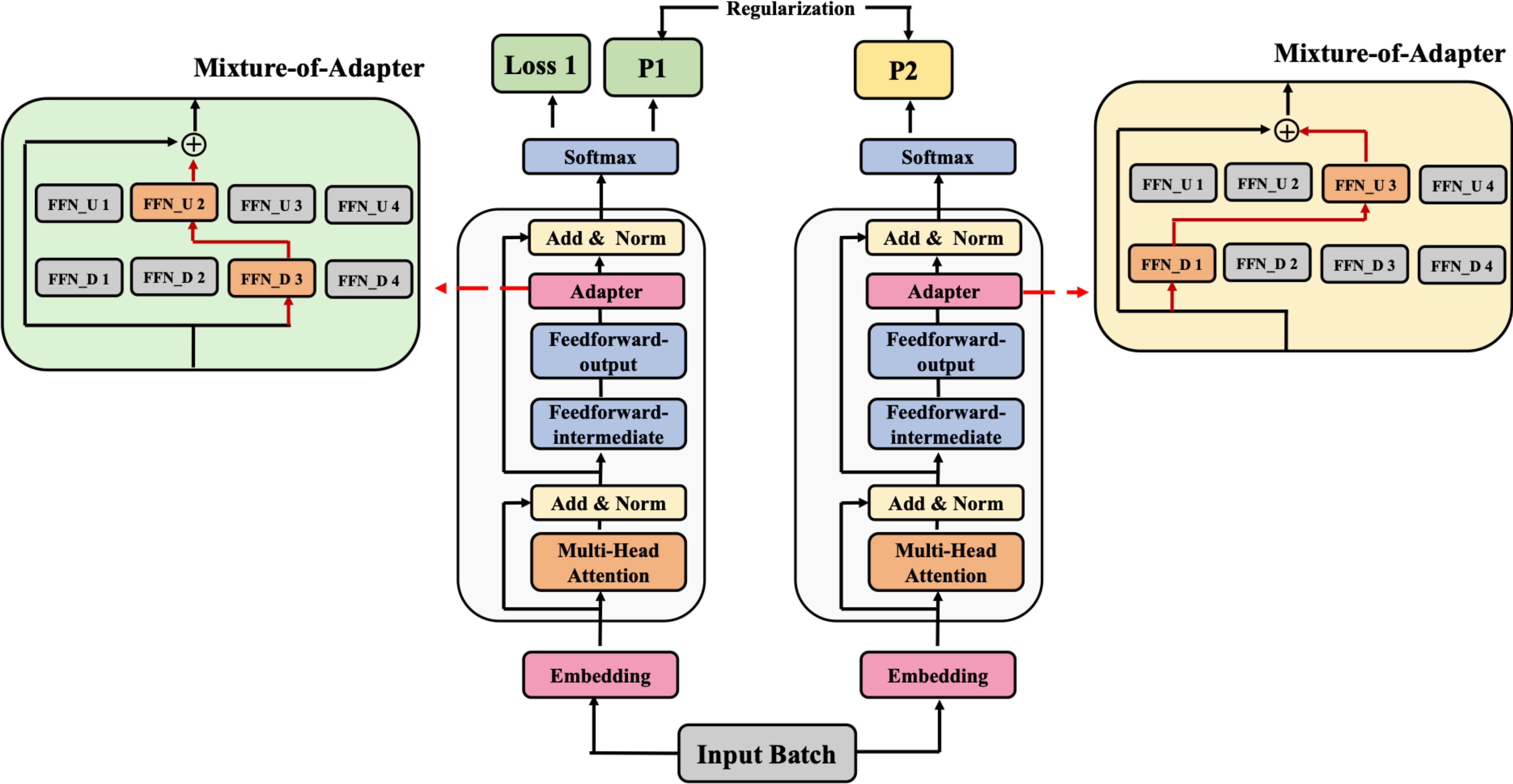
Bottleneck Adapters



<https://docs.adapterhub.ml/>

Mixture of Adapters

<https://arxiv.org/abs/2205.12410>



Mixture of Adapters

Regularization loss

- For each layer ℓ use M different feed-forward networks for projecting down to k and for projecting up to $\dim(h^\ell)$
- $A_\ell = \{W_{down}^{\ell,j}, W_{down}^{\ell,k}\}$ and $B_\ell = \{W_{up}^{\ell,j}, W_{up}^{\ell,k}\}$
- where $j, k \in [0, M - 1]$
- $h^\ell \leftarrow h^\ell + f(h^\ell \cdot W_{down}^{\ell,i}) \cdot W_{up}^{\ell,j}$
- Pick i, j at random
- Pick i, j twice for each input batch.

Mixture of Adapters

Regularization loss

- Fine tuning loss: $\mathcal{L} = - \sum_{c=1}^C \delta(x, \hat{x}) \log \text{softmax}((z^{\mathcal{A}}(x)))$
- where δ is 1 if the two arguments are equal
- \hat{x} is the right answer for input x
- $z^{\mathcal{A}}(x)$ are the logits for the fine-tuning output softmax activation (using adapter \mathcal{A})

Mixture of Adapters

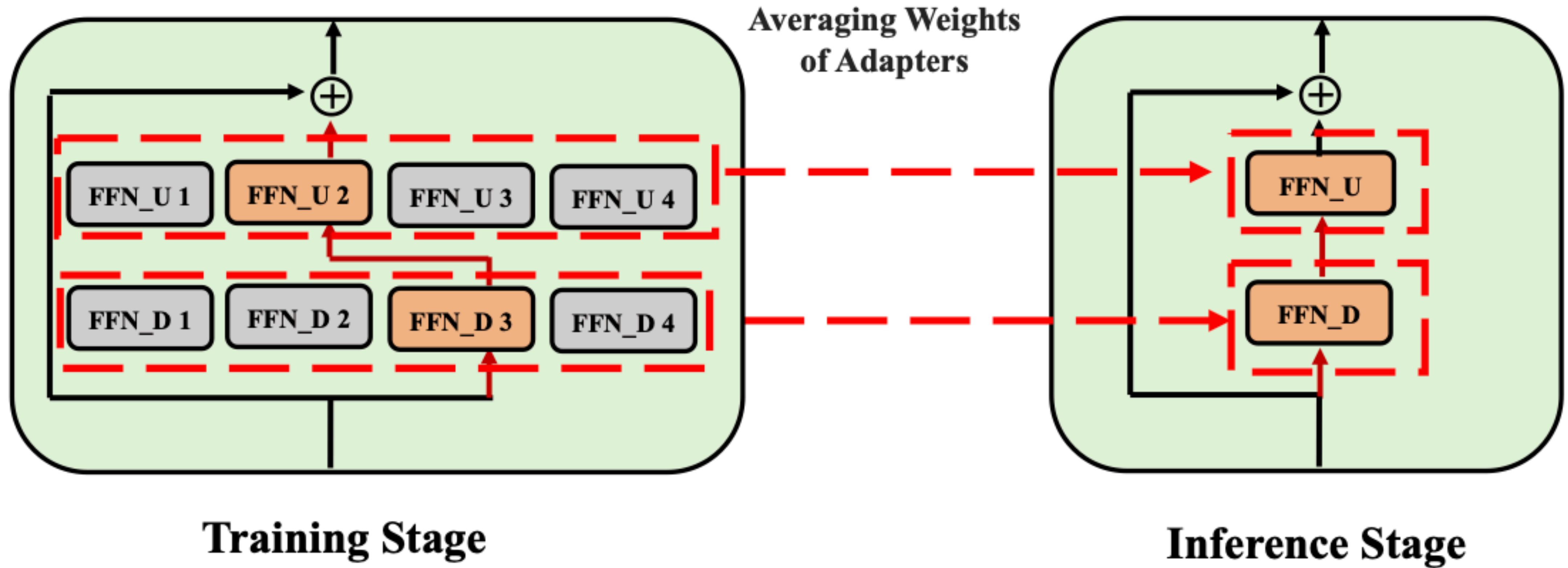
Regularization loss

- Let $\mathcal{A} = \{A_{\ell=1}^L\}$ and $\mathcal{B} = \{B_{\ell=1}^L\}$ be the adapter modules.
- Pick i, j twice for each input batch.
- Let $D(\mathcal{X}, \mathcal{Y}) = KL(z^{\mathcal{X}}(x) || z^{\mathcal{Y}}(x))$ where x is the input to the LLM with frozen parameters; only \mathcal{X}, \mathcal{Y} are trained against fine-tuning prediction loss.
- Add following consistency loss to fine-tuning a LLM

$$\bullet \mathcal{L} \leftarrow \mathcal{L} + \frac{1}{2}(D(\mathcal{A}, \mathcal{B}) + D(\mathcal{B}, \mathcal{A}))$$

Mixture of Adapters

<https://arxiv.org/abs/2205.12410>



$$W_{down}^{\ell} = \frac{1}{M} \sum_{j=1}^M W_{down}^{\ell,j}$$

$$W_{up}^{\ell} = \frac{1}{M} \sum_{j=1}^M W_{up}^{\ell,j}$$

Results on GLUE with ROBERTa-large

Model	#Param.	MNLI Acc	QNLI Acc	SST2 Acc	QQP Acc	MRPC Acc	CoLA Mcc	RTE Acc	STS-B Pearson	Avg.
Full Fine-tuning [†]	355.0M	90.2	94.7	96.4	92.2	90.9	68.0	86.6	92.4	88.9
Pfeiffer Adapter [†]	3.0M	90.2	94.8	96.1	91.9	90.2	68.3	83.8	92.1	88.4
Pfeiffer Adapter [†]	0.8M	90.5	94.8	96.6	91.7	89.7	67.8	80.1	91.9	87.9
Houlsby Adapter [†]	6.0M	89.9	94.7	96.2	92.1	88.7	66.5	83.4	91.0	87.8
Houlsby Adapter [†]	0.8M	90.3	94.7	96.3	91.5	87.7	66.3	72.9	91.5	86.4
LoRA [†]	0.8M	90.6	94.8	96.2	91.6	90.2	68.2	85.2	92.3	88.6
AdaMix Adapter	0.8M	90.9	95.4	97.1	92.3	91.9	70.2	89.2	92.4	89.9

<https://arxiv.org/abs/2205.12410>

Results on GLUE with BERT-base

Model	#Param.	Avg.
Full Fine-tuning [†]	110M	82.7
Houlsby Adapter [†]	0.9M	83.0
BitFit [◇]	0.1M	82.3
Prefix-tuning [†]	0.2M	82.1
LoRA [†]	0.3M	82.2
UNIPELT (AP) [†]	1.1M	83.1
UNIPELT (APL) [†]	1.4M	83.5
AdaMix Adapter	0.9M	84.5

Results on E2E with GPT2-medium

Model	#Param.	BLEU	NIST	MET	ROUGE-L	CIDEr
Full Fine-tuning [†]	354.92M	68.2	8.62	46.2	71.0	2.47
Lin AdapterL [†]	0.37M	66.3	8.41	45.0	69.8	2.40
Lin Adapter [†]	11.09M	68.9	8.71	46.1	71.3	2.47
Houlsby Adapter [†]	11.09M	67.3	8.50	46.0	70.7	2.44
FT ^{Top2} [†]	25.19M	68.1	8.59	46.0	70.8	2.41
PreLayer [†]	0.35M	69.7	8.81	46.1	71.4	2.49
LoRA [†]	0.35M	70.4	8.85	46.8	71.8	2.53
LoRA (repr.)	0.35M	69.8	8.77	46.6	71.8	2.52
AdaMix Adapter	0.42M	69.8	8.75	46.8	71.9	2.52
AdaMix LoRA	0.35M	71.0	8.89	46.8	72.2	2.54

Simple, Scalable Adaptation for Neural Machine Translation

Ankur Bapna

Naveen Arivazhagan

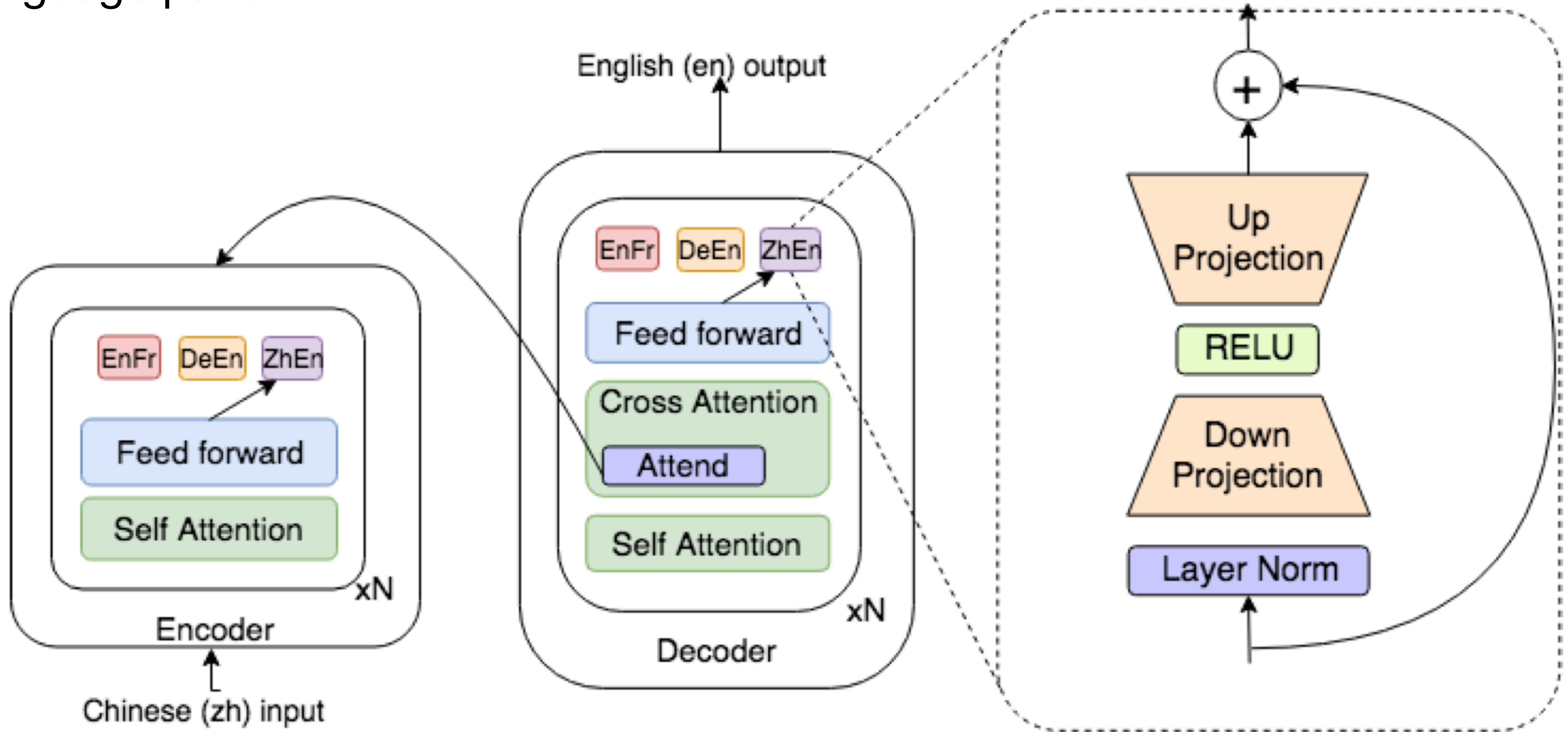
Orhan Firat

Google AI

`{ankurbpn, navari, orhanf}@google.com`

<https://arxiv.org/abs/1909.08478>

- Different adapters for different language pairs



LoRA: LOW-RANK ADAPTATION OF LARGE LANGUAGE MODELS

Edward Hu* **Yelong Shen*** **Phillip Wallis** **Zeyuan Allen-Zhu**
Yuanzhi Li **Shean Wang** **Lu Wang** **Weizhu Chen**

Microsoft Corporation

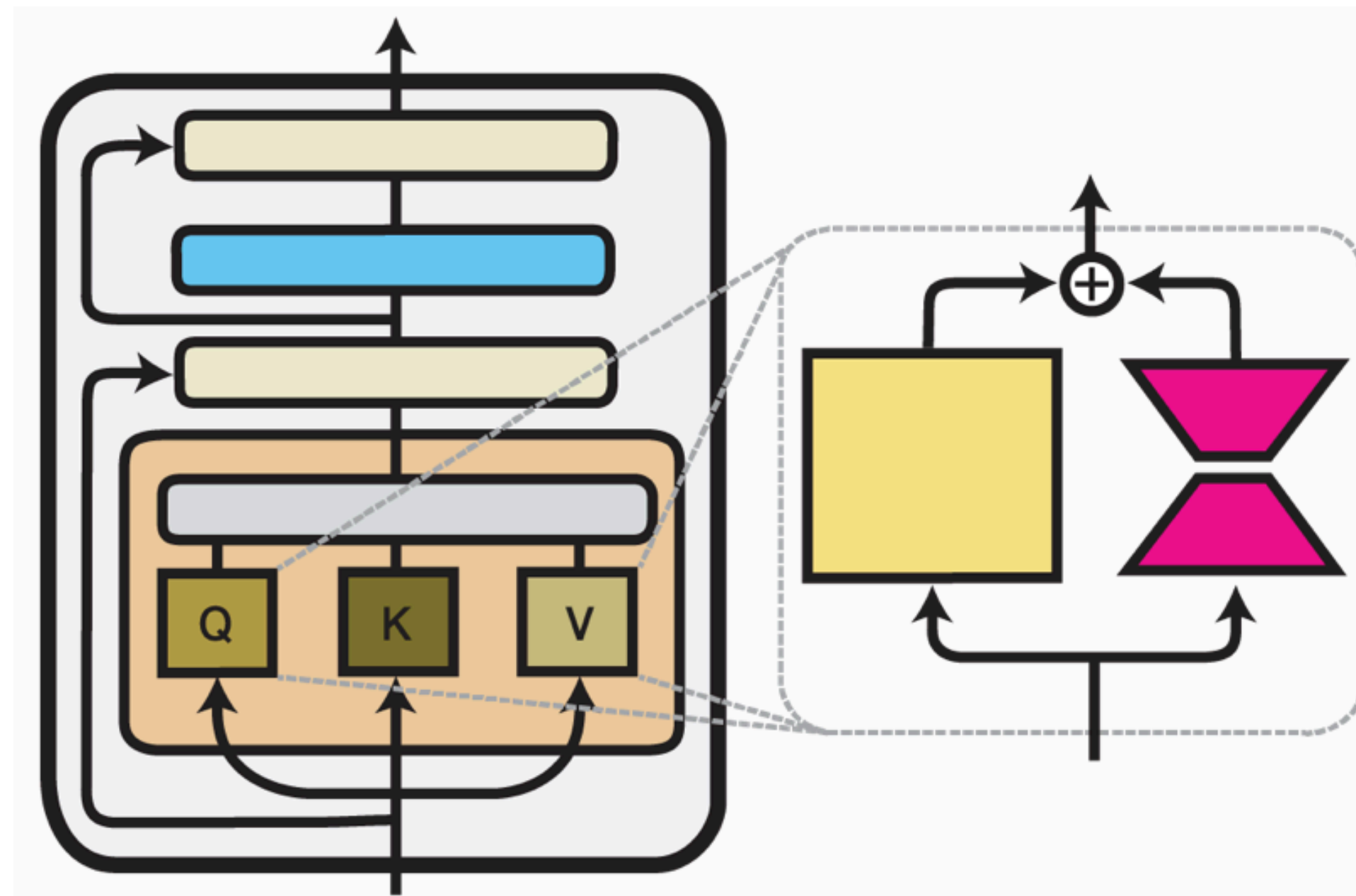
{edwardhu, yeshe, phwallis, zeyuana,
yuanzhil, swang, luw, wzchen}@microsoft.com

yuanzhil@andrew.cmu.edu

(Version 2)

LoRA

- Can be applied to any Transformer-based Large Language Model
- But specifically designed for autoregressive and causal LMs like GPTx
- Just like other Transformer adapters, LoRA adds a small set of parameters for fine-tuning and keeps the original parameters frozen
- This can help a lot when LLM parameter sizes are as large as 175 billion.

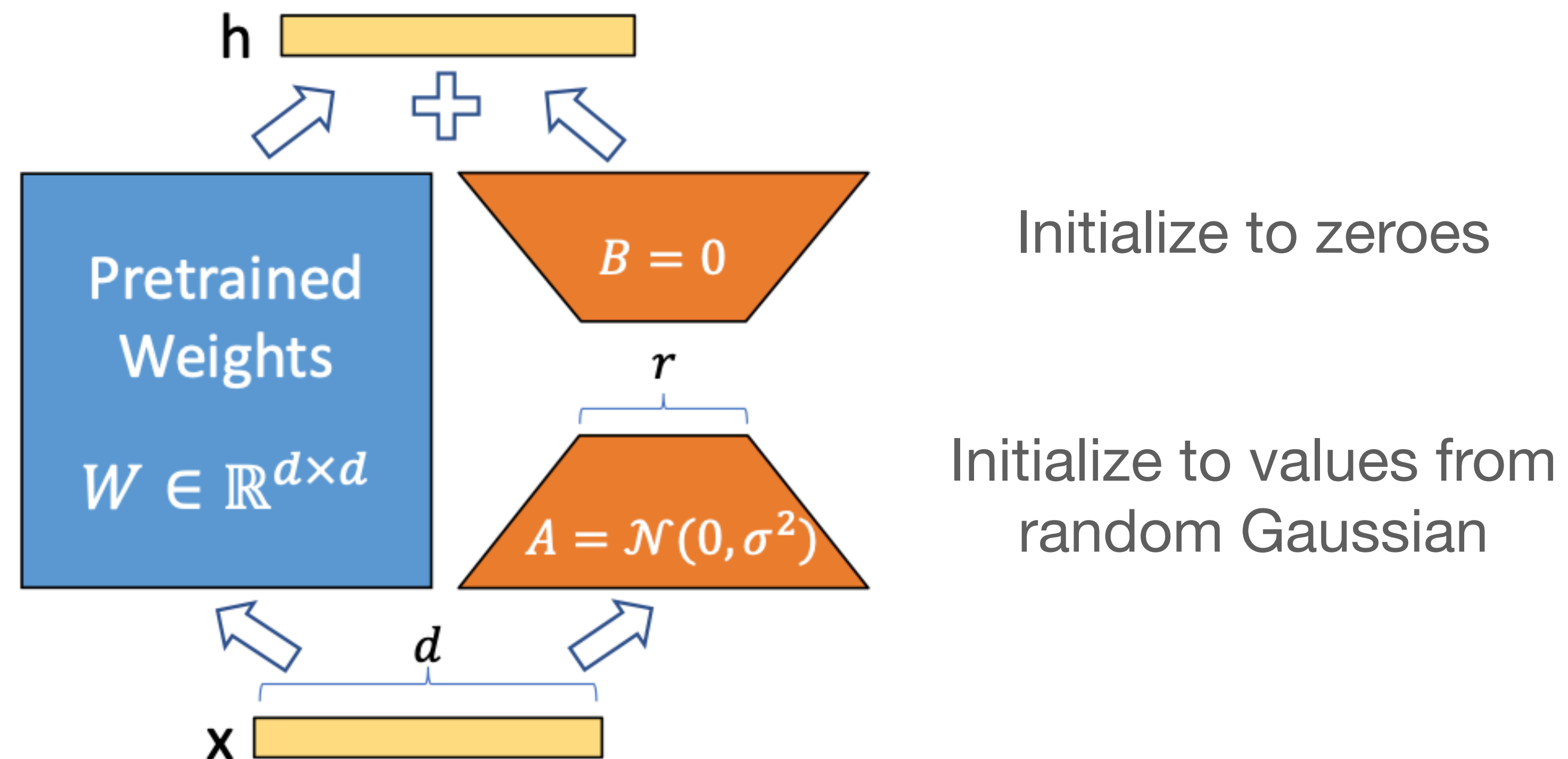


LoRA

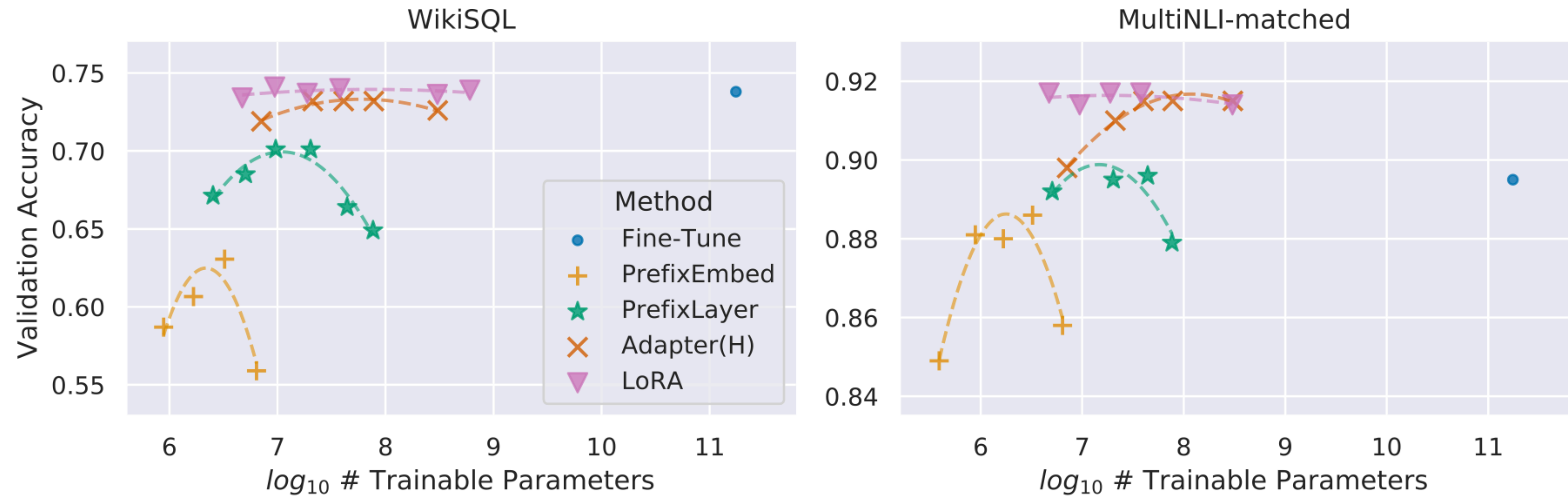
- Only use adapters in the attention matrices: Q, K, V
- Each matrix is called W_p here, p for pre-trained
- Adapter methods modify W_p to be $W_p + BA$ where $B \in \mathbb{R}^{d \times r}$, $A \in \mathbb{R}^{r \times k}$
- Rank $r \ll \min(d, k)$
- Let BA be zero at start of training
- Scale the parameters after backpropagation by $\frac{\alpha}{r}$ where α is a hyperparameter set to a constant value depending on r (set to the first r in training)

LoRA

- Initialize B to zeroes
- Initialize A using random Gaussian initialization

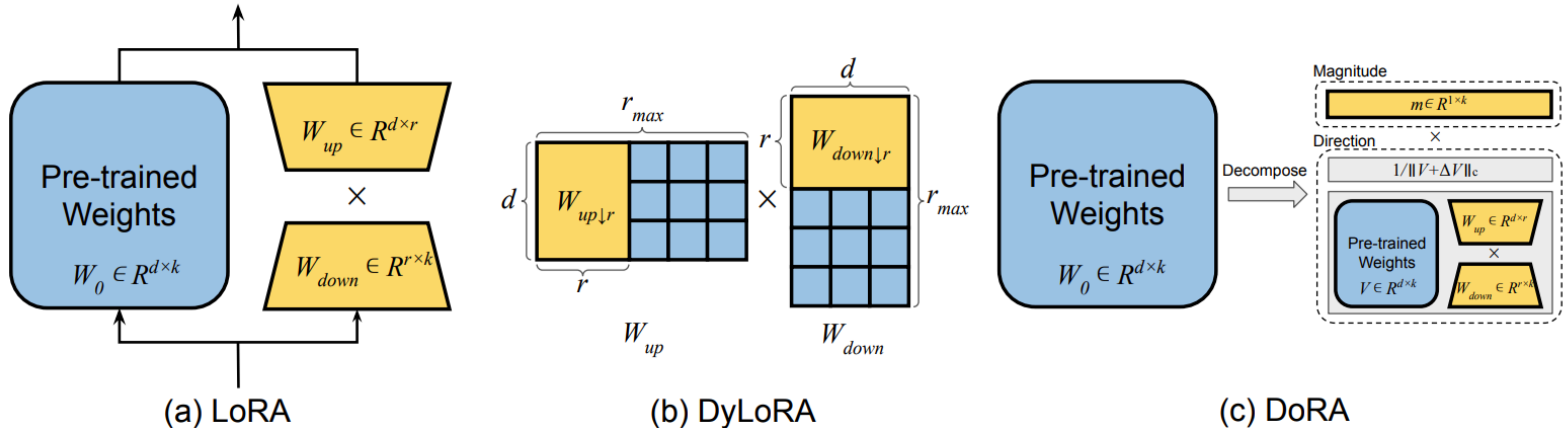


Model & Method	# Trainable Parameters	E2E NLG Challenge				
		BLEU	NIST	MET	ROUGE-L	CIDEr
GPT-2 M (FT)*	354.92M	68.2	8.62	46.2	71.0	2.47
GPT-2 M (Adapter ^L)*	0.37M	66.3	8.41	45.0	69.8	2.40
GPT-2 M (Adapter ^L)*	11.09M	68.9	8.71	46.1	71.3	2.47
GPT-2 M (Adapter ^H)	11.09M	67.3 \pm .6	8.50 \pm .07	46.0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



GPT-3 175B validation accuracy vs. number of trainable parameters

Extensions to LoRA



- DyLoRA: dynamically select rank (up to r_{max})
- AdaLoRA / SoRA: SVD decomposition ($\Delta W = P\Lambda Q$, rank controlled by pruning Λ)
- DoRA: Decompose weight matrix into magnitude and direction

DoRA: Weight decomposed low-rank adaptation

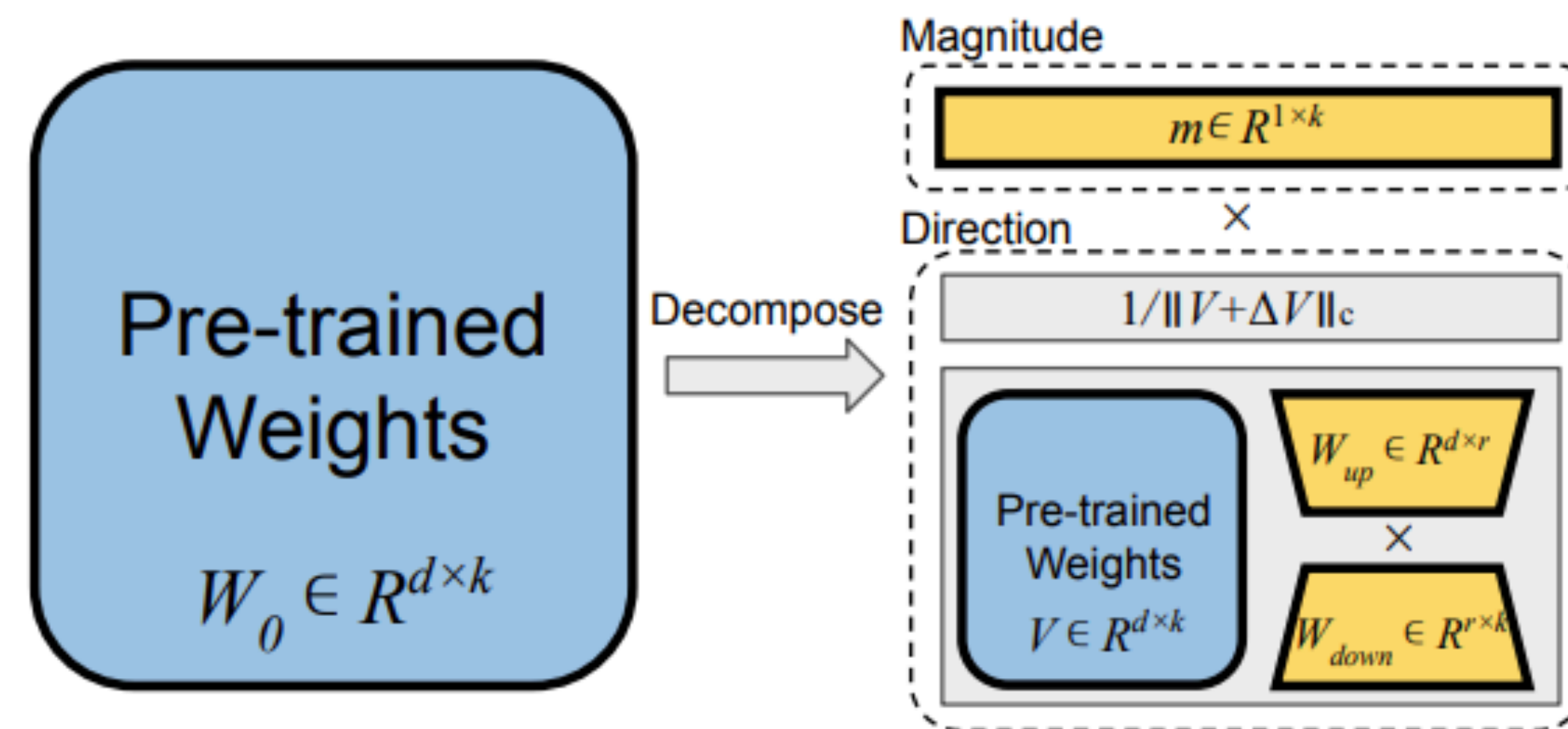
- Decompose weight matrix into magnitude and direction

$$W_0 = m \frac{V}{\|V\|_c} = \|W_0\|_c \frac{W_0}{\|W_0\|_c}$$

Vector-wise norm across each column
(each column is now a unit vector)

- Magnitude $m \in \mathbb{R}^{1 \times k}$
- Direction $V \in \mathbb{R}^{d \times k}$
- Only perform LoRA reparameterization on V
- Separately tune magnitude

$$W' = \underline{m} \frac{V + \Delta V}{\|V + \Delta V\|_c} = \underline{m} \frac{W_0 + \underline{BA}}{\|W_0 + \underline{BA}\|_c}$$



DoRA: Weight decomposed low-rank adaptation

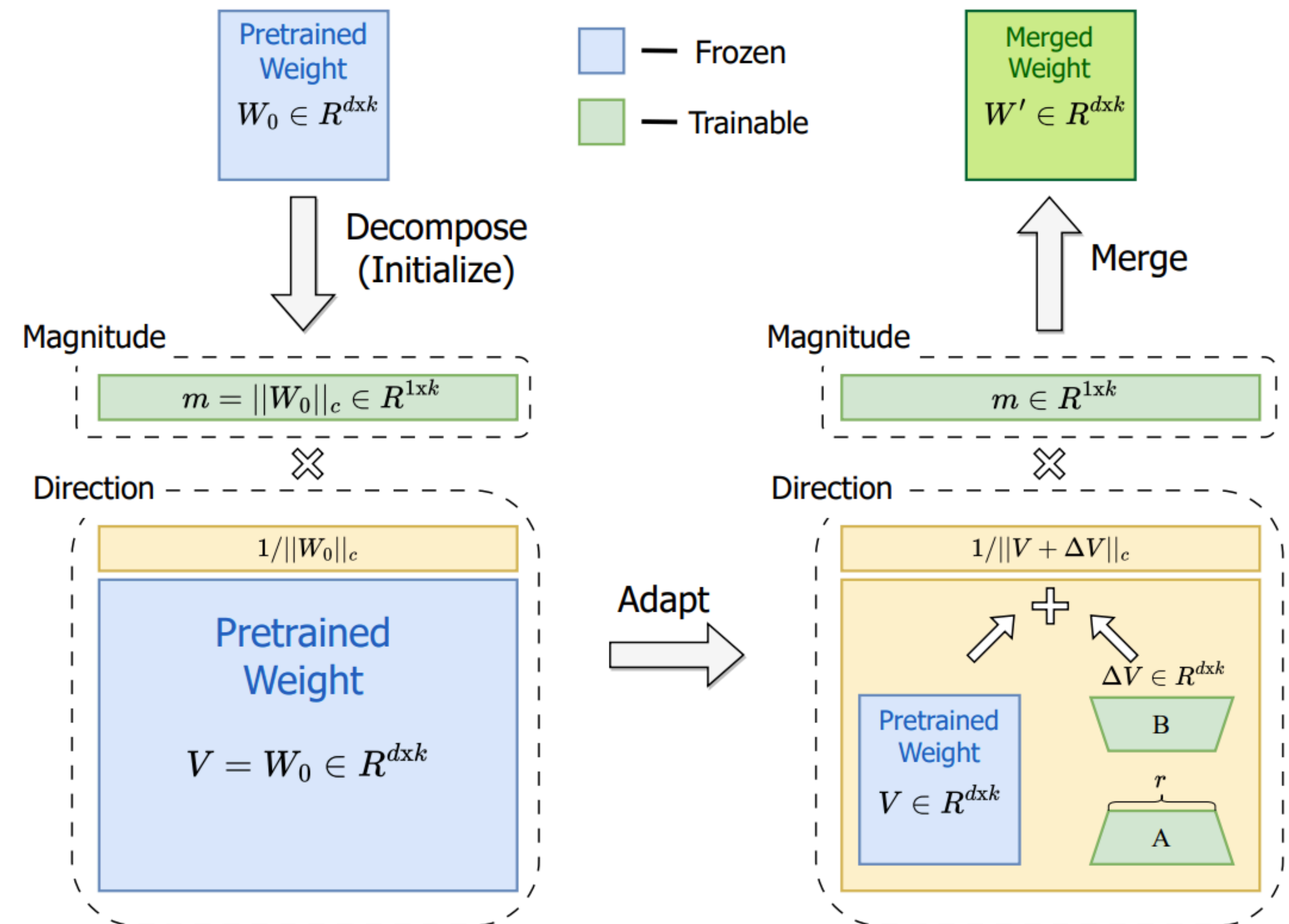
- Decompose weight matrix into magnitude and direction

$$W_0 = m \frac{V}{\|V\|_c} = \|W_0\|_c \frac{W_0}{\|W_0\|_c}$$

Vector-wise norm across each column
(each column is now a unit vector)

- Magnitude $m \in \mathbb{R}^{1 \times k}$
- Direction $V \in \mathbb{R}^{d \times k}$
- Only perform LoRA reparameterization on V
- Separately tune magnitude

$$W' = \underline{m} \frac{V + \Delta V}{\|V + \Delta V\|_c} = \underline{m} \frac{W_0 + \underline{BA}}{\|W_0 + \underline{BA}\|_c}$$



DoRA: Weight decomposed low-rank adaptation

Learned parameter adjustments (ΔD , ΔM) are more similar to those of full fine-tuning

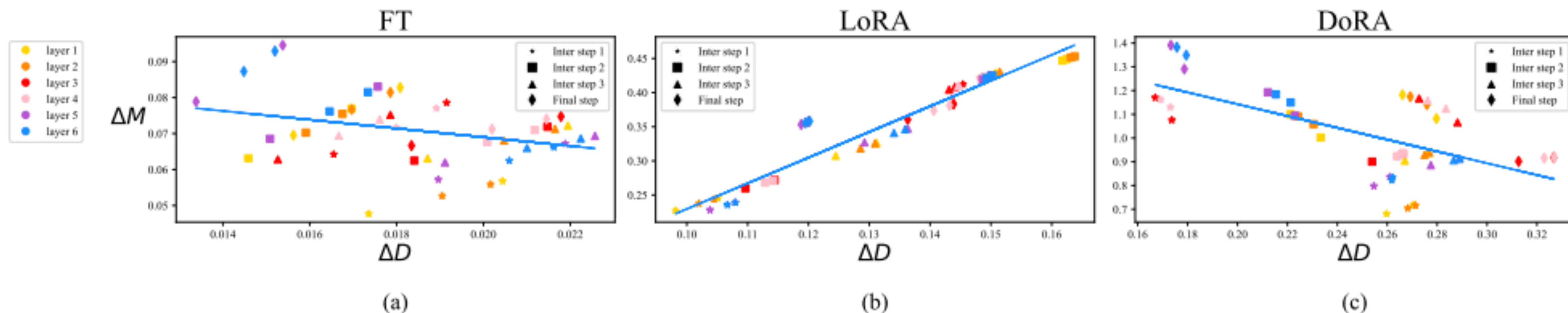


Figure 2. Magnitude and direction updates of (a) FT, (b) LoRA, and (c) DoRA of the query matrices across different layers and intermediate steps. Different markers represent matrices of different training steps and different colors represent the matrices of each layer.

DoRA: Weight decomposed low-rank adaptation

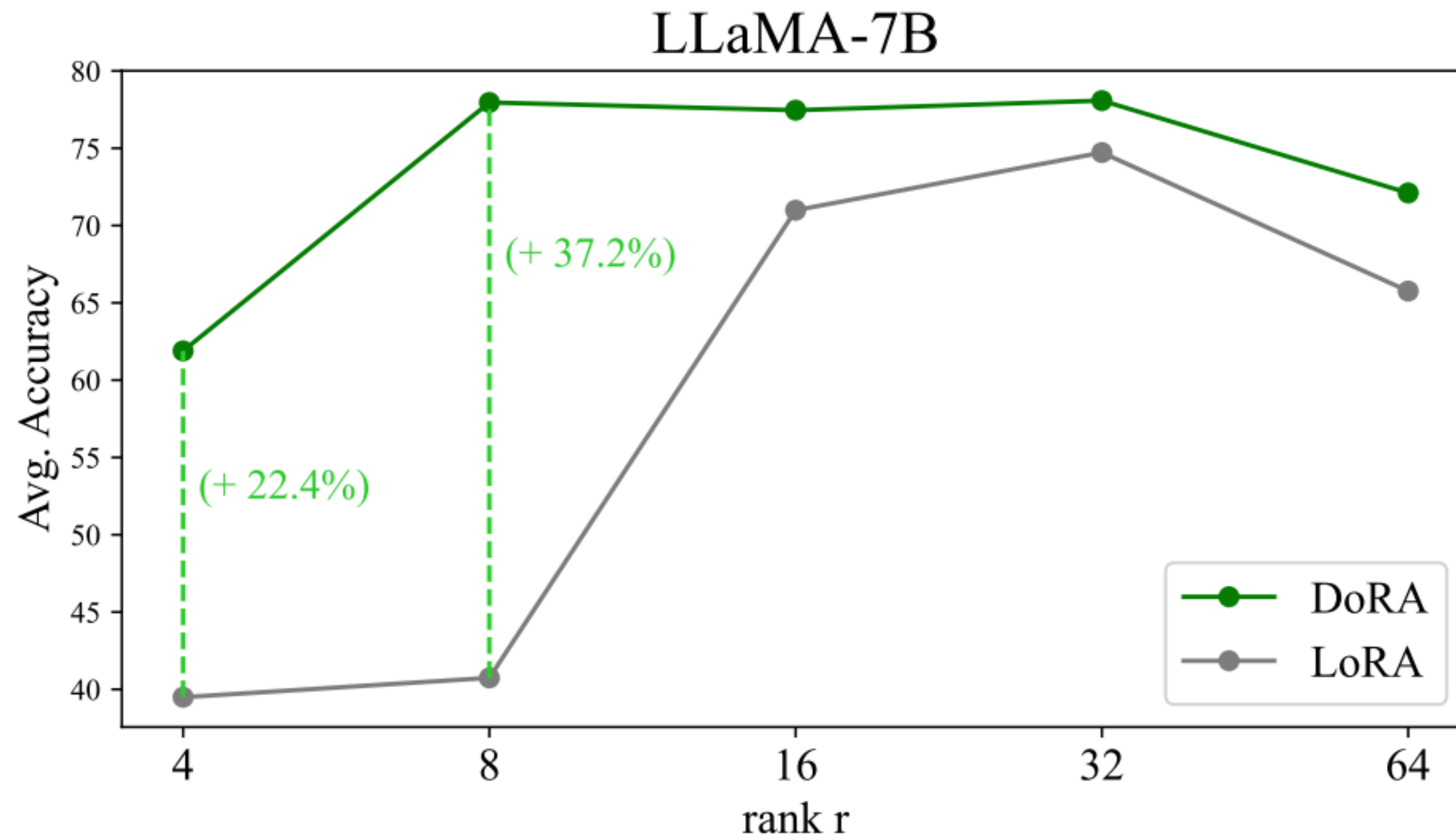
- Outperforms LoRA (on 8 common sense tasks)

Model	PEFT Method	# Params (%)	BoolQ	PIQA	SIQA	HellaSwag	WinoGrande	ARC-e	ARC-c	OBQA	Avg.
ChatGPT	-	-	73.1	85.4	68.5	78.5	66.1	89.8	79.9	74.8	77.0
LLaMA-7B	Prefix	0.11	64.3	76.8	73.9	42.1	72.1	72.9	54.0	60.6	64.6
	Series	0.99	63.0	79.2	76.3	67.9	75.7	74.5	57.1	72.4	70.8
	Parallel	3.54	67.9	76.4	78.8	69.8	78.9	73.7	57.3	75.2	72.2
	LoRA	0.83	68.9	80.7	77.4	78.1	78.8	77.8	61.3	74.8	74.7
	DoRA [†] (Ours)	0.43	70.0	82.6	79.7	83.2	80.6	80.6	65.4	77.6	77.5
	DoRA (Ours)	0.84	69.7	83.4	78.6	87.2	81.0	81.9	66.2	79.2	78.4
LLaMA-13B	Prefix	0.03	65.3	75.4	72.1	55.2	68.6	79.5	62.9	68.0	68.4
	Series	0.80	71.8	83	79.2	88.1	82.4	82.5	67.3	81.8	79.5
	Parallel	2.89	72.5	84.9	79.8	92.1	84.7	84.2	71.2	82.4	81.4
	LoRA	0.67	72.1	83.5	80.5	90.5	83.7	82.8	68.3	82.4	80.5
	DoRA [†] (Ours)	0.35	72.5	85.3	79.9	90.1	82.9	82.7	69.7	83.6	80.8
	DoRA (Ours)	0.68	72.4	84.9	81.5	92.4	84.2	84.2	69.6	82.8	81.5
LLaMA2-7B	LoRA	0.83	69.8	79.9	79.5	83.6	82.6	79.8	64.7	81.0	77.6
	DoRA [†] (Ours)	0.43	72.0	83.1	79.9	89.1	83.0	84.5	71.0	81.2	80.5
	DoRA (Ours)	0.84	71.8	83.7	76.0	89.1	82.6	83.7	68.2	82.4	79.7
LLaMA3-8B	LoRA	0.70	70.8	85.2	79.9	91.7	84.3	84.2	71.2	79.0	80.8
	DoRA [†] (Ours)	0.35	74.5	88.8	80.3	95.5	84.7	90.1	79.1	87.2	85.0
	DoRA (Ours)	0.71	74.6	89.3	79.9	95.5	85.6	90.5	80.4	85.8	85.2

<https://arxiv.org/pdf/2402.09353>

DoRA: Weight decomposed low-rank adaptation

- Performance more robust to rank

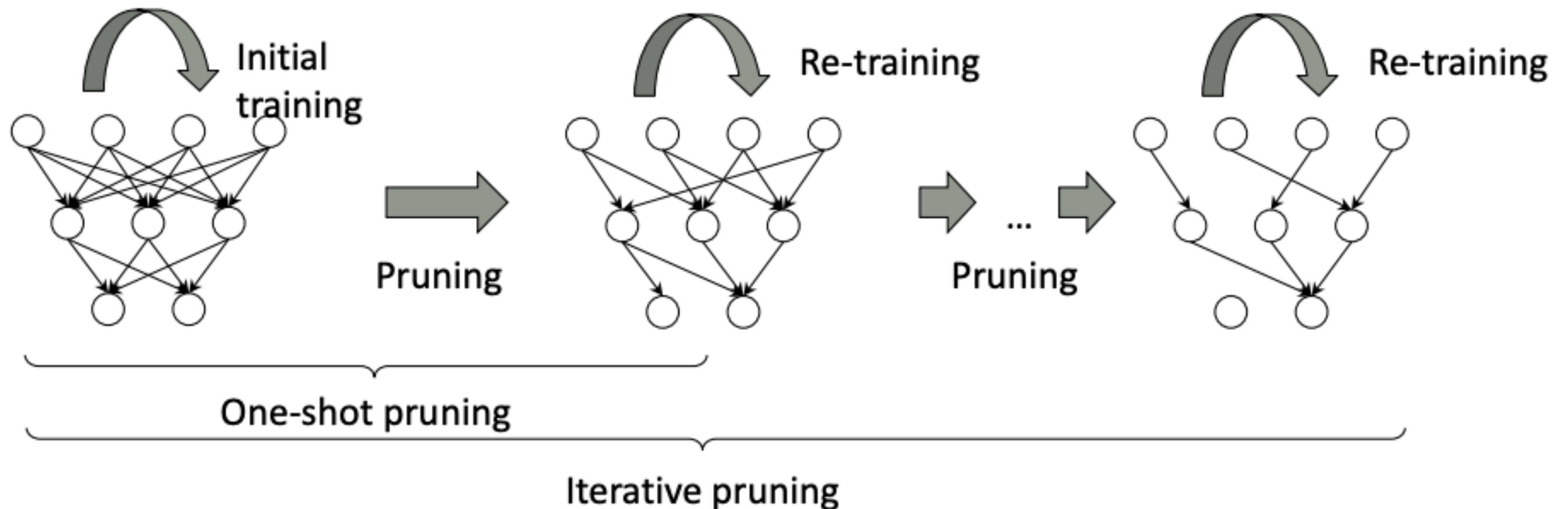


<https://arxiv.org/pdf/2402.09353>

Masked / Sparse fine-tuning

Sparse fine-tuning

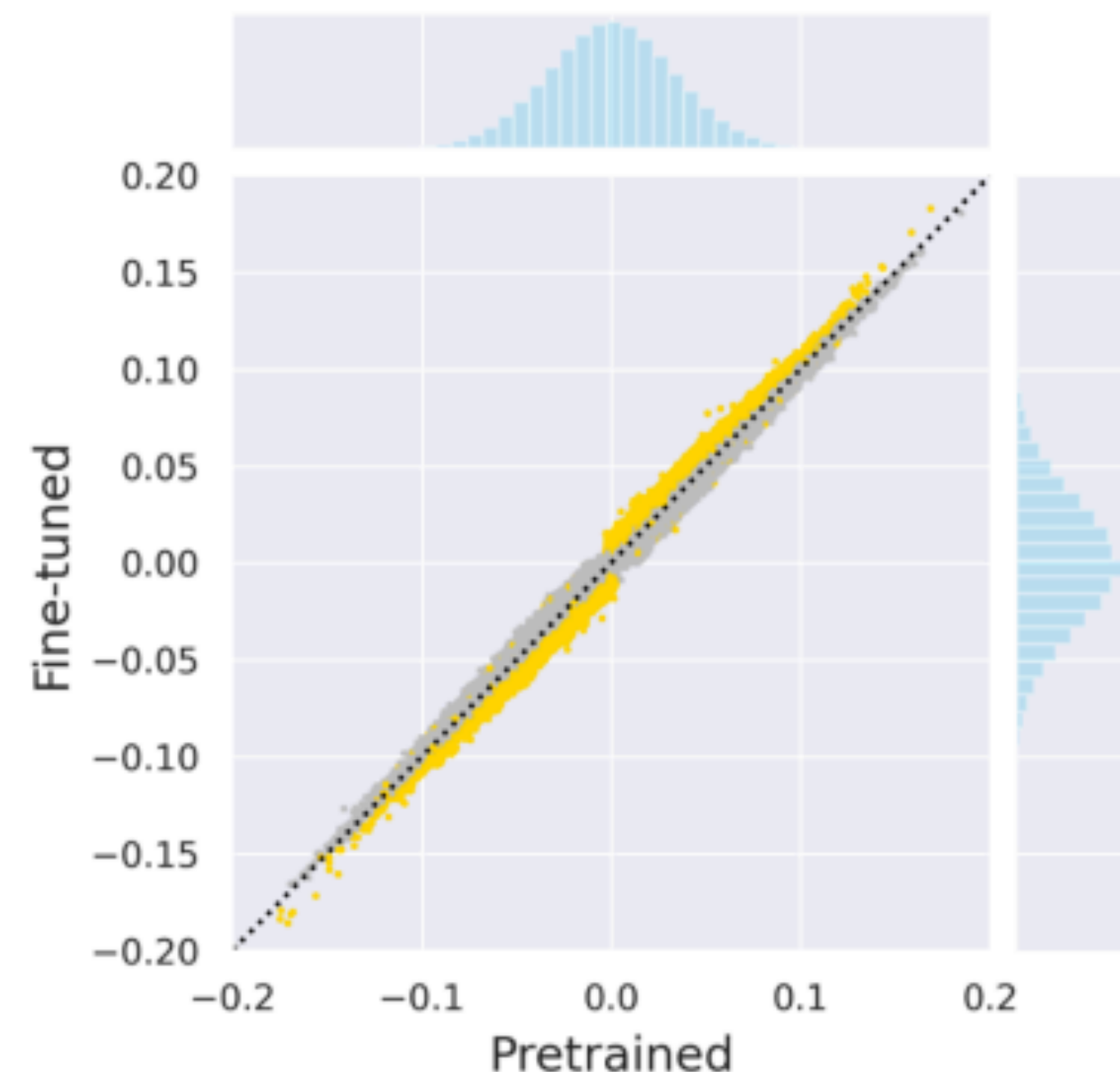
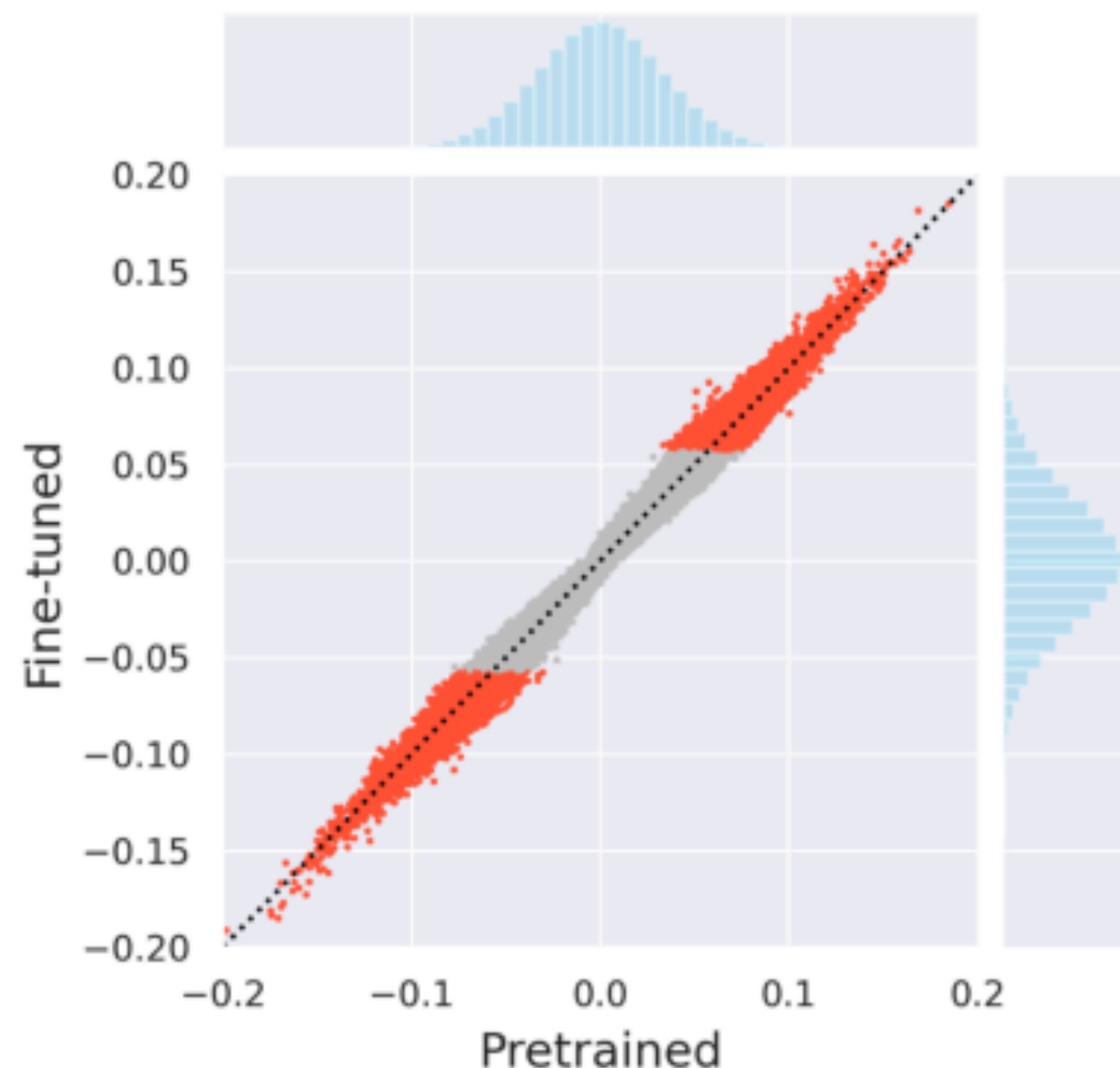
- Use mask to identify a sparse subset of weights and only update a subset of weights during training
- Related to **pruning** (removes subset of weights)
- Can repeat for several iterations



Pruning

- Use mask to prune away weights
- Different ways to pick what weights to keep vs prune
 - For fine-tuning: select weights which when **updated**, impact the model performance the most
 - Magnitude pruning vs Movement pruning

Fine-tuned weights stay close to their pre-trained values. Magnitude pruning (left) selects **weights that are far from 0.**



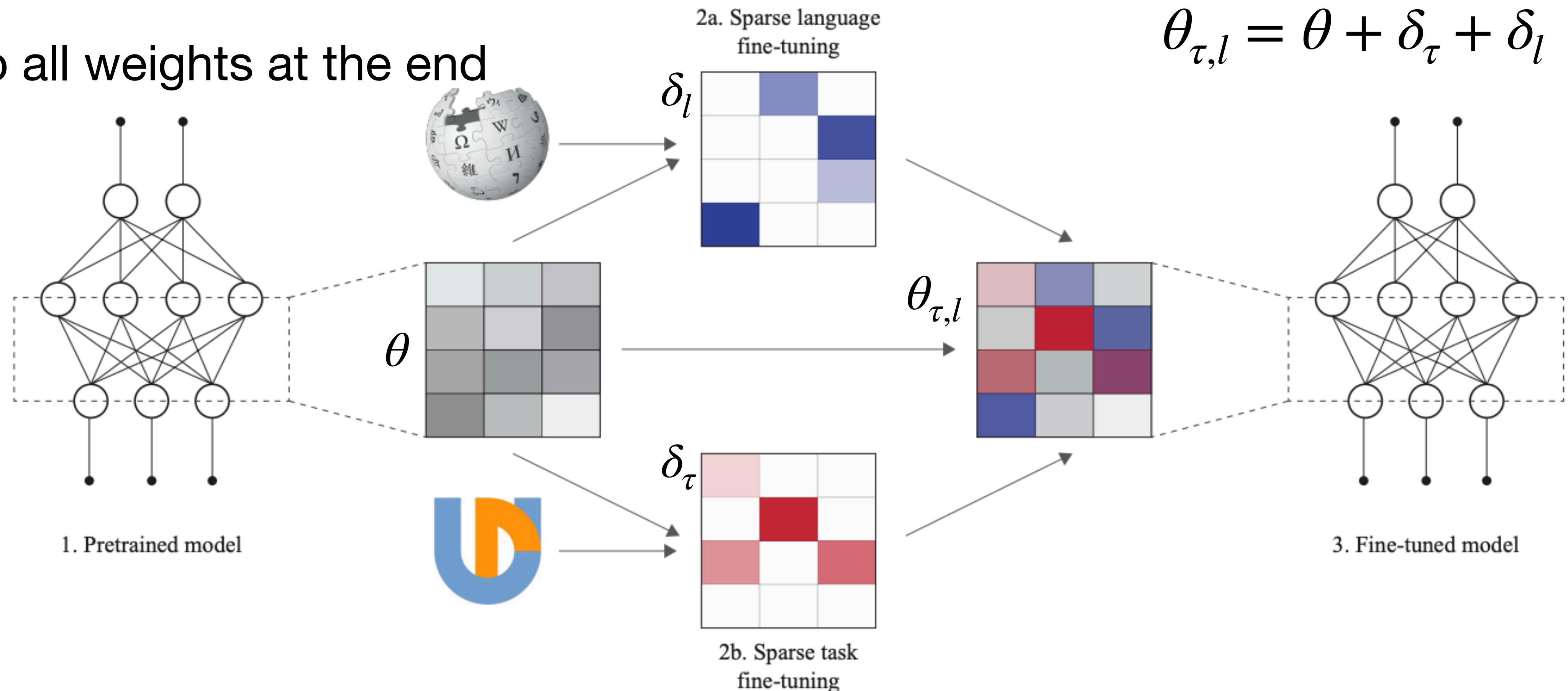
Movement pruning (right) selects **weights that move away from 0.**

Sparse fine-tuning

- Use mask to determine what weights to tune
- Tune weights for specific task τ and language l
- Keep all weights at the end

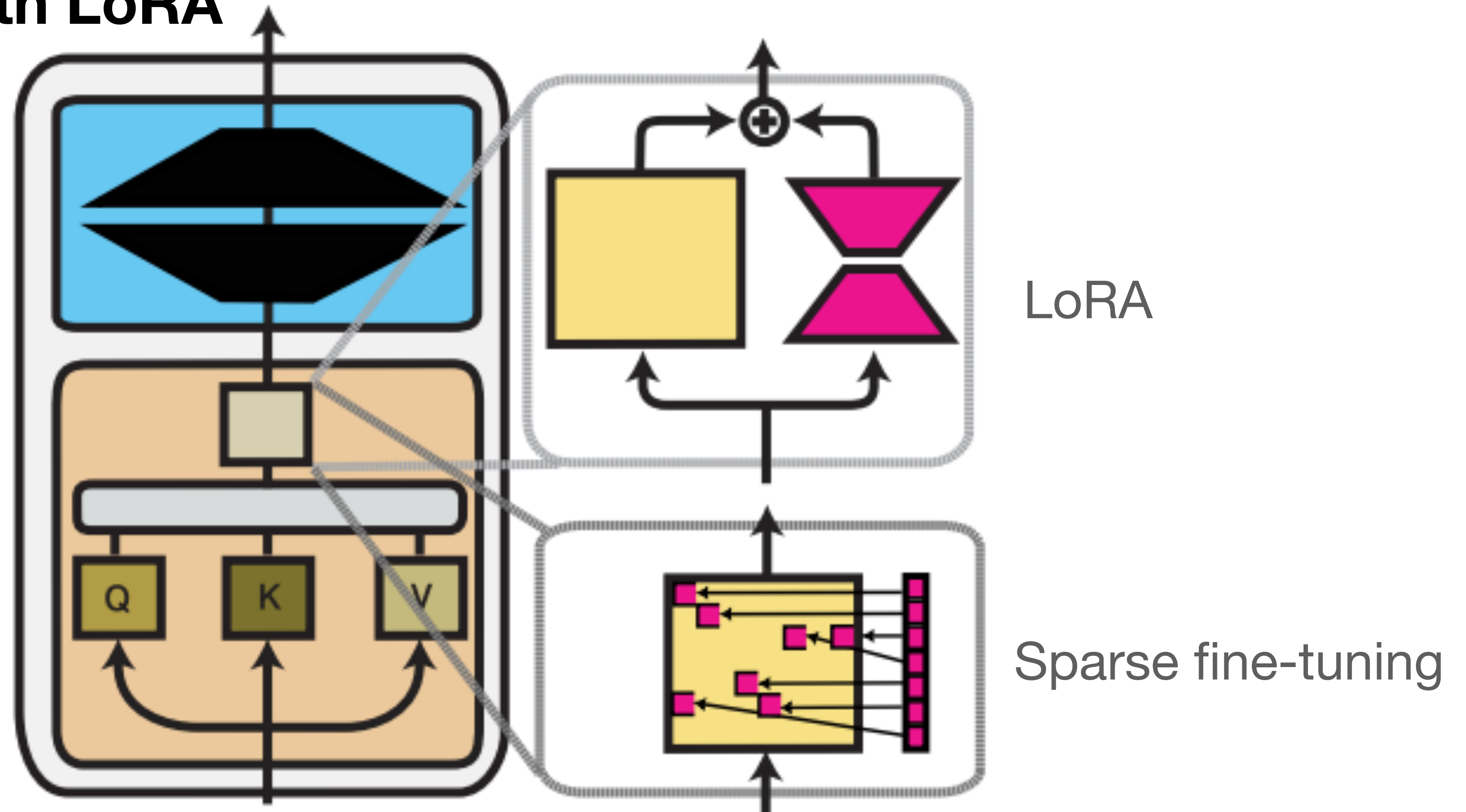
Weights are added to pretrained model

$$\theta_{\tau,l} = \theta + \delta_{\tau} + \delta_l$$



Sparse fine-tuning

Comparison with LoRA

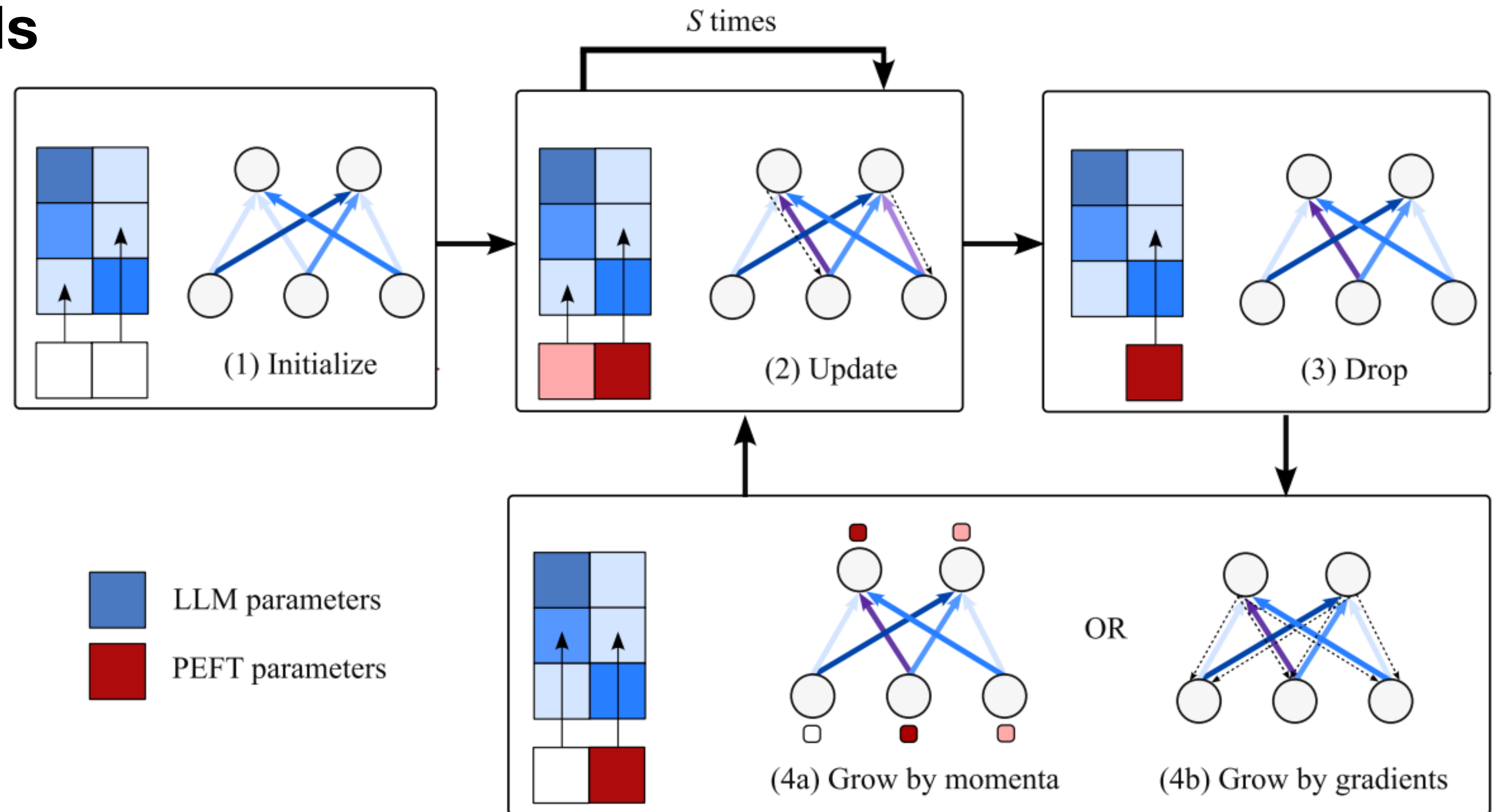


Scaling Sparse Fine-Tuning to Large Language Models
[Ansell et al. 2024] <https://arxiv.org/pdf/2401.16405.pdf>

Sparse fine-tuning

Scaling to LLMs

- Maintain vector of indices η vs dense binary mask
- Allow η to change over time



Scaling Sparse Fine-Tuning to Large Language Models
[Ansell et al. 2024] <https://arxiv.org/pdf/2401.16405.pdf>

Sparse fine-tuning

Scaling to LLMs

		Flan v2		GPT4-Alpaca
Model / Method		MMLU	TyDiQA	HumanEval
Llama2-7b	Original	45.8	50.9	13.5
	FullFT	50.5	55.5	15.2
	(IA) ³	46.7	51.6	14.7
	LoRA	49.3	55.3	15.7
	SpIEL-AG	50.7	56.2	15.6
	SpIEL-MA	48.8	55.8	16.2
Llama2-13b	Original	55.3	60.3	17.8
	FullFT	-	-	-
	(IA) ³	55.1	60.1	18.5
	LoRA	55.8	61.4	19.8
	SpIEL-AG	55.8	62.5	20.0
	SpIEL-MA	55.5	62.5	19.9

Table 3. GPU memory requirements (in GB) and average time per training step (in seconds) for fine-tuning SFT and LoRA on Flan v2 on an A100 GPU. We report values either without (left) or with (right) activation checkpointing.

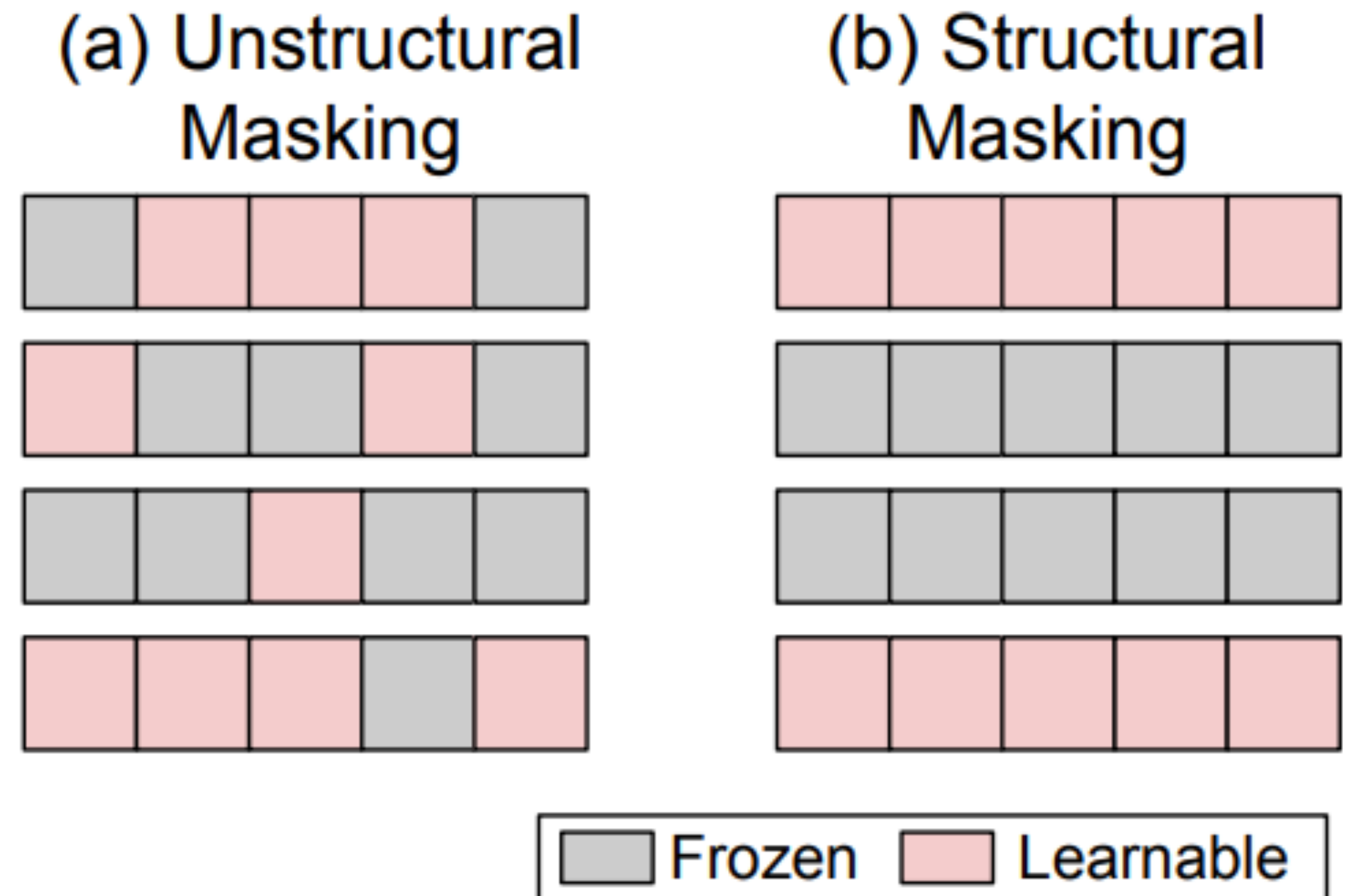
Method	LlaMA 2 7b		LlaMA 2 13b	
	Mem. ↓	Time ↓	Mem. ↓	Time ↓
LoRA	40 / 18	30.5 / 42.5	66 / 31	45.9 / 64.0
SpIEL-AG	34 / 20	33.4 / 44.8	56 / 36	56.2 / 76.3
SpIEL-MA	30 / 17	30.6 / 41.7	51 / 31	50.6 / 70.9
qLoRA	30 / 8	38.5 / 55.2	46 / 12	63.6 / 95.3
qSpIEL-AG	26 / 13	42.8 / 58.4	40 / 19	73.4 / 101.1
qSpIEL-MA	22 / 10	39.6 / 55.5	35 / 14	70.1 / 97.3

Sparse fine-tuning

- LT-SFT (Lottery Ticket) [Ansell et al. ACL 2022]
 - All parameters are first fine-tuned (once), then parameters that changed the most are selected for fine-tuning (multiple languages, tasks)
- FISH-Mask [Sung et al. NeurIPS 2021]
- ChildTuning [Xu et al. EMNLP 2021]
- DiffPruning [Guo et al. ACL 2021]
- BitFit [Zaken et al. ACL 2022]

Sparse fine-tuning

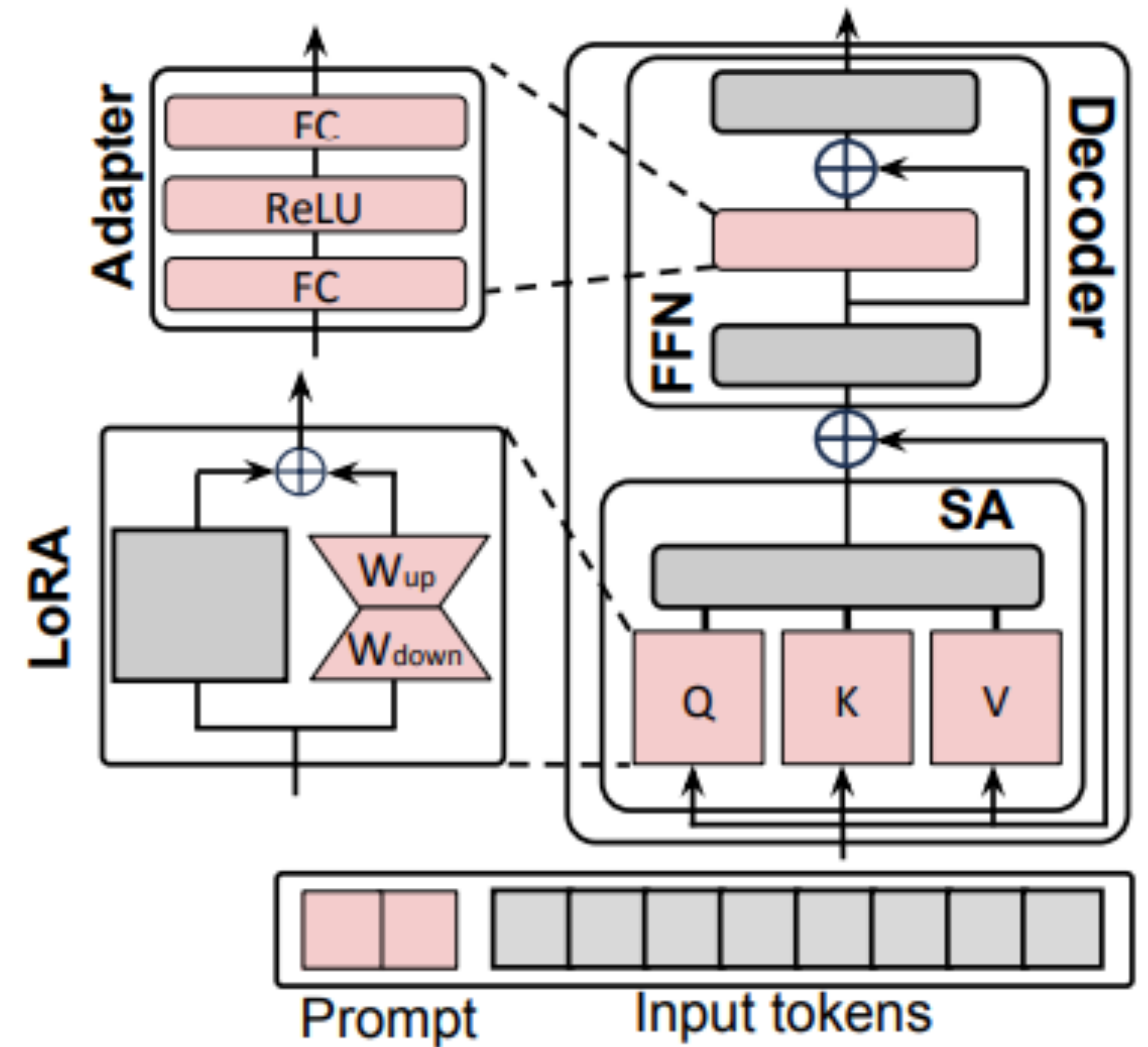
- **Unstructured:** non-zero masks distributed to various positions, inefficient when considering hardware
- **Structured:** organized-regular patterns for masks, better computational and hardware efficiency
 - *Bitfit*: Fine-tune bias parameters (does not handle large models which removes bias parameters)
 - *Xattn*: Fine-tunes cross-attention layers



Summary

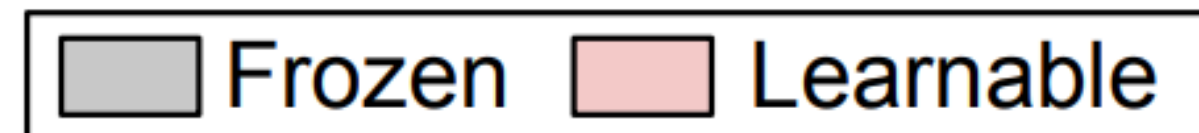
Different approaches to PEFT

- **Input:** Tune the input (prompt tuning)
- **Function:** Insert function into layers of pre-trained model (adapters)
- **Parameter:** Tune subset of parameters (sparse fine-tuning) or delta (LoRA)

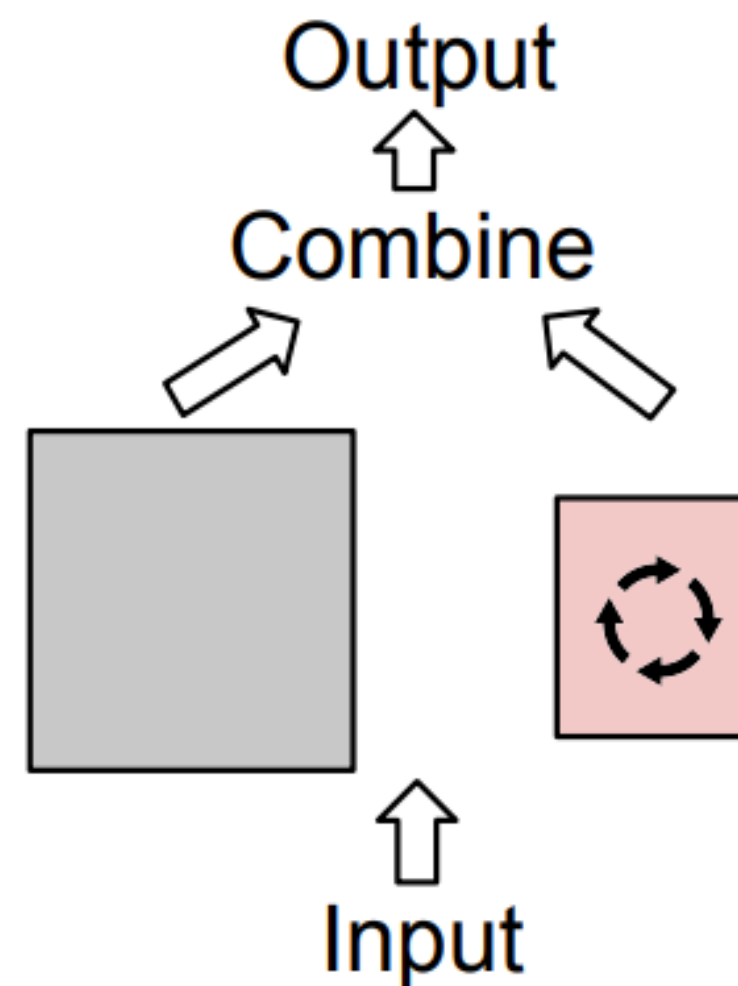


<https://arxiv.org/abs/2403.14608>

Types of PEFT

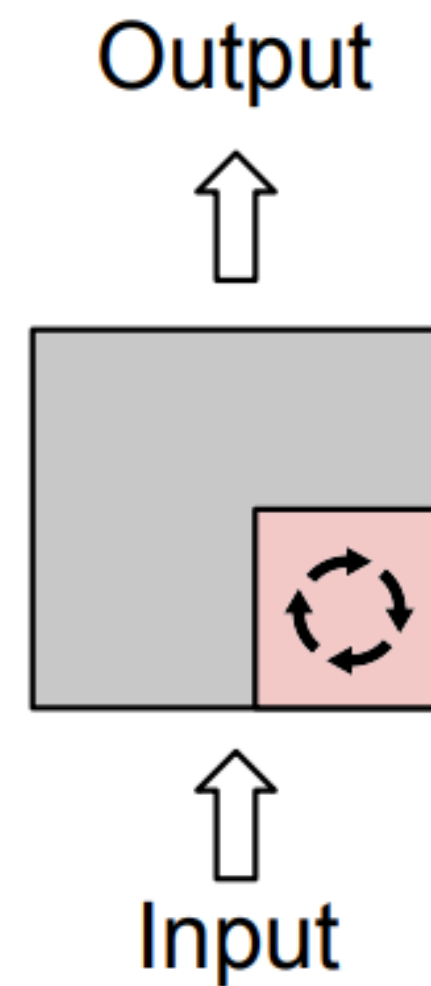


(a) Additive PEFT



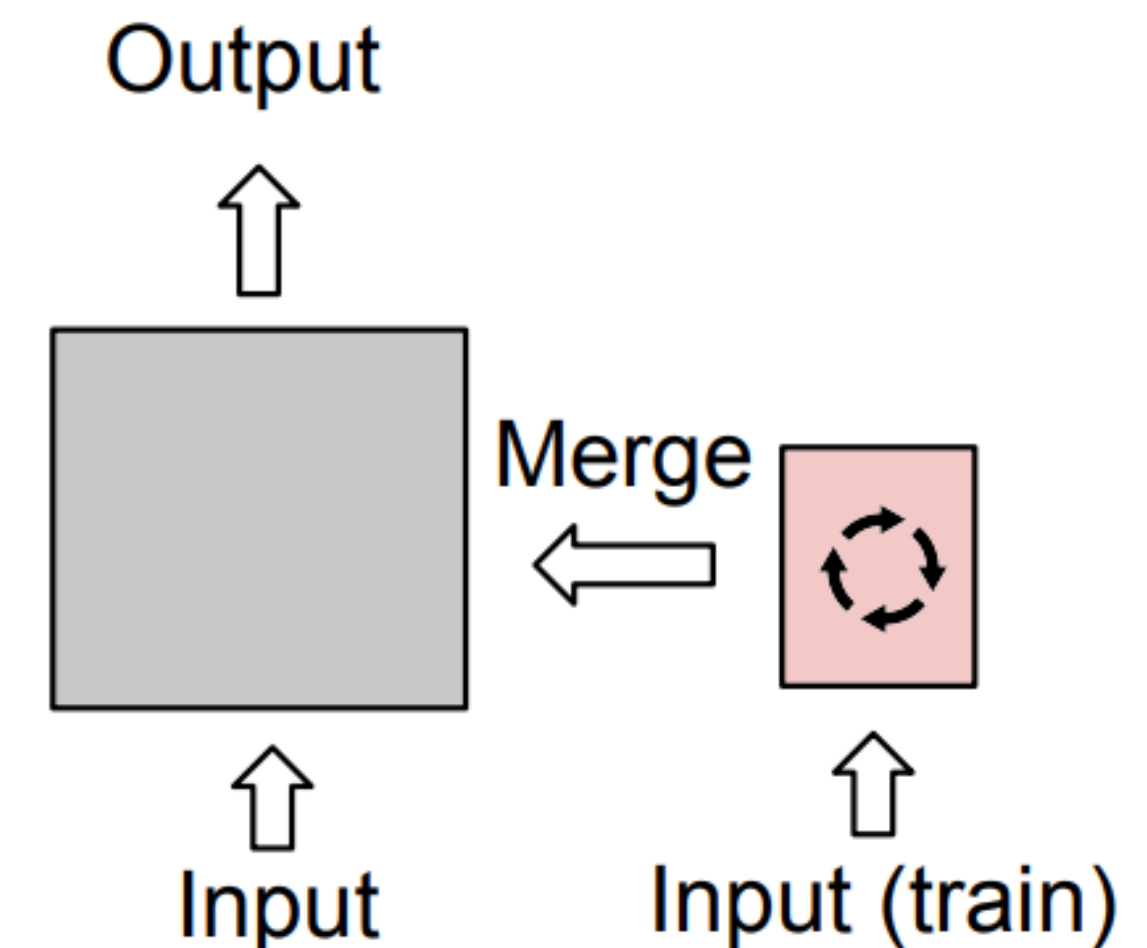
Additional
parameters
(Adapters, prefix tuning)

(b) Selective PEFT



Subset of existing
parameters
(masked / sparse FT)

(c) Reparameterization PEFT



Reparameterization
(LoRA)