Parameter Efficient Fine-Tuning (PEFT)

Anastasia Ianina

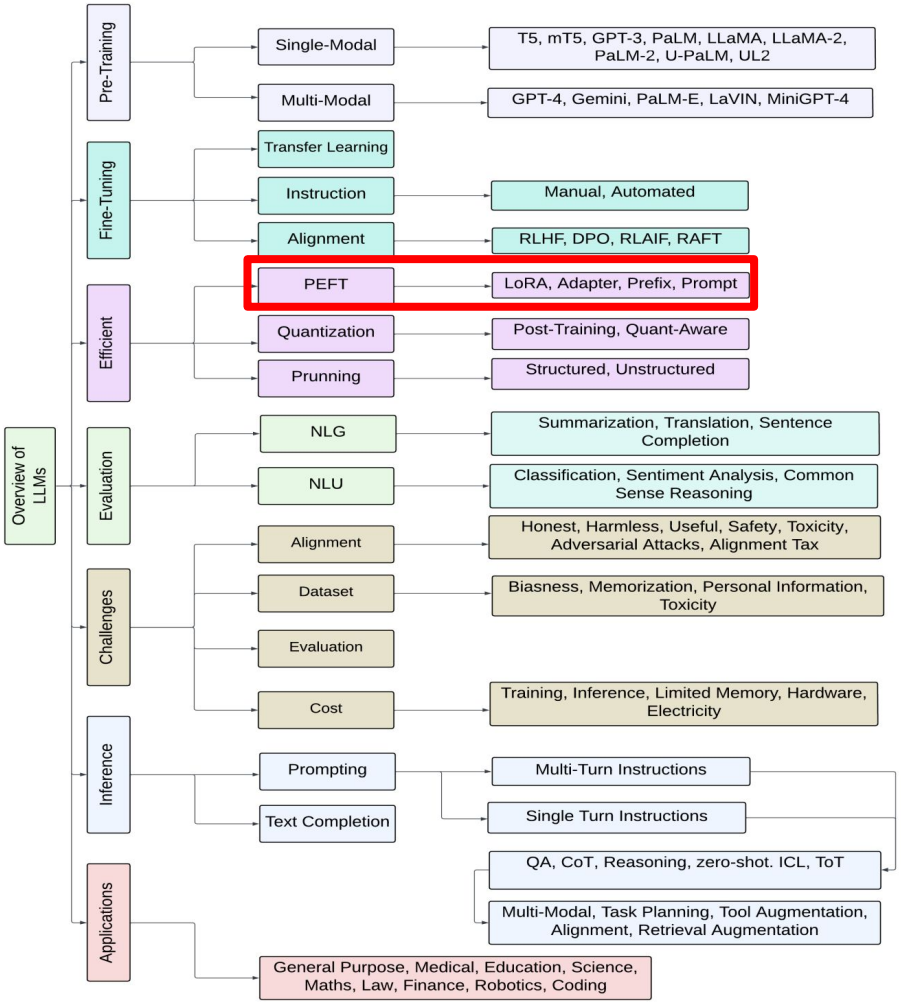
Harbour Space

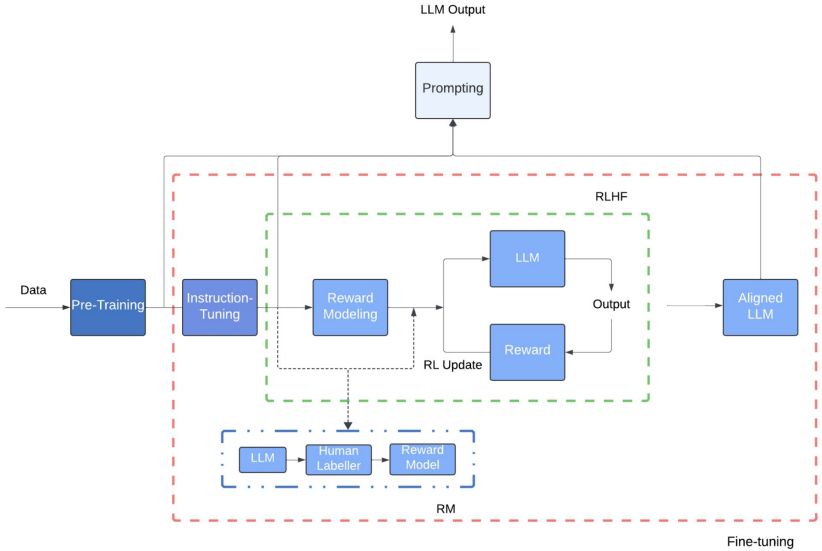
Outline

1. Supervised fine-tuning (SFT)
2. Parameter efficient fine-tuning (PEFT)
3. Adapters, IA3, BitFit, etc.
4. Prompt tuning, p-tuning, prefix tuning
5. LoRA, KronA, etc.
6. How to choose the right PEFT?
7. LLM training tips

Based on:
YSDA NLP course
A Guide to Parameter-Efficient Fine-Tuning by Vlad Lialin (MunichNLP)

A Comprehensive Overview of Large Language Models





Supervised fine-tuning

SFT

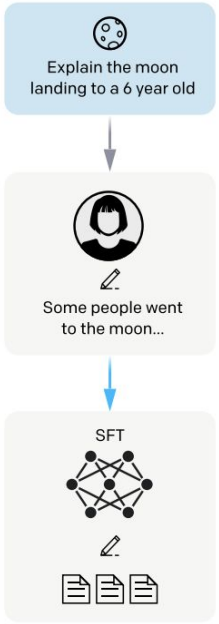
Step 1

Collect demonstration data, and train a supervised policy.

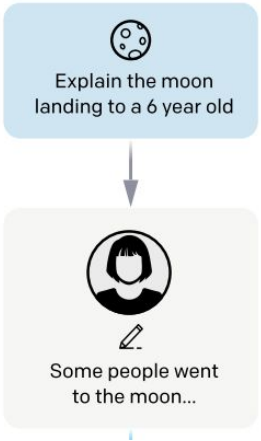
A prompt is sampled from our prompt dataset.

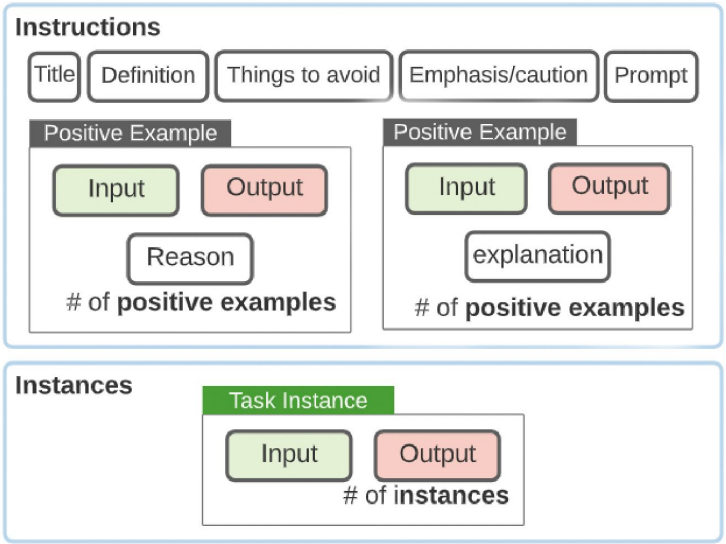
A labeler demonstrates the desired output behavior.

This data is used to fine-tune GPT-3 with supervised learning.



1. Prepare dataset for tuning
2. Train the whole model with the selected loss
   a. No frozen layers
   b. Training procedure is close to pre-train but with less data





SFT

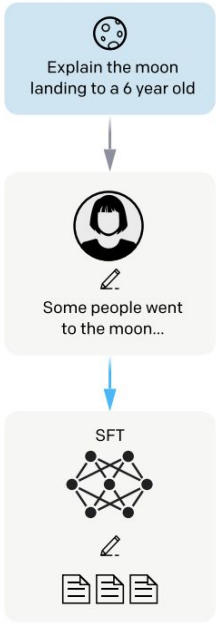
Step 1

Collect demonstration data, and train a supervised policy.

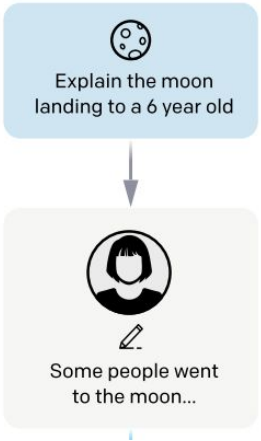
A prompt is sampled from our prompt dataset.

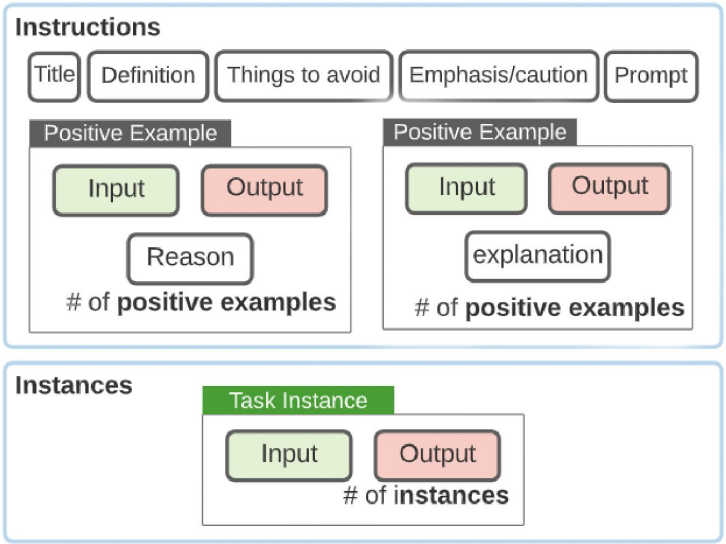
A labeler demonstrates the desired output behavior.

This data is used to fine-tune GPT-3 with supervised learning.



1. Prepare dataset for tuning
2. Train the whole model with the selected loss
   a. No frozen layers
   b. Training procedure is close to pre-train but with less data





Parameter Efficient Fine-Tuning

GPU memory breakdown

		OPT-1.3B, 16-bit float, seq 512
cuDNN and CUDA		~1Gb
Model weights	size(float) * N	2.6Gb
Gradients	size(float) * $N_{trainable}$	2.6Gb
Hidden states	*~size(float) L (20 h seq + 3 seq^2)*	1Gb per example
Optimizer states	2 * size(float) * $N_{trainable}$	5.2Gb
(maybe) fp32 copy of the gradients	4 * $N_{trainable}$	10.2Gb

Estimate: 12.4Gb, actual: 11.0Gb *(after empty_cache)*

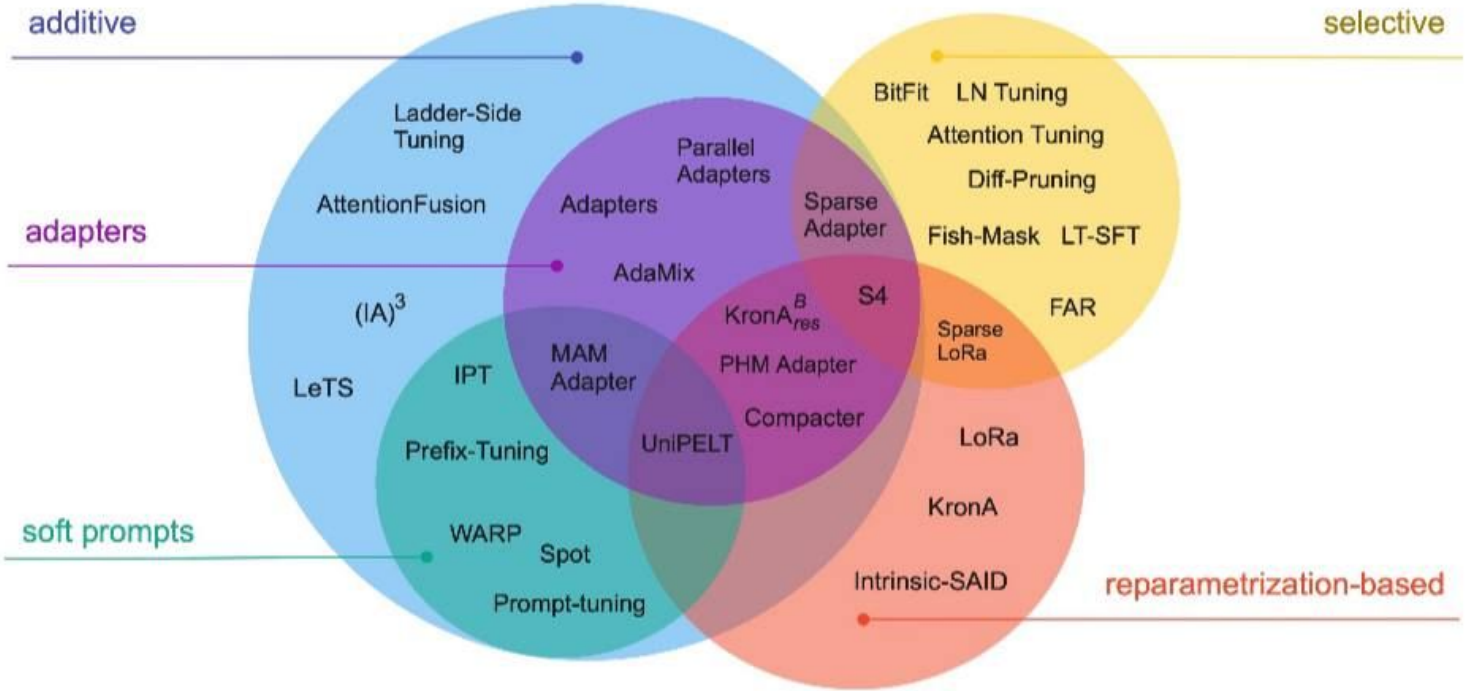
GPU memory breakdown

Training only 0.2M parameters

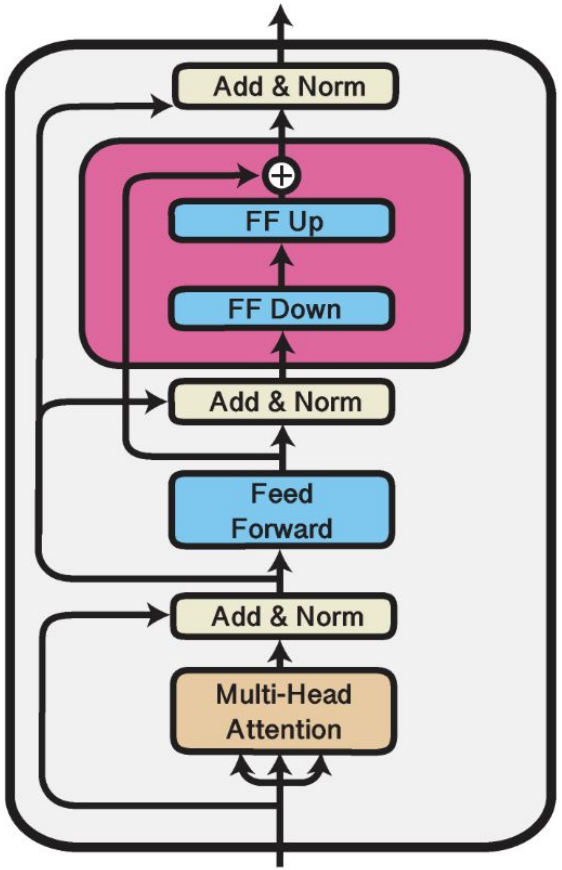
		OPT-1.3B, 16-bit float, seq 512
cuDNN and CUDA		~1Gb
Model weights	size(float) * N	2.6Gb
Gradients	size(float) * $N_{trainable}$	**0.4Mb**
Hidden states	*~size(float) L (20 h seq + 3 seq^2)*	1Gb per example
Optimizer states	2 * size(float) * $N_{trainable}$	0.8Mb
(maybe) fp32 copy of the gradients	4 * $N_{trainable}$	**1.6Mb**

Estimate: 4.6Gb, actual: **5.7Gb** *(after empty_cache)*

PEFT


additive
selective
adapters
soft prompts
reparametrization-based
Ladder-Side Tuning
AttentionFusion
Parallel Adapters
Adapters
Sparse Adapter
AdaMix
(IA)3
KronA_res^B
S4
BitFit
LN Tuning
Attention Tuning
Diff-Pruning
Fish-Mask
LT-SFT
FAR
Sparse LoRa
LeTS
IPT
MAM Adapter
PHM Adapter
Compacter
UniPELT
Prefix-Tuning
LoRa
KronA
WARP
Spot
Prompt-tuning
Intrinsic-SAID

Additive PEFT: Adapters

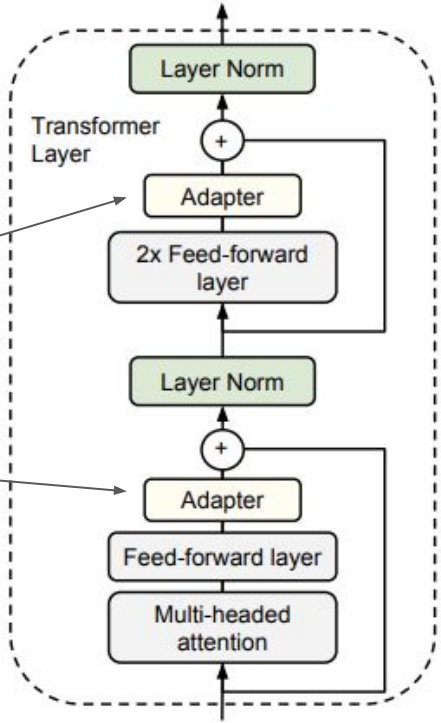


One-sentence idea:

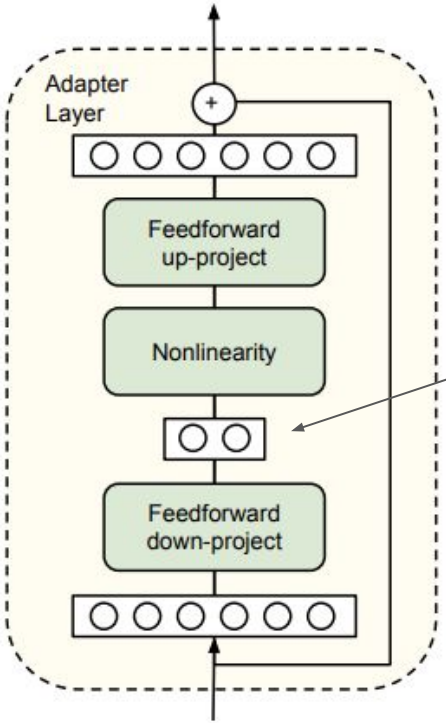
Add fully-connected networks after attention and FFN layers

```
def transformer_block_with_adapter(x):
    residual = x
    x = SelfAttention(x)
    x = FFN(x)   # adapter
    x = LN(x + residual)
    residual = x
    x = FFN(x)   # transformer FFN
    x = FFN(x)   # adapter
    x = LN(x + residual)
    return x
```

Additive PEFT: Adapters



Tune only these layers



Dimensionality is smaller than initial layer

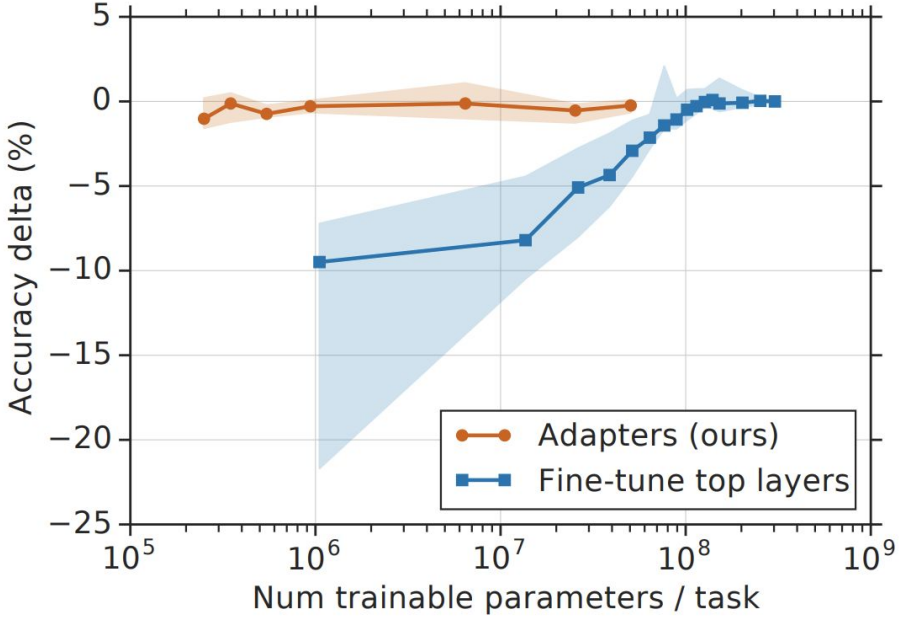
Adapters vs. fine-tune top layers

Why fine-tuning top layers is not perfect?

- Fine-tuning just top layers may lead to overfitting
- Frozen layers do not adapt to new information

Why adapters are good?

- Fast and effective training
- Adapters trained on different tasks can be combined

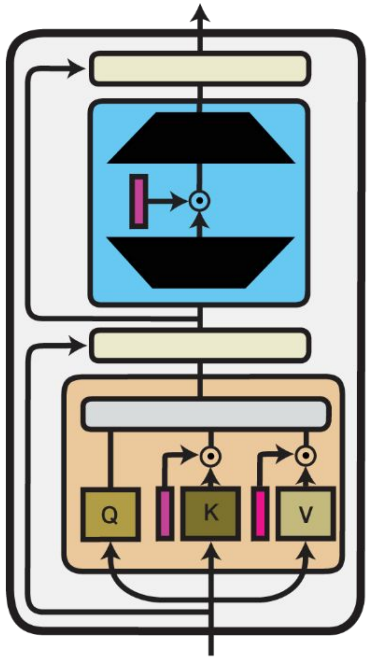


Additive PEFT: IA3 (T-Few)

One-sentence idea:

Rescale key, value, and hidden FFN activations

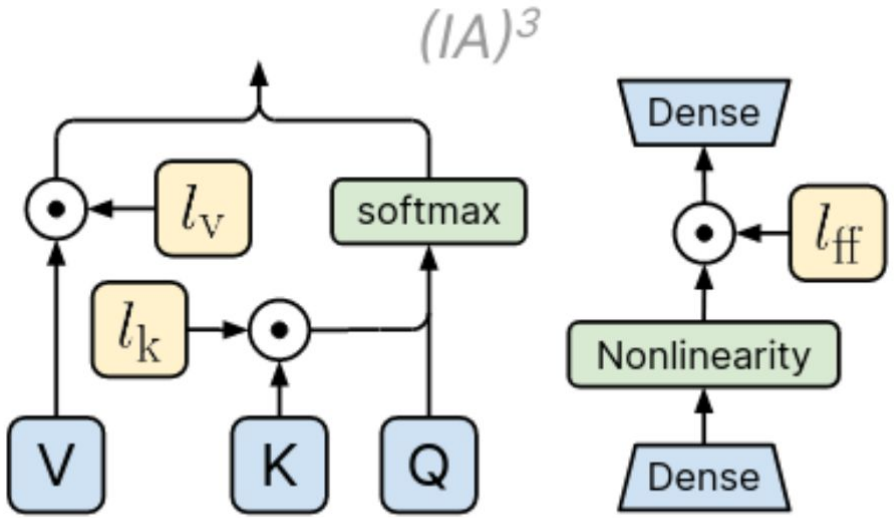
```
def transformer_block_with_ia3(x):
    residual = x
    x = ia3_self_attention(x)
    x = LN(x + residual)
    residual = x
    x = x @ W_1          # FFN in
    x = l_ff * gelu(x)   # (IA)3 scaling
    x = x @ W_2          # FFN out
    x = LN(x + residual)
    return x

def ia3_self_attention(x):
    k, q, v = x @ W_k, x @ W_q, x @ W_v
    k = l_k * k
    v = l_v * v
    return softmax(q @ k.T) @ V
```



https://arxiv.org/abs/2205.05638

Additive PEFT: IA3 (T-Few)

Method	Inference FLOPs	Training FLOPs	Disk space	Acc.
T-Few	1.1e12	2.7e16	4.2 MB	72.4%
T0 [1]	1.1e12	0	0 B	66.9%
T5+LM [14]	4.5e13	0	16 kB	49.6%
GPT-3 6.7B [4]	5.4e13	0	16 kB	57.2%
GPT-3 13B [4]	1.0e14	0	16 kB	60.3%
GPT-3 175B [4]	1.4e15	0	16 kB	66.6%



https://arxiv.org/abs/2205.05638

Selective PEFT: BitFit

One-sentence idea:

fine-tune only model biases

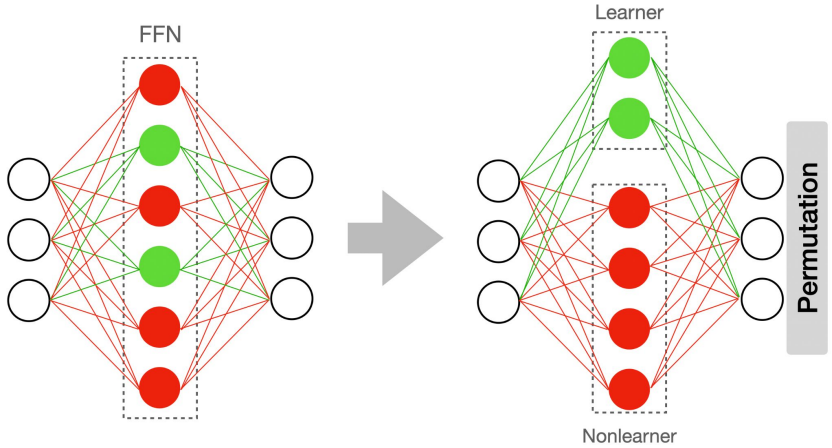
```
params = (p for n, p
          in model.named_parameters()
          if "bias" in n)
optimizer = Optimizer(params)
```

https://arxiv.org/abs/2106.10199

Selective PEFT: Freeze and Reconfigure

One-sentence idea:
selects columns of parameter matrices to train and reconfigures linear layers into trainable and frozen

```
def far_layer(x):
    h1 = x @ W_t
    h2 = x @ W_f
    return concat([h1, h2], dim=-1)
```



https://arxiv.org/abs/2205.01541

Prompt tuning

Prompting

A general prompt consists of:

- Input data that needs to be processed
- Examples of inputs and outputs
- Instructions
- Indicators

Model Input

Instruction: Write the words of the following sentence in reverse order.

Input: The dog crossed the road

Model Output

Output: road the crossed dog the

Model Input

Instruction: Reformat the following comma-separated list of names in <last name>, <first name> format separated with semicolons.

Input: Cameron Wolfe, John Doe

Model Output

Output: Wolfe, Cameron; Doe, John

Indicators are highlighted in red

Model Input

Q: What is 10 + 5?

A: 15

Q: What is 3 + 3?

A: 6

Q: What is 10 + 1?

A:

Model Input

Instruction

Answer all questions by responding with another question in the style of Shakespeare.

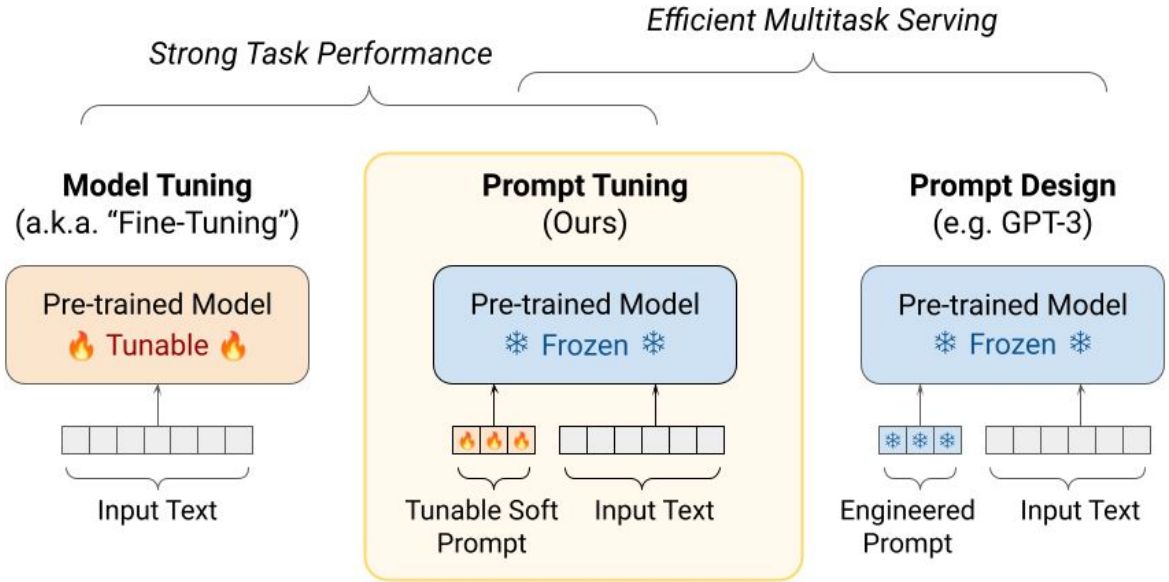
Question

What is the meaning of life?

Response

Prompt tuning

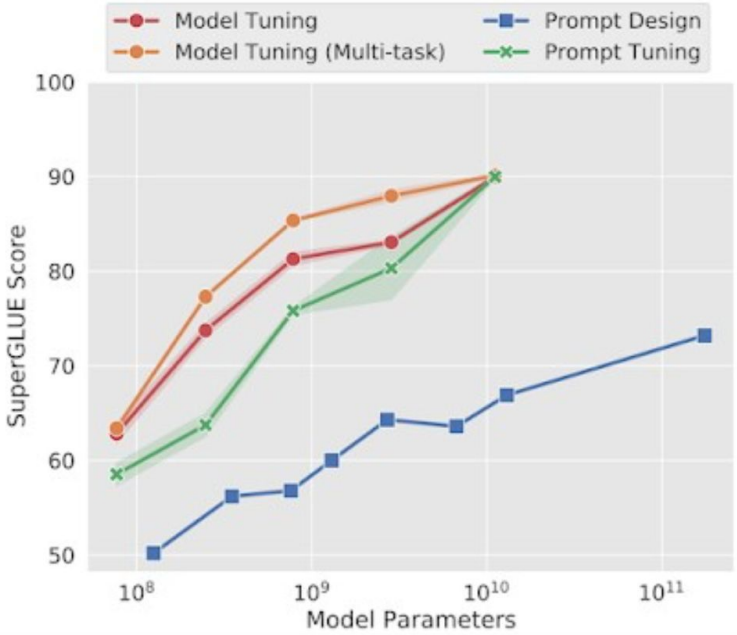
- Model tuning requires making a task specific copy of the entire pre-trained model for each downstream task and inference must be performed in separate batches
- Prompt tuning only requires storing a small task-specific prompt for each task, and enables mixed-task inference using the original pretrained model.



[The Power of Scale for Parameter-Efficient Prompt Tuning]

Prompt tuning

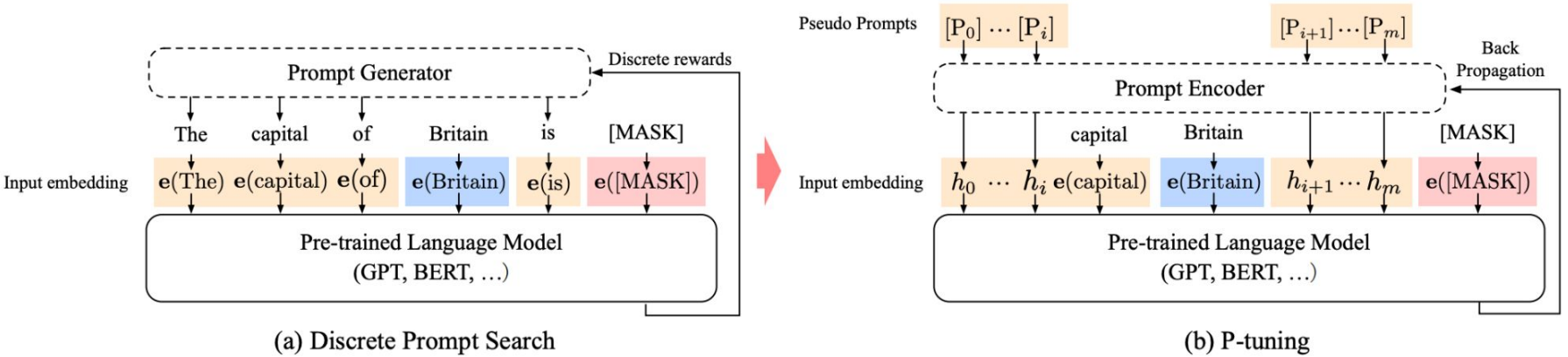
- Prompt tuning becomes more competitive at scale
- Prompt tuning tends to give stronger zero-shot performance than model tuning, especially on datasets with large domain shifts like TextbookQA



Dataset	Domain	Model	Prompt	Δ
SQuAD	Wiki	94.9 ±0.2	94.8 ±0.1	−0.1
TextbookQA	Book	54.3 ±3.7	**66.8** ±2.9	+12.5
BioASQ	Bio	77.9 ±0.4	**79.1** ±0.3	+1.2
RACE	Exam	59.8 ±0.6	**60.7** ±0.5	+0.9
RE	Wiki	88.4 ±0.1	**88.8** ±0.2	+0.4
DuoRC	Movie	**68.9** ±0.7	67.7 ±1.1	−1.2
DROP	Wiki	**68.9** ±1.7	67.1 ±1.9	−1.8

The Power of Scale for Parameter-Efficient Prompt Tuning

P-tuning



- One more encoder added and its embeddings are optimized via gradient descent
- P-tuning works better than few-shot, Prompt Encoder is able to find nearly every manually generated prompt
- In most situations P-tuning does not generate artifacts like few-shot does
- P-tuning is faster than model tuning

[GPT understands, too]

P-tuning

Prompt	P@1 w/o PT	P@1 w/ PT
[X] is located in [Y]. *(original)*	31.3	57.8
[X] is located in which country or state? [Y].	19.8	57.8
[X] is located in which country? [Y].	31.4	58.1
[X] is located in which country? In [Y].	51.1	58.1

Table 1: Discrete prompts suffer from instability (high variance), while P-Tuning stabilizes and improves performance. Results are precision@1 on LAMA-TREx P17 with BERT-base-cased. "PT" refers to P-Tuning, which trains additional continuous prompts in concatenation with discrete prompts.

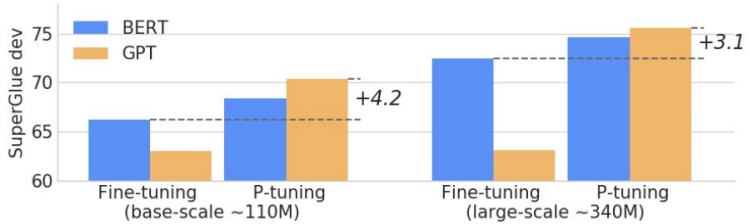


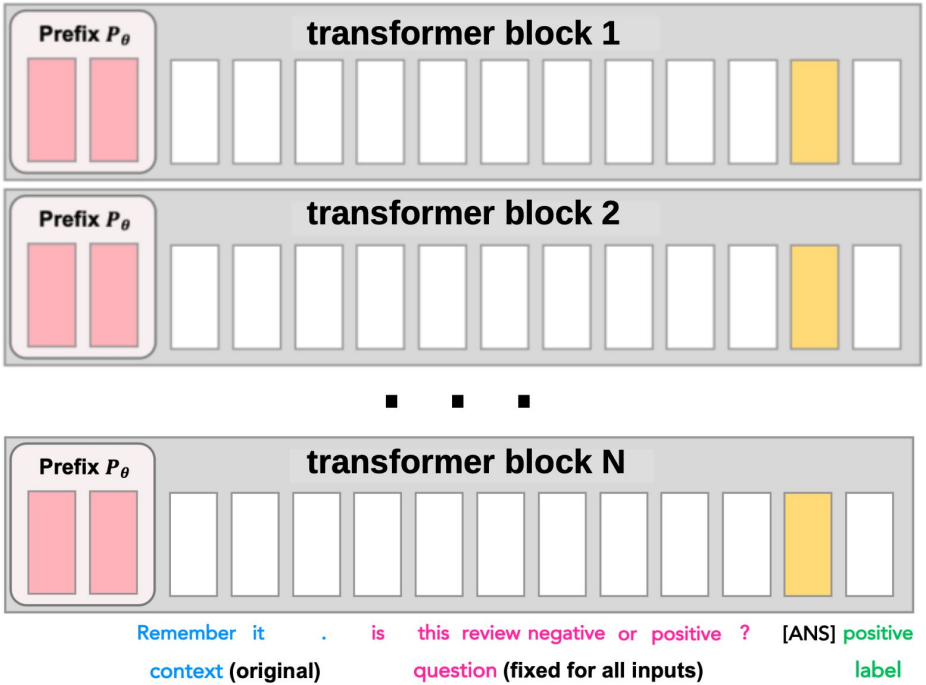
Figure 1: Average scores on 7 dev datasets of SuperGLUE using P-Tuning.

Prompt type	Model	P@1
Original (MP)	BERT-base	31.1
	BERT-large	32.3
	E-BERT	36.2
Discrete	LPAQA (BERT-base)	34.1
	LPAQA (BERT-large)	39.4
	AutoPrompt (BERT-base)	43.3
P-tuning	BERT-base	48.3
	BERT-large	**50.6**

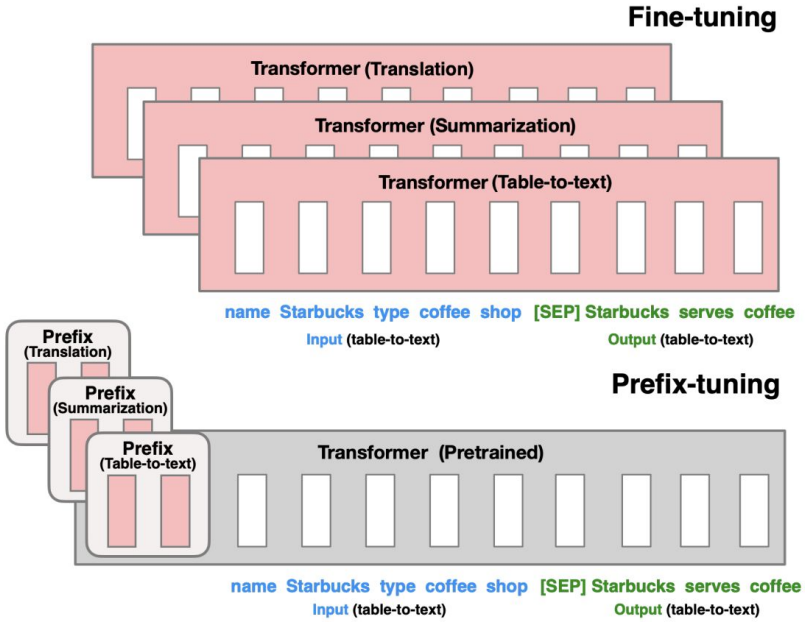
Model	MP	P-tuning
BERT-base (109M)	31.7	52.3 (+20.6)
-AutoPrompt (Shin et al., 2020)	-	45.2
BERT-large (335M)	33.5	54.6 (+21.1)
RoBERTa-base (125M)	18.4	49.3 (+30.9)
-AutoPrompt (Shin et al., 2020)	-	40.0
RoBERTa-large (355M)	22.1	53.5 (+31.4)
GPT2-medium (345M)	20.3	46.5 (+26.2)
GPT2-xl (1.5B)	22.8	54.4 (+31.6)
MegatronLM (11B)	23.1	**64.2** (+41.1)

GPT understands, too

Prefix tuning

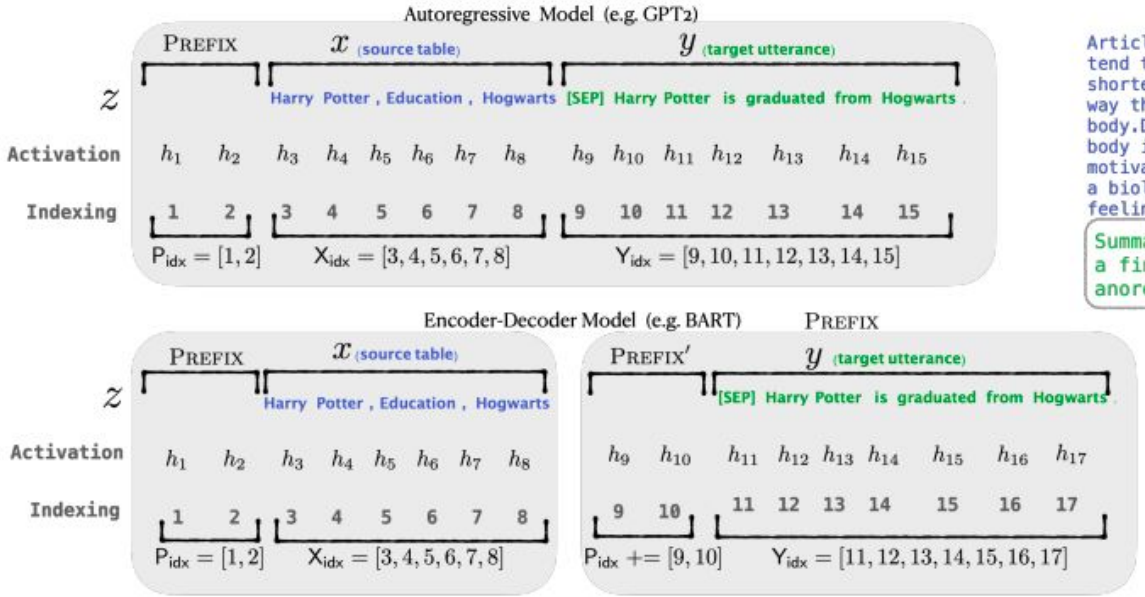


One-sentence idea: fine-tune part of your attention input



[Prefix-Tuning: Optimizing Continuous Prompts for Generation]

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

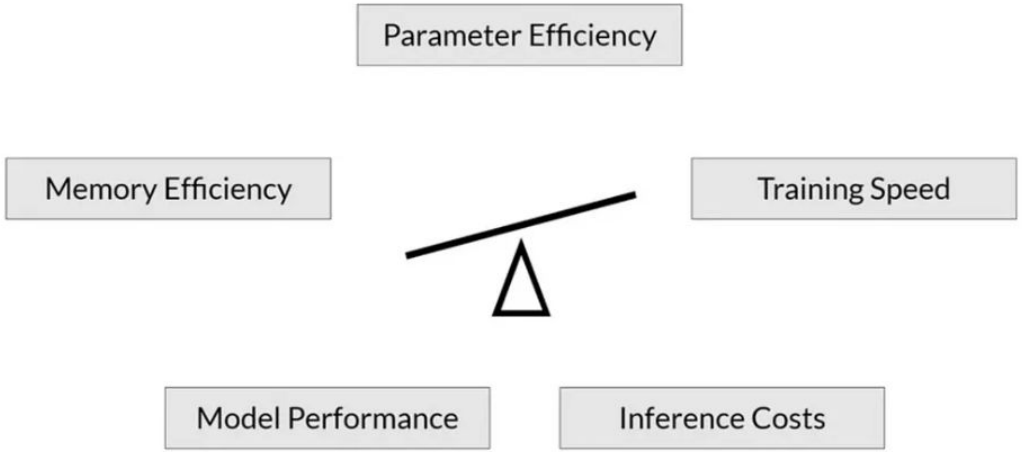
```
Table:  name[Clowns] customer-
rating[1 out of 5]  eatType[coffee
shop] food[Chinese] area[riverside]
near[Clare Hall]
```

```
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 .
```

Layers of prefixes are processed one by one which leads to training and inference deceleration

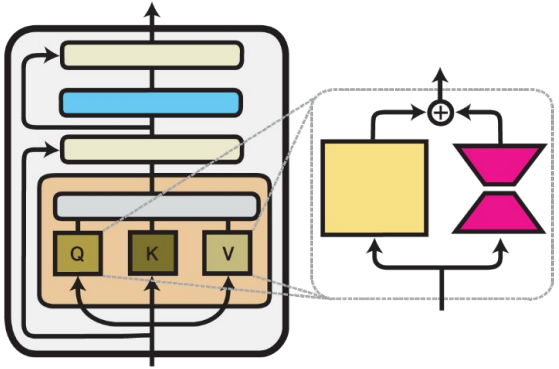
[Prefix-Tuning: Optimizing Continuous Prompts for Generation]

PEFT:
Reparametrization-based methods



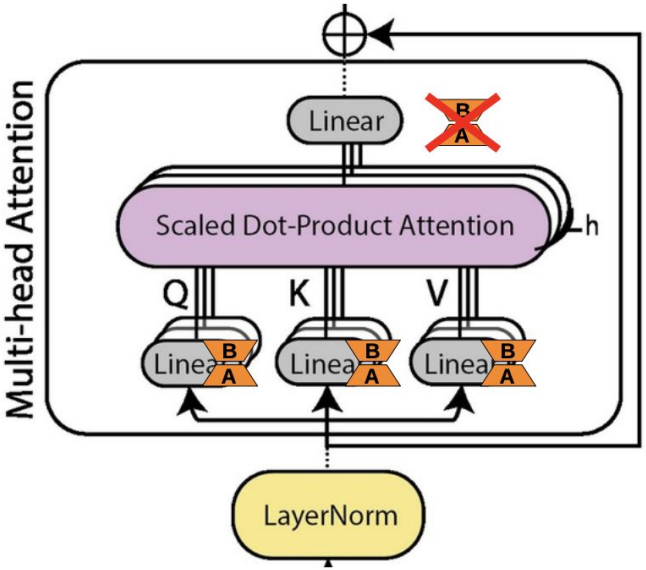
LoRA

One-sentence idea:
Parameter update for a weight matrix in LoRA is decomposed into a product of two low-rank matrices



- Add adapters in parallel with linear layers
- Originally LoRA adapters are added to Q, K & V (
  to MLP, embeddings, logits, etc.)

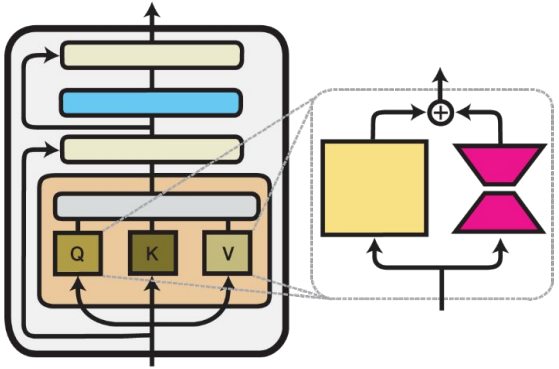
```
def lora_linear(x):
    h = x @ W # regular linear
    h += x @ W_A @ W_B # low-rank update
    return scale * h
```



https://arxiv.org/abs/2106.09685

LoRA

One-sentence idea:
Parameter update for a weight matrix in LoRA is decomposed into a product of two low-rank matrices

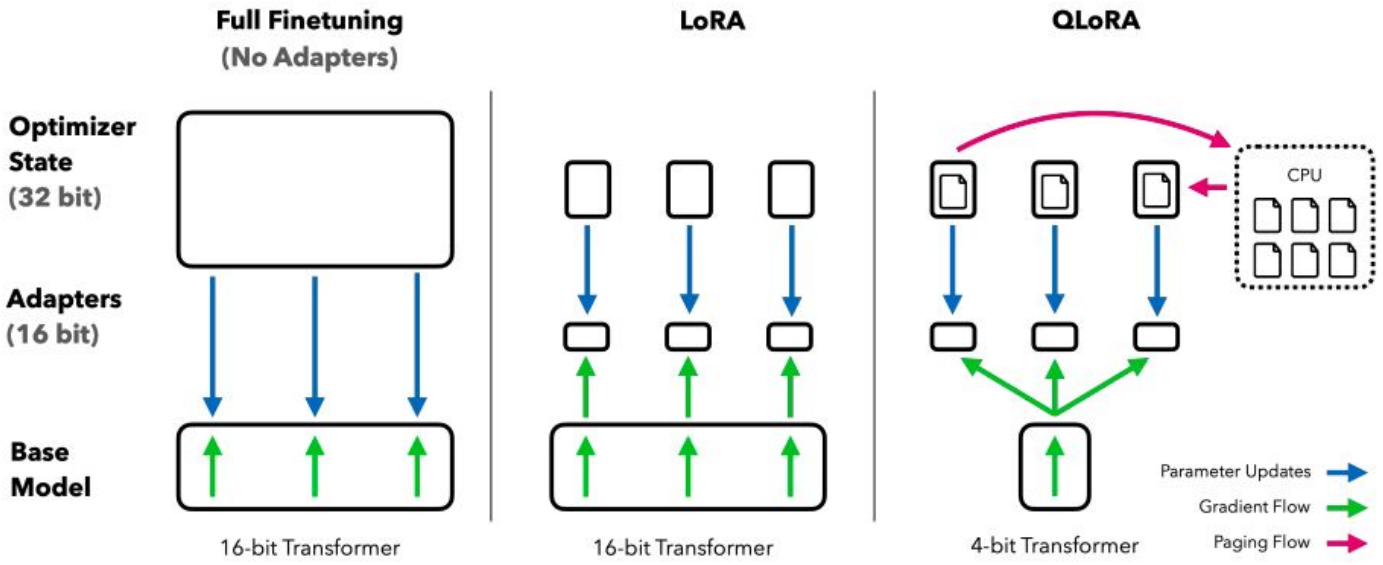


Model&Method	# Trainable Parameters	WikiSQL Acc. (%)	MNLI-m Acc. (%)	SAMSum R1/R2/RL
GPT-3 (FT)	175,255.8M	**73.8**	89.5	52.0/28.0/44.5
GPT-3 (BitFit)	14.2M	71.3	91.0	51.3/27.4/43.5
GPT-3 (PreEmbed)	3.2M	63.1	88.6	48.3/24.2/40.5
GPT-3 (PreLayer)	20.2M	70.1	89.5	50.8/27.3/43.5
GPT-3 ($\text{Adapter}^{\text{H}}$)	7.1M	71.9	89.8	53.0/28.9/44.8
GPT-3 ($\text{Adapter}^{\text{H}}$)	40.1M	73.2	**91.5**	53.2/29.0/45.1
GPT-3 (LoRA)	4.7M	73.4	**91.7**	**53.8/29.8/45.9**
GPT-3 (LoRA)	37.7M	**74.0**	**91.6**	53.4/29.2/45.1

https://arxiv.org/abs/2106.09685

QLoRA

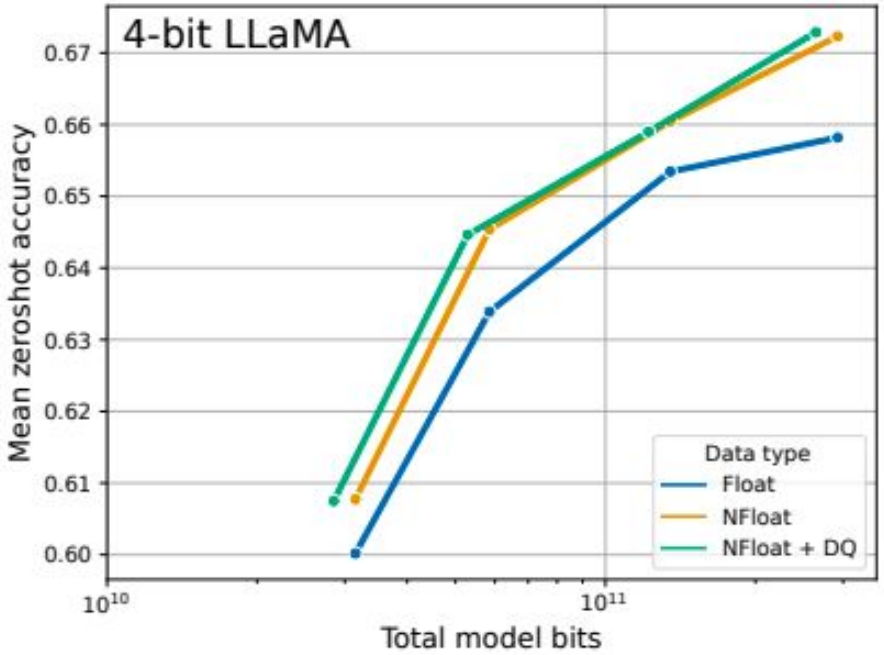
Initial model is quantized to 4bit, then LoRA is trained with 32bit precision



https://arxiv.org/abs/2305.14314

QLoRA

QLoRA uses 4bit quantization called NormalFloat



https://arxiv.org/abs/2305.14314

KronA

One-sentence idea:

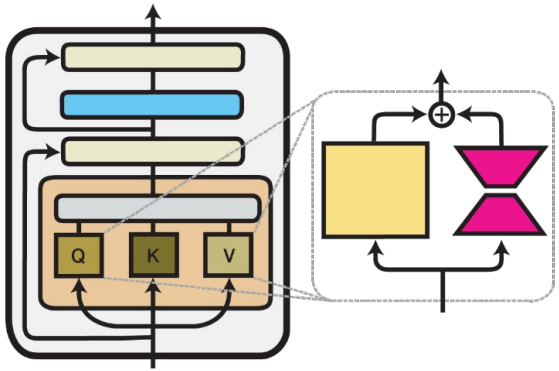
Matrix factorization through a Kronecker product

δW = WA ⊗ WB

$$\mathbf{A} \otimes \mathbf{B} : \mathbf{R}^{n \times m} \times \mathbf{R}^{k \times l} \rightarrow \mathbf{R}^{nk \times ml}$$

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{1,1}\mathbf{B} & \cdots & a_{1,n}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{m,1}\mathbf{B} & \cdots & a_{m,n}\mathbf{B} \end{bmatrix}$$

```
def krona_linear(x):
    x = x @ W  # regular linear
    x += kronecker_vector_prod(x, W_A, W_B)
    return scale * x

# same as x @ kronecker_product(A, B)
def kronecker_vector_prod(x, A, B):
    x = x.reshape(A.shape[1], B.shape[1])
    x = A.T @ x @ B
    return x.reshape(-1)
```



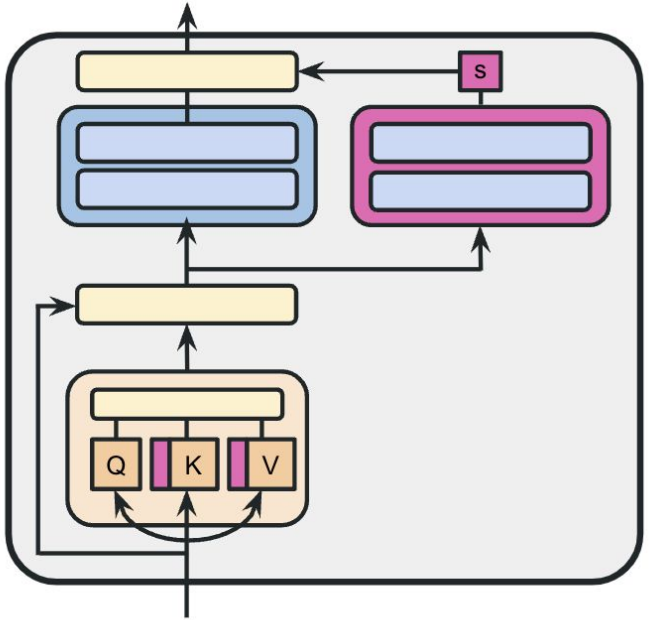
https://arxiv.org/abs/2212.10650

Hybrid approach: MAM Adapters

One-sentence idea:

scaled parallel adapter for FFN layer + soft prompt

```
def transformer_block_mam(x):
    x = concat([x, soft_prompt], dim=seq)
    residual = x
    x = SelfAttention(x)
    x = LN(x + residual)
    x_a = FFN(x) # parallel adapter
    x_a = scale * x_a
    x = LN(x + x_adapter)
    return x
```



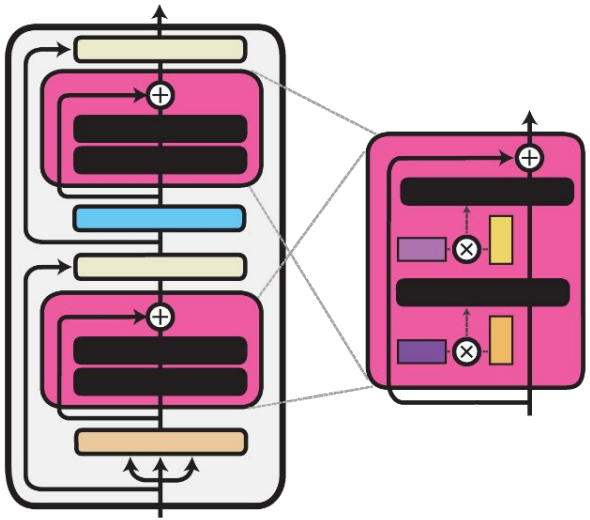
arxiv.org/abs/2110.04366

Hybrid approach: Compacter

One-sentence idea:

Kronecker product, low-rank matrices, and parameter sharing across layers to produce adapter weights

```
def compacter(x):
    x = LPHM(x)   # Essentially an FFN
    x = gelu(x)   # but
    x = LPHM(x)   # LPHM replaces linear
    return x

def lphm_forward(x):
    B = B_d @ B_u
    W = batched_kronecker_product(A, B)
    W = sum(W, dim=0)
    return x @ W + b
```



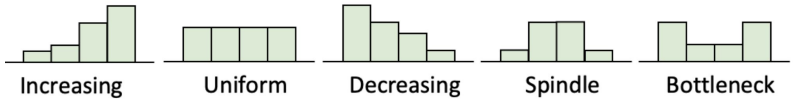
arxiv.org/abs/2110.04366

Hybrid approach: S4

One-sentence idea:
Automatically found combination of
Adapters, Prefix-Tuning, BitFit, and LoRA

Search process:
At 0.1% additional parameters

1. Find optimal grouping pattern: **spindle**
2. Find optimal parameter allocation pattern: **uniform**
3. Find which groups need tuning: all
4. find optimal combinations of PEFT techniques sequentially for G_1, G_2, G_3, G_4

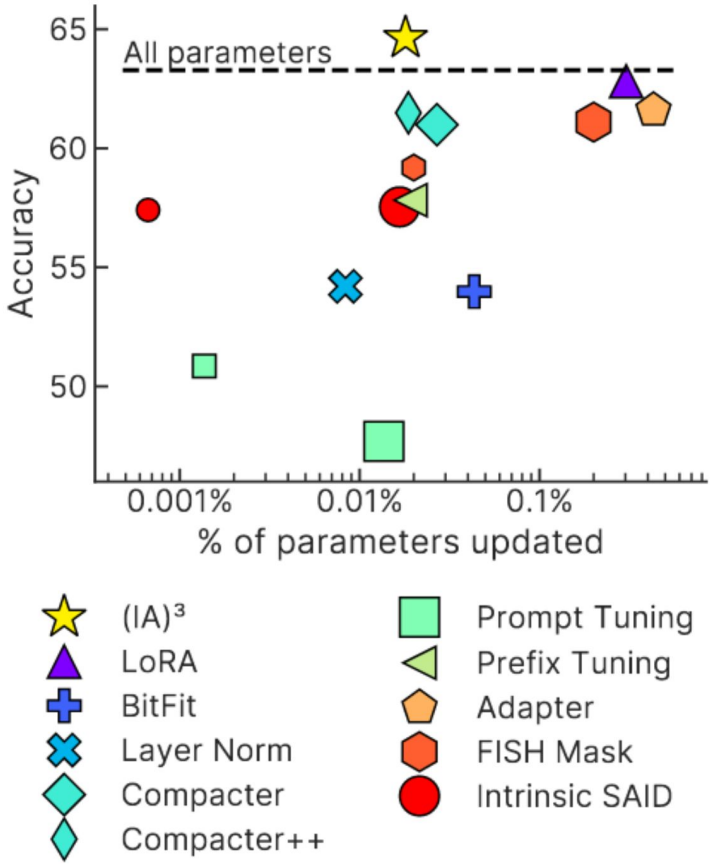


$$G_1 : A, L \quad G_3 : A, P, B$$
$$G_2 : A, P \quad G_4 : P, B, L$$

Which method to choose?

Method	% Trainable parameters	% Changed parameters	Evaluated on <1B	<20B	>20B
Adapters (Houlsby et al., 2019)	0.1 - 6	0.1 - 6	yes	yes	yes
AdaMix (Wang et al., 2022)	0.1 - 0.2	0.1 - 0.2	yes	no	no
SparseAdapter (He et al., 2022b)	2.0	2.0	yes	no	no
BitFit (Ben-Zaken et al., 2021)	0.05 - 0.1	0.05 - 0.1	yes	yes	yes
DiffPruning (Guo et al., 2020)	200	**0.5**	yes	no	no
Fish-Mask (Sung et al., 2021)	0.01 - 0.5	0.01 - 0.5	yes	yes	no
Prompt Tuning (Lester et al., 2021)	0.1	0.1	yes	yes	yes
Prefix-Tuning (Li and Liang, 2021)	0.1 - 4.0	0.1 - 4.0	yes	yes	yes
IPT (Qin et al., 2021)	56.0	56.0	yes	no	no
MAM Adapter (He et al., 2022a)	0.5	0.5	yes	no	no
Parallel Adapter (He et al., 2022a)	0.5	0.5	yes	no	no
Intrinsinc SAID (Aghajanyan et al., 2020)	0.001 - 0.1	**∼0.1 or 100**	yes	yes	no
LoRa (Hu et al., 2022)	0.01 - 0.5	**∼0.5 or ∼30**	yes	yes	yes
UniPELT (Mao et al., 2021)	1.0	1.0	yes	no	no
Compacter (Karimi Mahabadi et al., 2021)	0.05-0.07	**∼0.07 or ∼0.1**	yes	yes	no
PHM Adapter (Karimi Mahabadi et al., 2021)	0.2	**∼0.2 or ∼1.0**	yes	no	no
KronA (Edalati et al., 2022)	0.07	**∼0.07 or ∼30.0**	yes	no	no
KronA^{B}_{res} (Edalati et al., 2022)	0.07	**∼0.07 or ∼1.0**	yes	no	no
$(IA)^3$ (Liu et al., 2022)	0.02	0.02	no	yes	no
Ladder Side-Tuning(Sung et al., 2022)	7.5	7.5	yes	yes	no
FAR (Vucetic et al., 2022)	6.6-26.4	6.6-26.4	yes	no	no
S4-model (Chen et al., 2023)	0.5	**more than 0.5**	yes	yes	no

Additive PEFT: IA3 (T-Few)



https://arxiv.org/abs/2205.05638

What matters in PEFT?

- Number of parameters
- Training efficiency
  - Do we add parameters to the network? How expensive are they?
  - Does the method require to backpropagate through the original network?
  - Can the method efficiently utilize the GPU?
- Inference efficiency
  - Do we add parameters to the network? How expensive are they?
- Accuracy
  - Do we get good performance out of the network in the end?

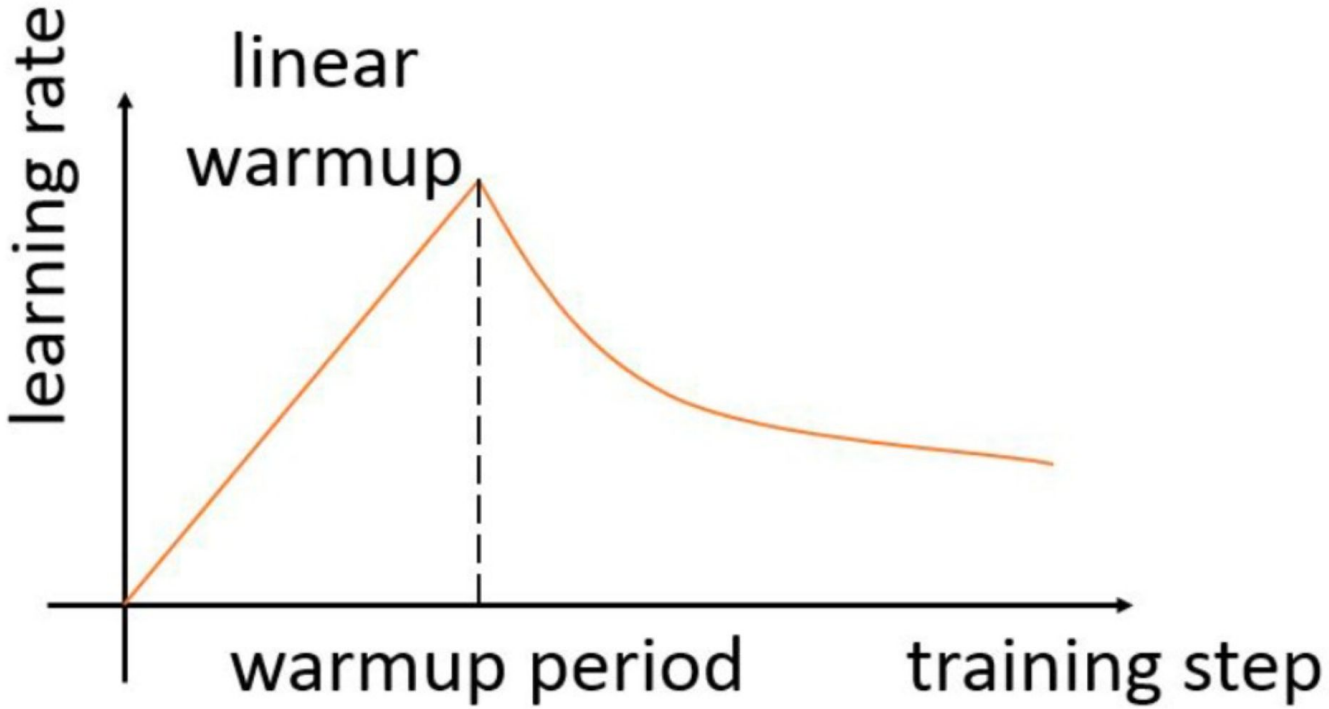
Which method to choose?

- 10s-100s of examples = prompt engineering, IA3
- 100s, 1000s of examples = prompt tuning, IA3
- more examples = LoRA adapters

LLM training tips

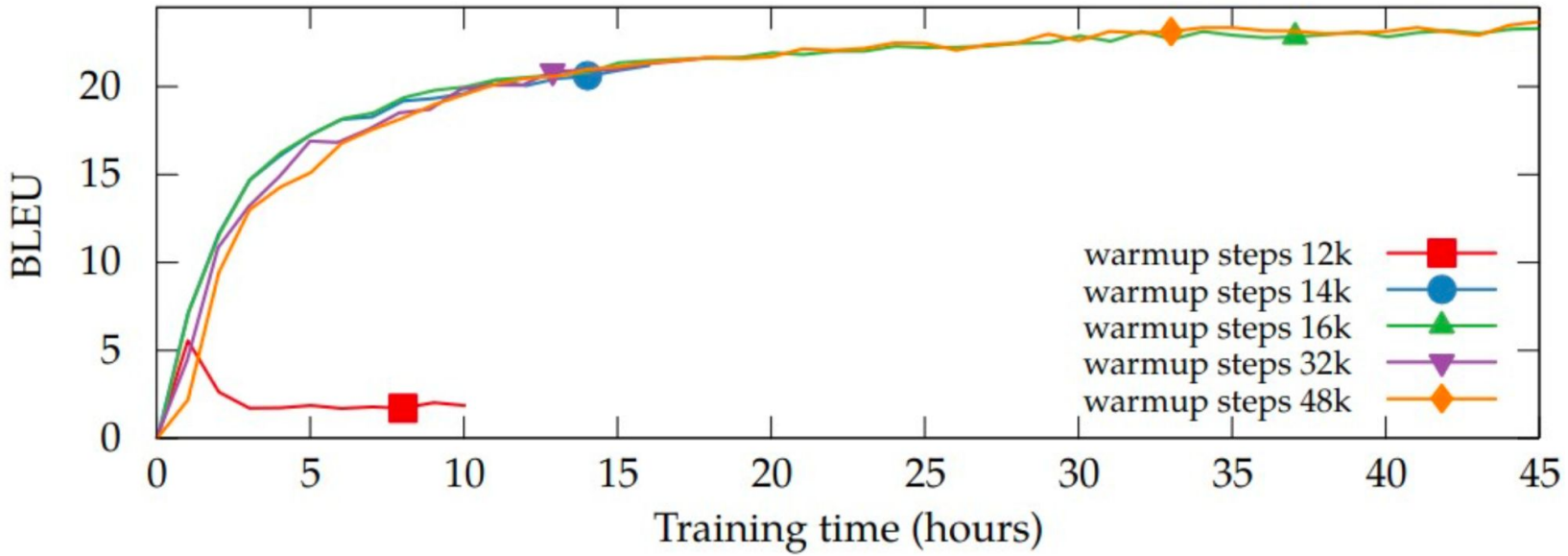
How to choose learning rate?

Learning rate “warm-up”
from Vaswani et al. (2017)



Warm-up

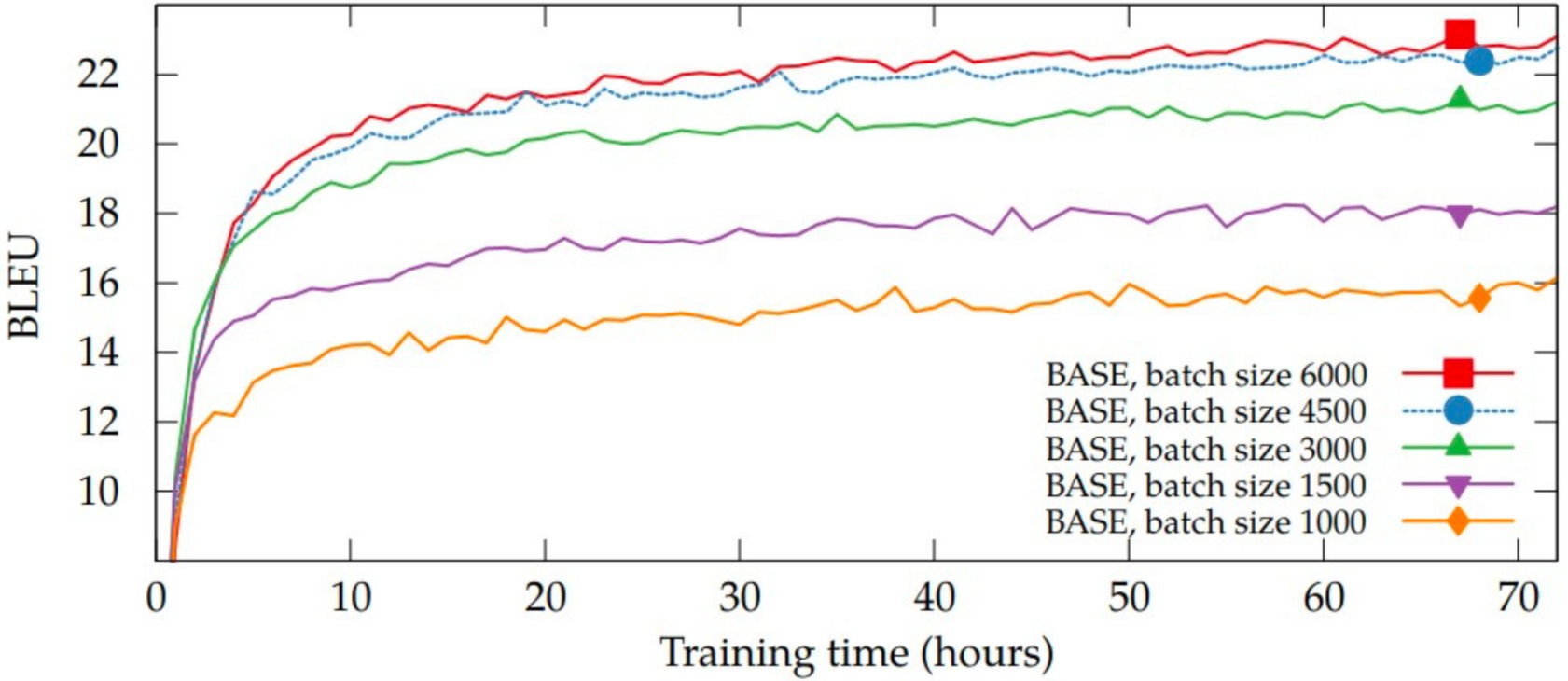
Learning rate “warm-up”
from Vaswani et al. (2017)



[Training Tips for the Transformer Model]

How to choose batch size?

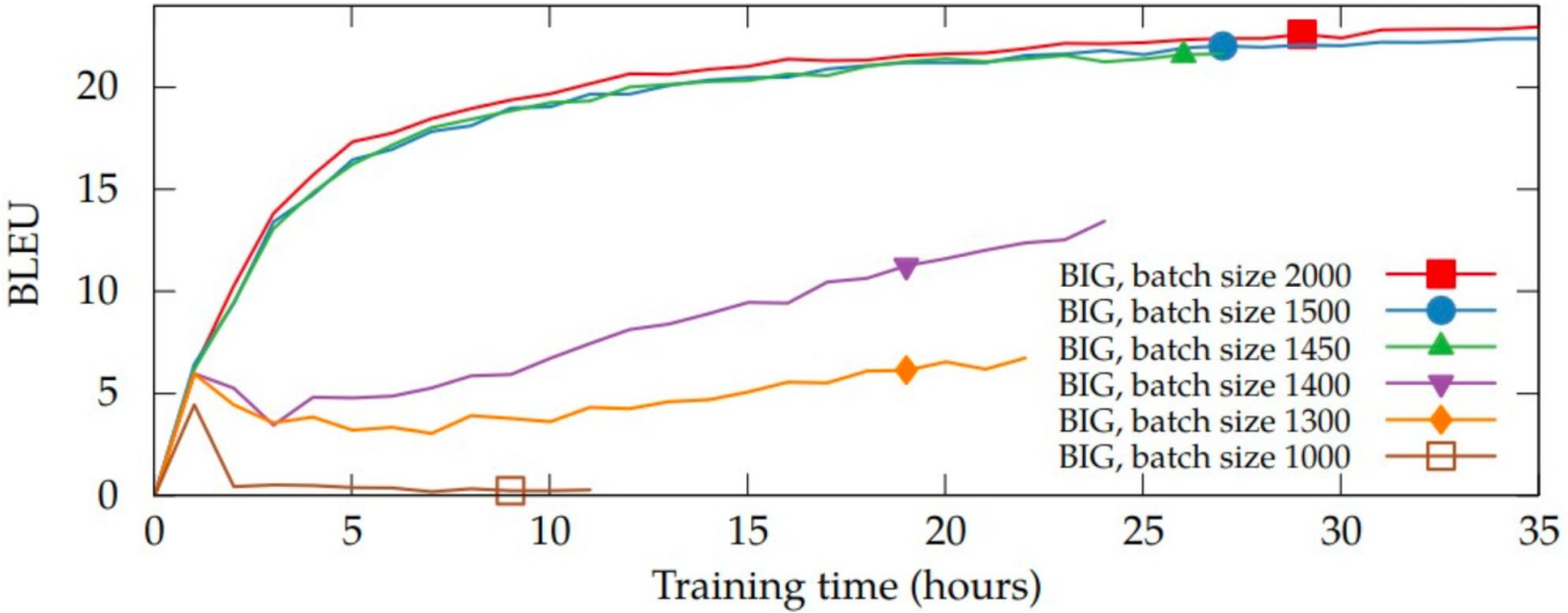
Batch should be big enough!



[Training Tips for the Transformer Model]

How to choose batch size?

Batch should be big enough!



[Training Tips for the Transformer Model]

Alternatives to position embeddings

- Vanilla Transformer - deterministic embeddings based on sin and cos
- BERT - learnable parameters of size 512
- GPT-2 and GPT-3 - learnable parameters of size 2048

Relative position encoding

Instead of adding positional information to input embeddings let's modify attention:

$$\text{RelativeAttention} = \text{Softmax}\left(\frac{QK^\top + S_{rel}}{\sqrt{D_h}}\right)V$$

Srel is a learned function of [query position – key position]

Self-Attention with Relative Position Representations

Rotary embeddings

Multiply Q and K vectors by rotation matrix:

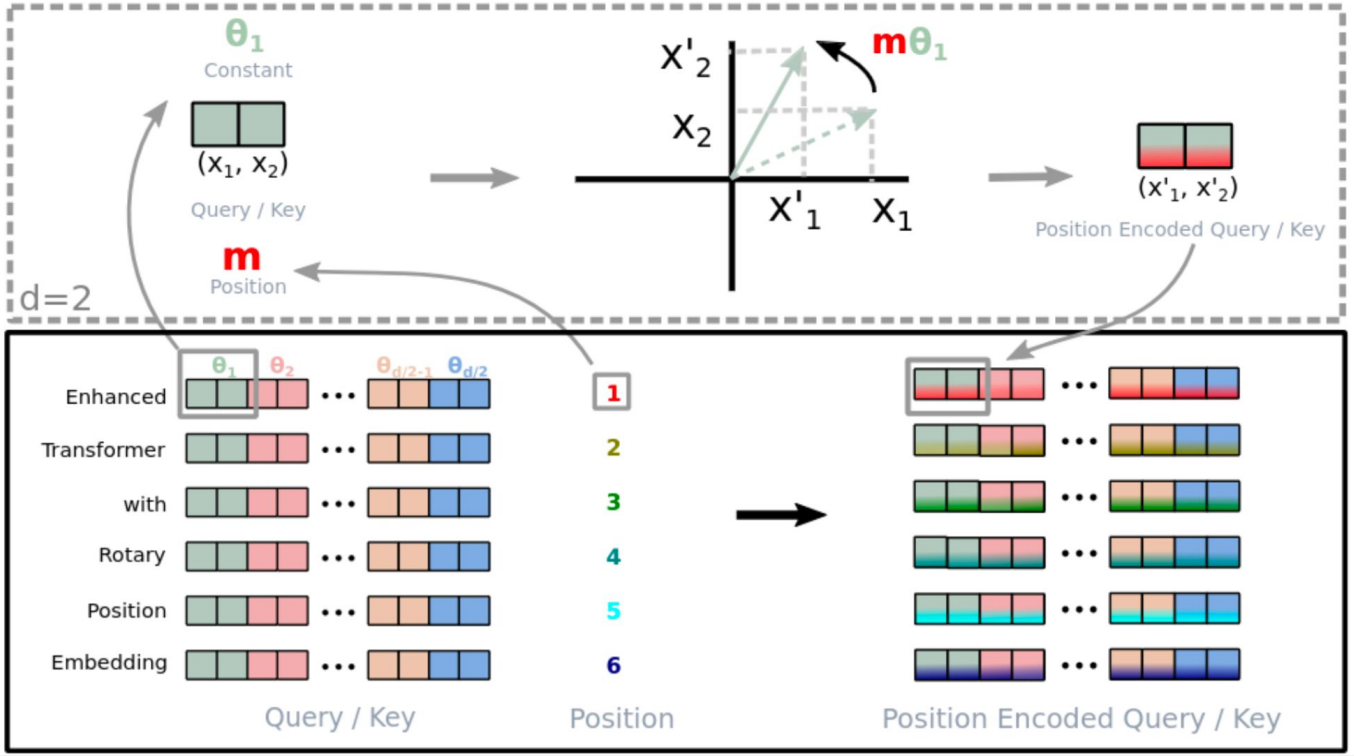
$$\boldsymbol{R}^{d}_{\Theta,m} = \begin{pmatrix} \cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ \sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & \cdots & 0 & 0 \\ 0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos m\theta_{d/2} & -\sin m\theta_{d/2} \\ 0 & 0 & 0 & 0 & \cdots & \sin m\theta_{d/2} & \cos m\theta_{d/2} \end{pmatrix}$$

… using an array of fixed (non-trainable) angles

$$\Theta = \{\theta_i = 10000^{-2(i-1)/d}, i \in [1, 2, ..., d/2]\}.$$

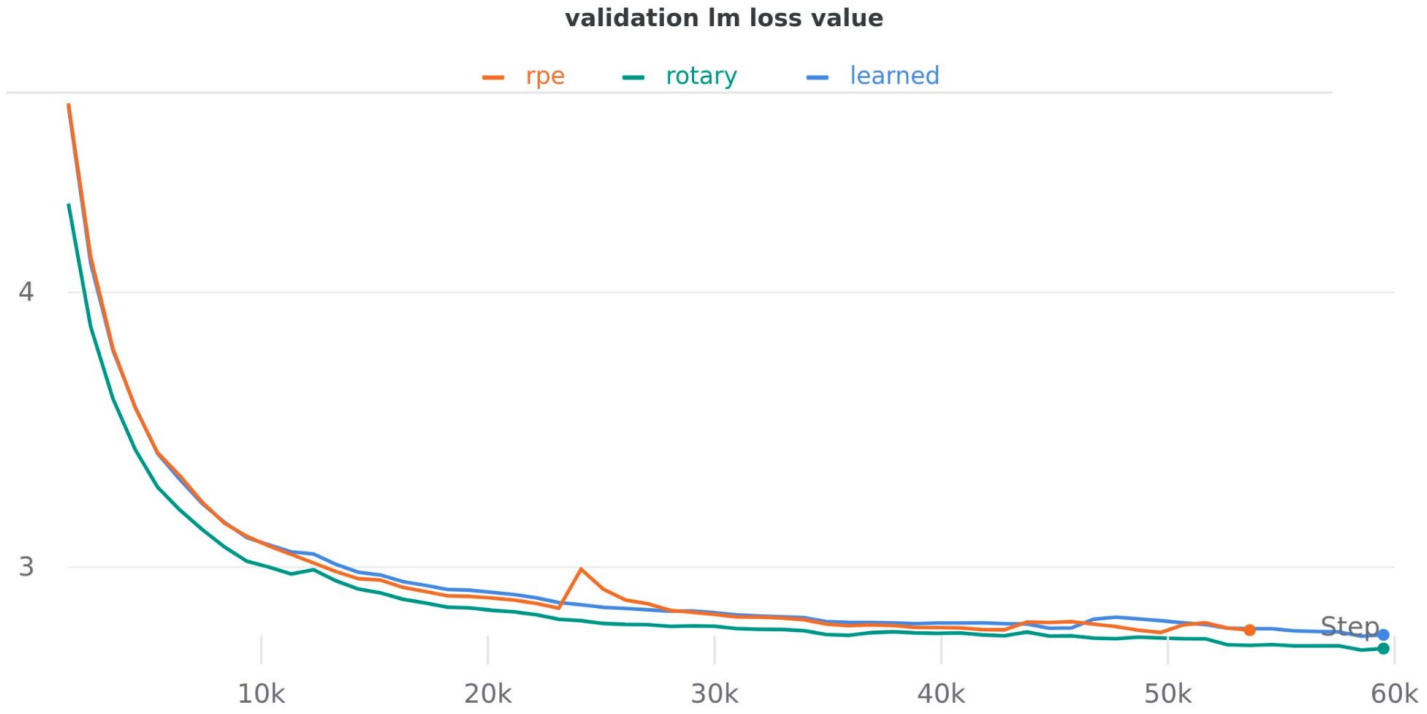
https://arxiv.org/abs/2104.09864
https://blog.eleuther.ai/rotary-embeddings/

Rotary embeddings



https://arxiv.org/abs/2104.09864
https://blog.eleuther.ai/rotary-embeddings/

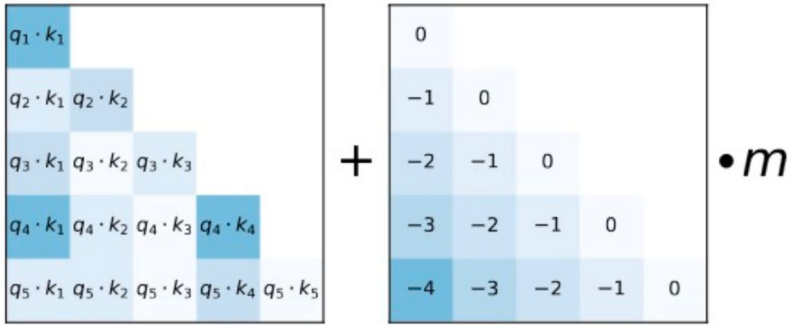
Rotary embeddings



https://arxiv.org/abs/2104.09864
https://blog.eleuther.ai/rotary-embeddings/

ALiBi

Add a Linear Bias to each attention head's pre-softmax logits

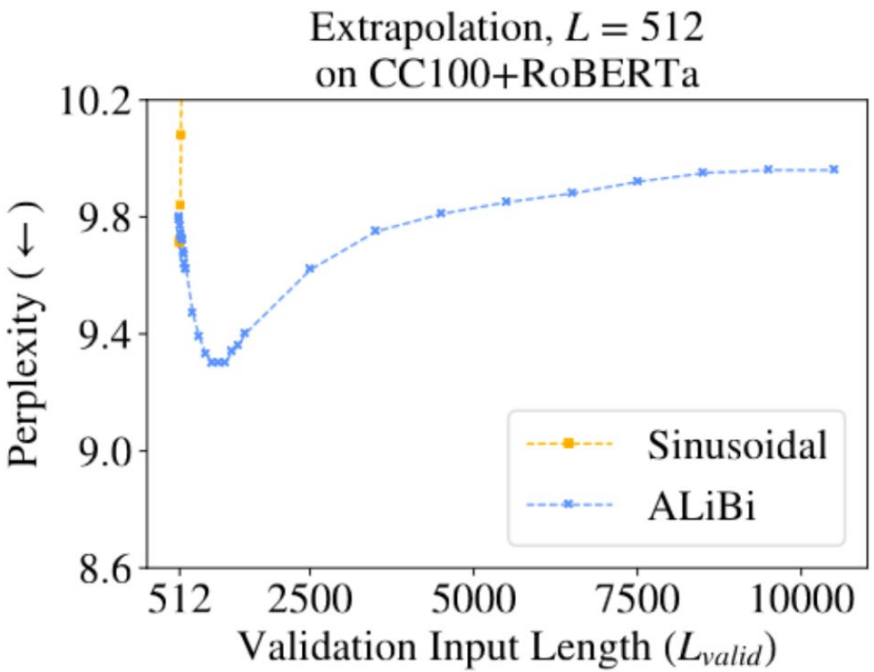


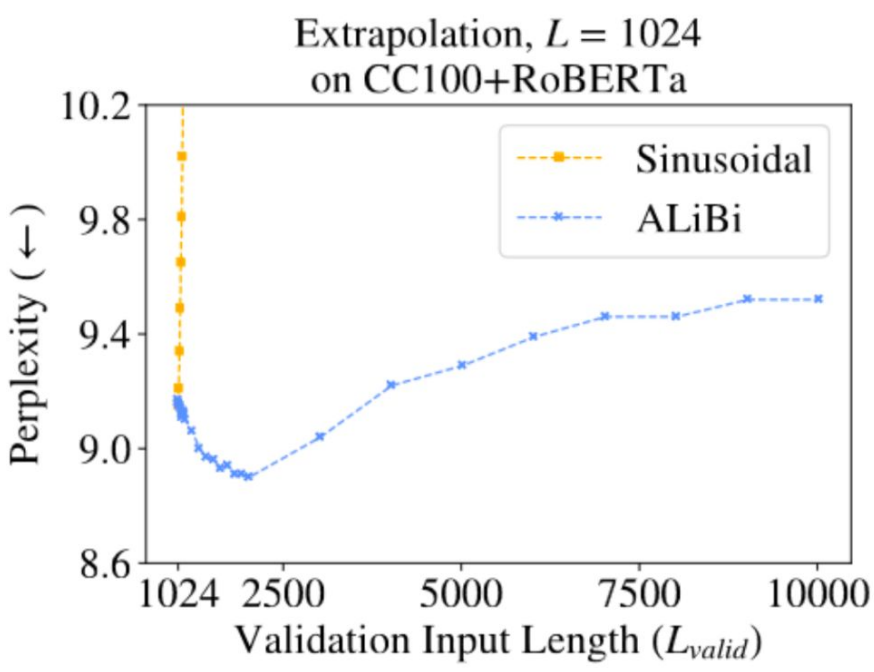
Here, m is a constant (non-trained) vector with one value per head:
*m[i] = 2 ^ (- i * scale)*

e.g. if scale = 0.5, $m = \frac{1}{2^{0.5}}, \frac{1}{2^{1}}, \frac{1}{2^{1.5}}, \cdots, \frac{1}{2^{8}}$

https://arxiv.org/abs/2108.12409

ALiBi





https://arxiv.org/abs/2108.12409