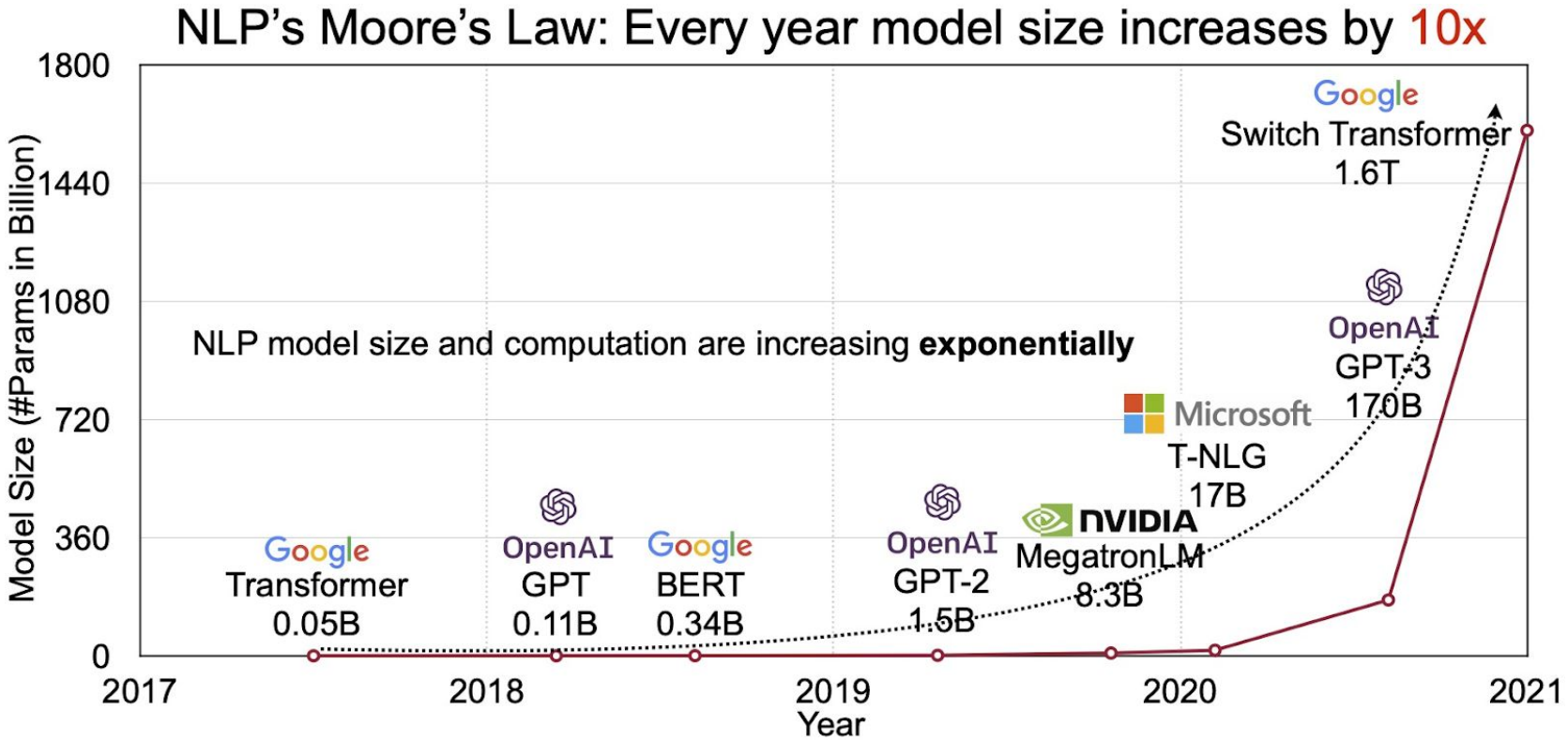
Parameter-efficient LM tuning

Qihang Fang, Jiannan Wang

Challenges for full fine-tuning



Challenges for full fine-tuning

- Computation and storage costs

Model	Full Finetuning
bigscience/T0_3B (3B params)	47.14GB GPU / 2.96GB CPU
bigscience/mt0-xxl (12B params)	OOM GPU
bigscience/bloomz-7b1 (7B params)	OOM GPU

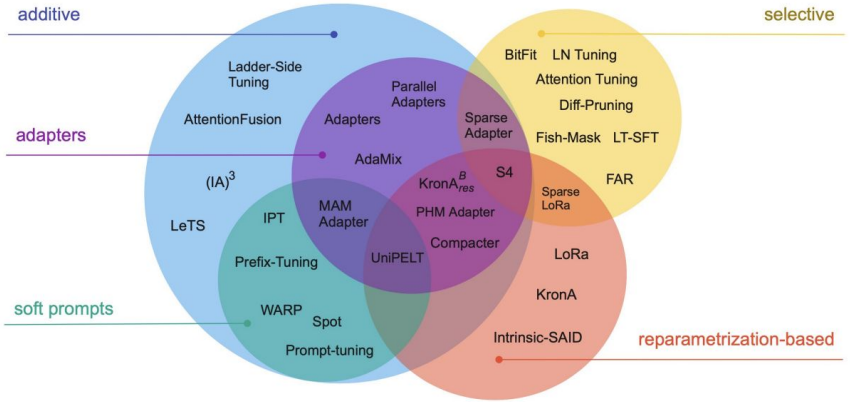
GPU Memory usage (A100 80GB)

- Catastrophic forgetting
  - Full fine-tune LLMs may lead to catastrophic forgetting.

Parameter-Efficient Fine-tuning (PEFT)

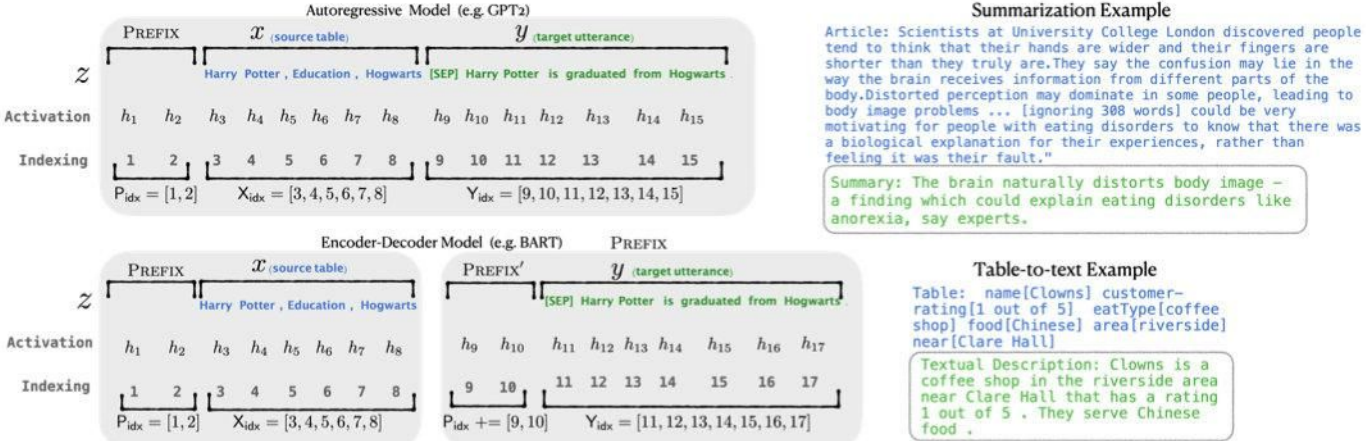
- PEFT approaches only **fine-tune a small number of (extra) model parameters** while freezing most parameters of the pretrained LLMs, thereby greatly decreasing the computational and storage costs.

- Three main classes
  - **Additive**
    - Add extra parameters or layers
  - **Reparametrization**
    - Leverage low-rank representations
  - Selective
    - Select parameters to update



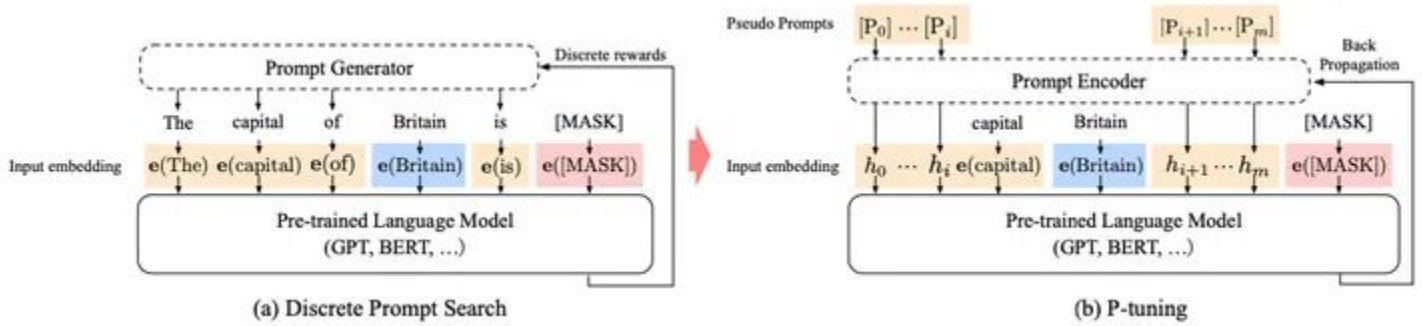
PEFT - Additive

- Prefix tuning
  - Define a template with trainable prefix parameters for generation task.
  - Prompts will be added to each attention layer.
  - Only the prefix parameters are optimized.
  - Add new MLP layers to finetune the prefix.



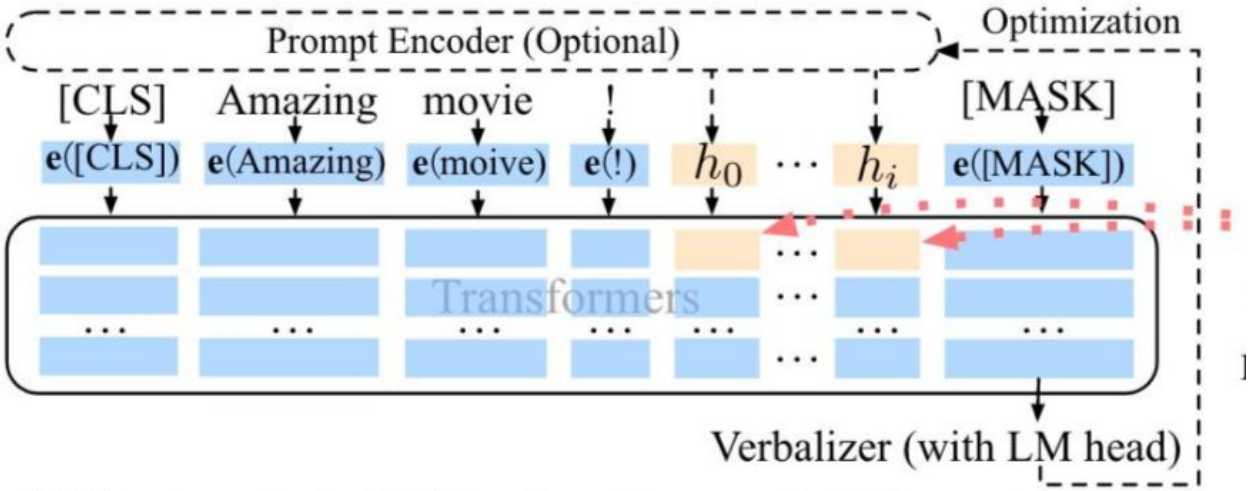
PEFT - Additive

- P-tuning
  - Define a new template for NLU task.
  - Considering the embeddings in pretrained model are discrete enough and related to the prompt. The author utilize a LSTM to get the embedding for the trainable prompts.
  - Add anchors to the template, such as “?”.



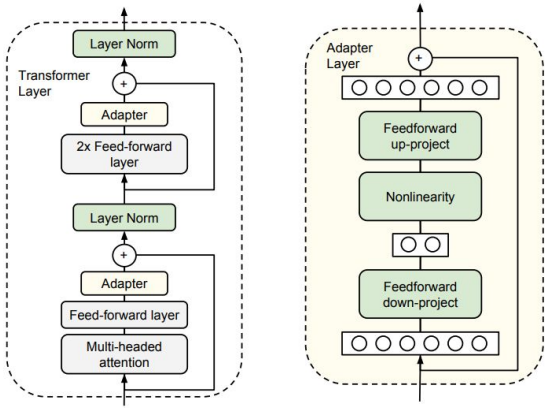
PEFT - Additive

- Prompt tuning
  - Freeze the network and utilize trainable prompt tokens.
  - No need for additional prompt encoder.
  - Only finetune the trainable prompt tokens.



PEFT - Additive

- Adapter
  - Add another layer to the LLMs.
  - Although we separate LORA into **Reparametrization** class, it can also be seen as adapter-based method.



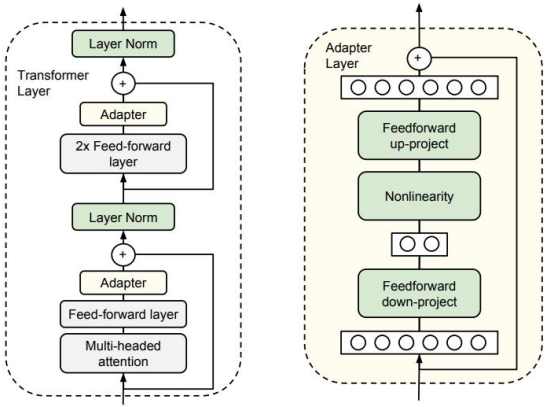
[Adapter Houlsby et.al. 2019]

LoRA: Low-Rank Adaptation of Large Language Models

Qihang Fang, Jiannan Wang

Motivation

- Computation and storage costs
- Drawbacks of adapter layer
  - Introduce Inference latency



[Adapter Houlsby et.al. 2019]

LORA

- Matrix rank:
  - the maximal number of linearly independent columns or rows.

$$rank(A) \leq \min(m, n)$$

- Rank-Deficient Matrix:

$$rank\left(\begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \end{bmatrix}\right) = 1$$

LORA

- Previous works have shown that over-parametrized large models reside on a **low intrinsic dimension**.

- The main idea behind LoRA is that the change in weights during model adaptation also has a low intrinsic rank/dimension.

- Concretely, if W□□ represents the weights of a single layer and ΔW□□ represents the change of weights during model adaptation, the authors propose that ΔW□□ is a low-rank matrix i.e.

$$rank(\Delta W_{n\times k}) << \min(n, k)$$

LORA

- Large models are trained to capture the general representation of their domain (language for LLMs, audio + language for models like Whisper, and vision for image generation models).

- These models capture a variety of features which allow them to be used for diverse tasks with reasonable zero-shot accuracy.

- However, **when adapting such a model to a specific task or dataset, only a few features need to be emphasized or re-learnt.** This means that the update matrix (ΔW) can be a low-rank matrix.

LORA

$$h = W_0 x + \Delta W x = W_0 x + BAx$$

$W_0 \in \mathbb{R}^{d \times k}$ $B \in \mathbb{R}^{d \times r}, A \in \mathbb{R}^{r \times k}$

$r << min(n, k)$

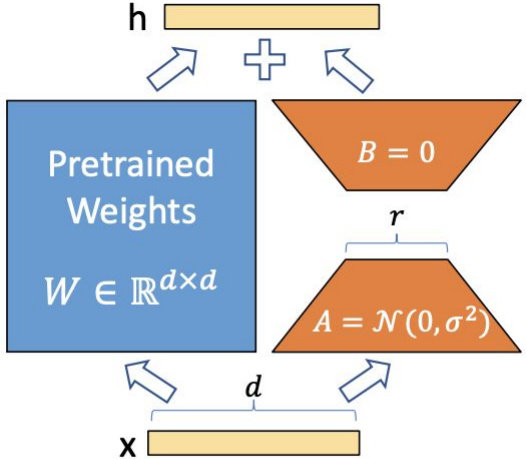


Figure 1: Our reparametrization. We only train A and B.

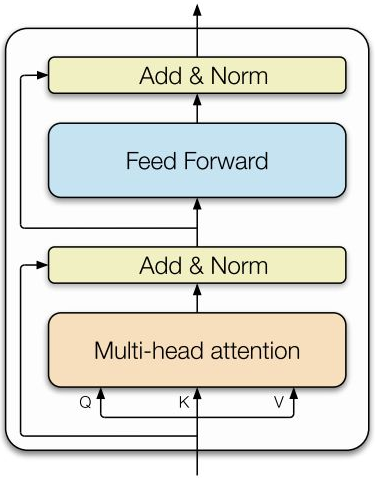
LORA

- Advantages:
  - **Reduction of training time and space**: We only need to train matrices *A* and *B* with less dimensions.
  - **No additional inference time:** *W* and *ΔW* can be calculated simultaneously, and we only need to add them together. There is no other serial multiplication.
  - **Easier task switching:** Swapping only the LoRA weights as opposed to all the parameters allows cheaper and faster switching between tasks.

LORA

- Apply to transformer:
  - adapt four weight matrices in the self-attention module (Wq, Wk, Wv, Wo)
  - Freeze the origin MLP module

- Benefit:
  - reduction in memory and storage usage



$$\text{Att}\colon \mathbb{R}^{\text{key}} \times \mathbb{R}^{\text{seq}\times\text{key}} \times \mathbb{R}^{\text{seq}\times\text{val}} \to \mathbb{R}^{\text{val}}$$

$$\text{Att}(Q,K,V) = \left(\underset{\text{seq}}{\text{softmax}} \frac{Q \underset{\text{key}}{\odot} K}{\sqrt{|\text{key}|}} \right) \underset{\text{seq}}{\odot} V$$

$$\begin{aligned} Q(x) &= x \cdot W_Q + b_k, \\ K(x) &= x \cdot W_K + b_q, \\ V(x) &= x \cdot W_V + b_v, \end{aligned}$$

$$W_Q,\ W_K \in \mathbf{R}^{\text{input}\times\text{key}}, W_V \in \mathbf{R}^{\text{input}\times\text{val}}$$
$$b_Q,\ b_K \in \mathbf{R}^{\text{key}},\ b_V \in \mathbf{R}^{\text{val}}$$

Experiments - Comparison with other PEFT methods

Model & Method	# Trainable Parameters	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B	Avg.
RoB_{base} (FT)*	125.0M	**87.6**	94.8	90.2	**63.6**	92.8	**91.9**	78.7	91.2	86.4
RoB_{base} (BitFit)*	0.1M	84.7	93.7	**92.7**	62.0	91.8	84.0	81.5	90.8	85.2
RoB_{base} (Adpt^{D})*	0.3M	$87.1_{\pm.0}$	$94.2_{\pm.1}$	$88.5_{\pm1.1}$	$60.8_{\pm.4}$	$93.1_{\pm.1}$	$90.2_{\pm.0}$	$71.5_{\pm2.7}$	$89.7_{\pm.3}$	84.4
RoB_{base} (Adpt^{D})*	0.9M	$87.3_{\pm.1}$	$94.7_{\pm.3}$	$88.4_{\pm.1}$	$62.6_{\pm.9}$	$93.0_{\pm.2}$	$90.6_{\pm.0}$	$75.9_{\pm2.2}$	$90.3_{\pm.1}$	85.4
RoB_{base} (LoRA)	0.3M	$87.5_{\pm.3}$	$\mathbf{95.1}_{\pm.2}$	$89.7_{\pm.7}$	$63.4_{\pm1.2}$	$\mathbf{93.3}_{\pm.3}$	$90.8_{\pm.1}$	$\mathbf{86.6}_{\pm.7}$	$\mathbf{91.5}_{\pm.2}$	**87.2**
$\text{RoB}_{\text{large}}$ (FT)*	355.0M	90.2	**96.4**	**90.9**	68.0	94.7	**92.2**	86.6	92.4	88.9
$\text{RoB}_{\text{large}}$ (LoRA)	0.8M	$\mathbf{90.6}_{\pm.2}$	$96.2_{\pm.5}$	$\mathbf{90.9}_{\pm1.2}$	$\mathbf{68.2}_{\pm1.9}$	$\mathbf{94.9}_{\pm.3}$	$91.6_{\pm.1}$	$\mathbf{87.4}_{\pm2.5}$	$\mathbf{92.6}_{\pm.2}$	**89.0**
$\text{RoB}_{\text{large}}$ (Adpt^{P})†	3.0M	$90.2_{\pm.3}$	$96.1_{\pm.3}$	$90.2_{\pm.7}$	$\mathbf{68.3}_{\pm1.0}$	$\mathbf{94.8}_{\pm.2}$	$\mathbf{91.9}_{\pm.1}$	$83.8_{\pm2.9}$	$92.1_{\pm.7}$	88.4
$\text{RoB}_{\text{large}}$ (Adpt^{P})†	0.8M	$\mathbf{90.5}_{\pm.3}$	$\mathbf{96.6}_{\pm.2}$	$89.7_{\pm1.2}$	$67.8_{\pm2.5}$	$\mathbf{94.8}_{\pm.3}$	$91.7_{\pm.2}$	$80.1_{\pm2.9}$	$91.9_{\pm.4}$	87.9
$\text{RoB}_{\text{large}}$ (Adpt^{H})†	6.0M	$89.9_{\pm.5}$	$96.2_{\pm.3}$	$88.7_{\pm2.9}$	$66.5_{\pm4.4}$	$94.7_{\pm.2}$	$92.1_{\pm.1}$	$83.4_{\pm1.1}$	$91.0_{\pm1.7}$	87.8
$\text{RoB}_{\text{large}}$ (Adpt^{H})†	0.8M	$90.3_{\pm.3}$	$96.3_{\pm.5}$	$87.7_{\pm1.7}$	$66.3_{\pm2.0}$	$94.7_{\pm.2}$	$91.5_{\pm.1}$	$72.9_{\pm2.9}$	$91.5_{\pm.5}$	86.4
$\text{RoB}_{\text{large}}$ (LoRA)†	0.8M	$\mathbf{90.6}_{\pm.2}$	$96.2_{\pm.5}$	$\mathbf{90.2}_{\pm1.0}$	$68.2_{\pm1.9}$	$\mathbf{94.8}_{\pm.3}$	$91.6_{\pm.2}$	$\mathbf{85.2}_{\pm1.1}$	$\mathbf{92.3}_{\pm.5}$	**88.6**
DeB_{XXL} (FT)*	1500.0M	91.8	**97.2**	92.0	72.0	**96.0**	92.7	93.9	92.9	91.1
DeB_{XXL} (LoRA)	4.7M	$\mathbf{91.9}_{\pm.2}$	$96.9_{\pm.2}$	$\mathbf{92.6}_{\pm.6}$	$\mathbf{72.4}_{\pm1.1}$	$\mathbf{96.0}_{\pm.1}$	$\mathbf{92.9}_{\pm.1}$	$\mathbf{94.9}_{\pm.4}$	$\mathbf{93.0}_{\pm.2}$	**91.3**

Experiments - Comparison with other PEFT methods

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	$\mathbf{70.4}_{\pm.1}$	$\mathbf{8.85}_{\pm.02}$	$\mathbf{46.8}_{\pm.2}$	$\mathbf{71.8}_{\pm.1}$	$\mathbf{2.53}_{\pm.02}$
GPT-2 L (FT)*	774.03M	68.5	8.78	46.0	69.9	2.45
GPT-2 L (Adapter^{L})	0.88M	$69.1_{\pm.1}$	$8.68_{\pm.03}$	$46.3_{\pm.0}$	$71.4_{\pm.2}$	$\mathbf{2.49}_{\pm.0}$
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	$\mathbf{70.4}_{\pm.1}$	$\mathbf{8.89}_{\pm.02}$	$\mathbf{46.8}_{\pm.2}$	$\mathbf{72.0}_{\pm.2}$	$2.47_{\pm.02}$

Experiments - Comparison with other PEFT methods

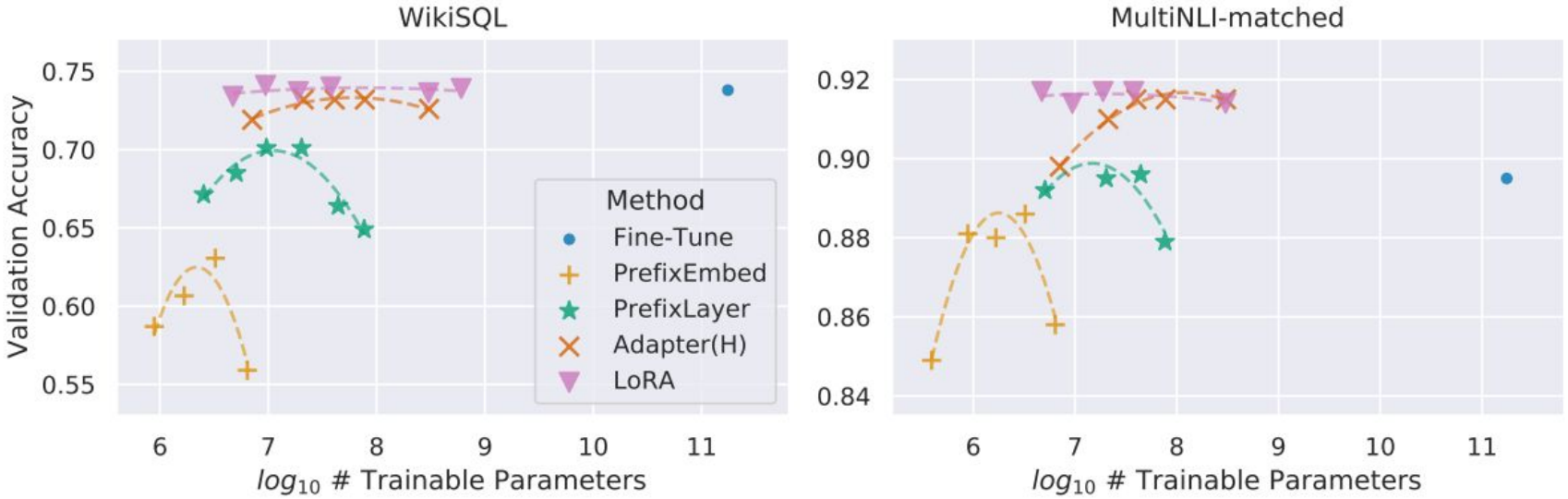


Figure 2: GPT-3 175B validation accuracy vs. number of trainable parameters of several adaptation methods on WikiSQL and MNLI-matched. LoRA exhibits better scalability and task performance. See Section F.2 for more details on the plotted data points.

Understanding the low-rank updates

- Given a parameter budget constraint, which **subset of weight matrices in a pre-trained Transformer should we adapt** to maximize downstream performance?

	# of Trainable Parameters = 18M						
Weight Type	W_q	W_k	W_v	W_o	W_q, W_k	W_q, W_v	W_q, W_k, W_v, W_o
Rank r	8	8	8	8	4	4	2
WikiSQL ($\pm$0.5%)	70.4	70.0	73.0	73.2	71.4	**73.7**	**73.7**
MultiNLI ($\pm$0.1%)	91.0	90.8	91.0	91.3	91.3	91.3	**91.7**

Table 5: Validation accuracy on WikiSQL and MultiNLI after applying LoRA to different types of attention weights in GPT-3, given the same number of trainable parameters. Adapting both W_q and W_v gives the best performance overall. We find the standard deviation across random seeds to be consistent for a given dataset, which we report in the first column.

More is better!

Understanding the low-rank updates

- Is the “optimal” adaptation matrix ΔW really rank-deficient? If so, what is a good rank to use in practice?

	Weight Type	$r = 1$	$r = 2$	$r = 4$	$r = 8$	$r = 64$
WikiSQL($\pm 0.5\%$)	W_q	68.8	69.6	70.5	70.4	70.0
	W_q, W_v	73.4	73.3	73.7	73.8	73.5
	W_q, W_k, W_v, W_o	74.1	73.7	74.0	74.0	73.9
MultiNLI ($\pm 0.1\%$)	W_q	90.7	90.9	91.1	90.7	90.7
	W_q, W_v	91.3	91.4	91.3	91.6	91.4
	W_q, W_k, W_v, W_o	91.2	91.7	91.7	91.5	91.4

Table 6: Validation accuracy on WikiSQL and MultiNLI with different rank r. To our surprise, a rank as small as one suffices for adapting both W_q and W_v on these datasets while training W_q alone needs a larger r. We conduct a similar experiment on GPT-2 in Section H.2.

Small rank is enough.

Understanding the low-rank updates

- Is the “optimal” adaptation matrix ΔW really rank-deficient? If so, what is a good rank to use in practice?

	Weight Type	$r = 1$	$r = 2$	$r = 4$	$r = 8$	$r = 64$
WikiSQL(±0.5%)	W_q	68.8	69.6	70.5	70.4	70.0
	W_q, W_v	73.4	73.3	73.7	73.8	73.5
	W_q, W_k, W_v, W_o	74.1	73.7	74.0	74.0	73.9
MultiNLI (±0.1%)	W_q	90.7	90.9	91.1	90.7	90.7
	W_q, W_v	91.3	91.4	91.3	91.6	91.4
	W_q, W_k, W_v, W_o	91.2	91.7	91.7	91.5	91.4

Table 6: Validation accuracy on WikiSQL and MultiNLI with different rank r. To our surprise, a rank as small as one suffices for adapting both W_q and W_v on these datasets while training W_q alone needs a larger r. We conduct a similar experiment on GPT-2 in Section H.2.

Small rank is enough.

Understanding the low-rank updates

- Is the “optimal” adaptation matrix ΔW really rank-deficient? If so, what is a good rank to use in practice?
  - Subspace similarity between different r.

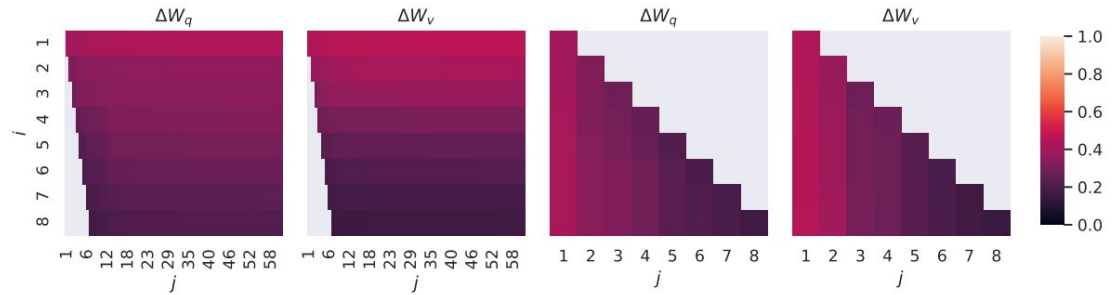


Figure 3: Subspace similarity between column vectors of $A_{r=8}$ and $A_{r=64}$ for both ΔW_q and ΔW_v. The third and the fourth figures zoom in on the lower-left triangle in the first two figures. The top directions in $r = 8$ are included in $r = 64$, and vice versa.

Directions corresponding to the top singular vector overlap significantly between A_r=8 and Ar=64, while others do not.

Understanding the low-rank updates

- What is the connection between ΔW and W? Does ΔW highly correlate with W? How large is ΔW comparing to W?

	$r = 4$			$r = 64$		
	ΔW_q	W_q	Random	ΔW_q	W_q	Random
$\|\|U^\top W_q V^\top\|\|_F =$	0.32	21.67	0.02	1.90	37.71	0.33
$\|\|W_q\|\|_F = 61.95$	$\|\|\Delta W_q\|\|_F = 6.91$			$\|\|\Delta W_q\|\|_F = 3.57$		

Table 7: The Frobenius norm of $U^\top W_q V^\top$ where U and V are the left/right top r singular vector directions of either (1) ΔW_q, (2) W_q, or (3) a random matrix. The weight matrices are taken from the 48th layer of GPT-3.

1. ΔW has a stronger correlation with W compared to a random matrix.
2. ΔW only amplifies directions that are not emphasized in W.
3. Larger r has larger amplification factor.

(21.67/0.32=0.01 for r=4, and 1.9/37.71=0.05 for r=64)

QLoRA: Efficient Finetuning of Quantized LLMs

Qihang Fang, Jiannan Wang

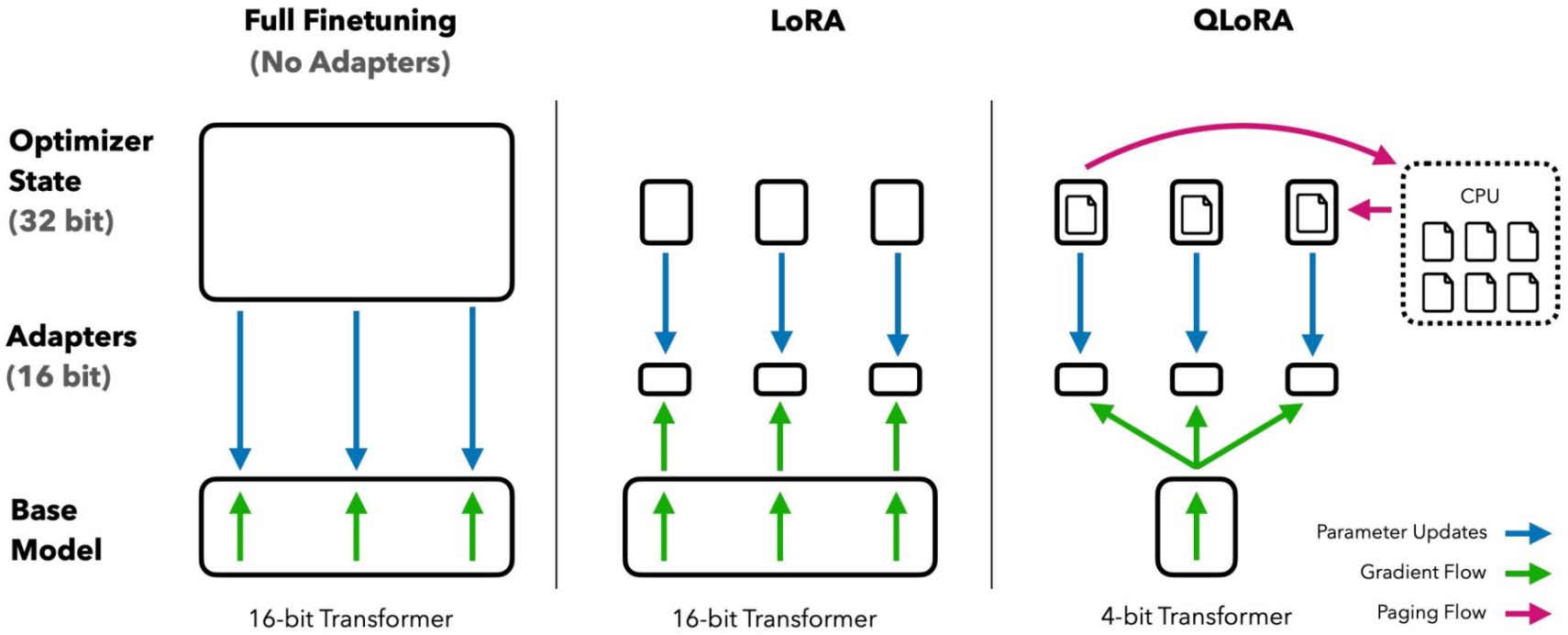
Motivation

- Computation and storage costs

- 4-Bit quantization can only be used for inference.
  - Finetune StableLM-3B model with microbatchsize of 4.

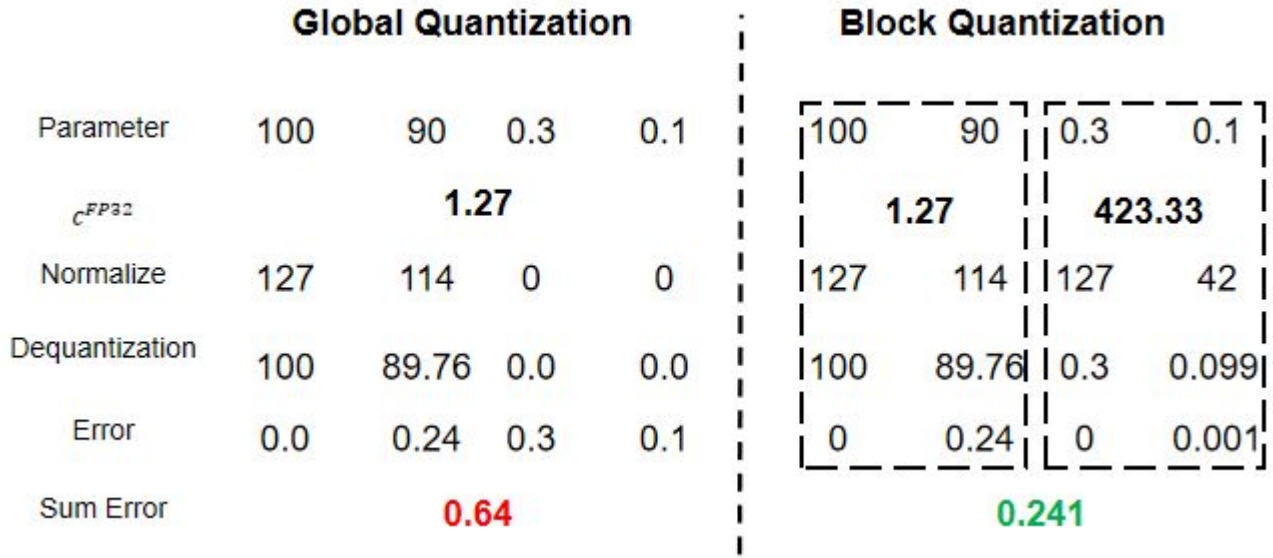
Settings	Inference Memory
Default (bf16-mixed)	40.21 GB
--precision bf16-true	13.52 GB
--precision bf16-true --quantize bnb.nf4	4.57 GB
--precision bf16-true --quantize bnb.nf4-dq	4.26 GB

QLORA


Full Finetuning
(No Adapters)
LoRA
QLoRA
Optimizer State (32 bit)
Adapters (16 bit)
Base Model
CPU
16-bit Transformer
16-bit Transformer
4-bit Transformer
Parameter Updates
Gradient Flow
Paging Flow

4-bit NormalFloat Quantization

- Block quantization:
  - Separate parameter into different blocks to increase the precision.



4-bit NormalFloat Quantization

- Quantile Quantization:
  - Estimating the quantile of the input parameters through the empirical cumulative distribution function.

	Traditional Quantization				Quantile Quantization			
Parameters	100	0.5	0.3	0.1	100	0.5	0.3	0.1
Dequantization	66.7	33.3	33.3	33.3	50.3	50.3	0.2	0.2
Error	33.3	32.8	33.0	33.2	49.7	49.8	0.1	0.1
Sum Error	132.3				99.7			

4-bit NormalFloat Quantization

- Quantile Quantization:
  - Advantages:
    - Ignore outliers
    - Adapt to different numerical ranges
  - Drawbacks:
    - We need to calculate the quantile for each batch.

4-bit NormalFloat Quantization

- NormalFloat Quantization:
  - Steps:
    - Assume that the parameters of a pretrained network agree with the gaussian distributions with the mean of 0.
    - Scale the distribution to [-1, 1].
    - Calculate the quantile based on the new distribution.
    - Quantize the parameters.
  - Drawbacks:
    - We need to store a 32-bit variance of each block.

4-bit NormalFloat Quantization

- Double Quantization:
  - Motivation:
    - Quantize the variances of blocks.
  - Steps:
    - Quantize the variances of multiple blocks to 8-bit.
    - Store one 32-bit variances for these blocks.

Page Optimizer

- Motivation:
  - Avoid OOM during the gradient backward.

- Solution:
  - Move some pages of optimizer to CPU while calculate the gradient.
  - Move these pages back while optimizing.

Experiments - Fully fine-tuneing replication and Data type

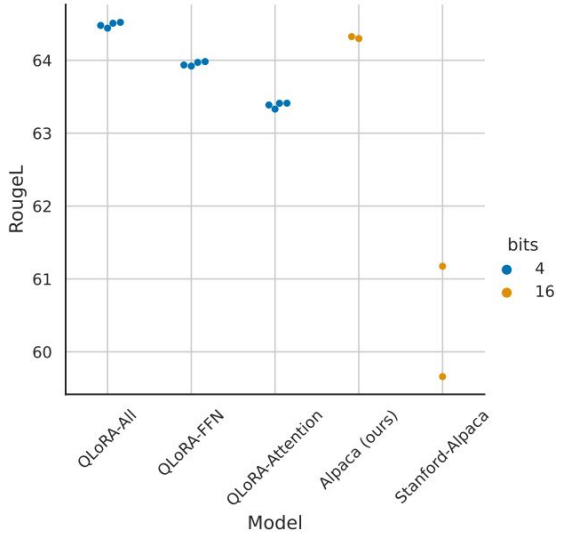


Figure 2: RougeL for LLaMA 7B models on the Alpaca dataset. Each point represents a run with a different random seed. We improve on the Stanford Alpaca fully finetuned default hyperparameters to construct a strong 16-bit baseline for comparisons. Using LoRA on all transformer layers is critical to match 16-bit performance.

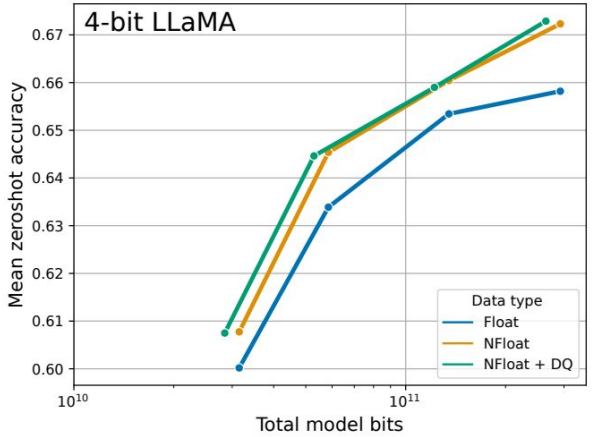


Figure 3: Mean zero-shot accuracy over Winogrande, HellaSwag, PiQA, Arc-Easy, and Arc-Challenge using LLaMA models with different 4-bit data types. The NormalFloat data type significantly improves the bit-for-bit accuracy gains compared to regular 4-bit Floats. While Double Quantization (DQ) only leads to minor gains, it allows for a more fine-grained control over the memory footprint to fit models of certain size (33B/65B) into certain GPUs (24/48GB).

Experiments - Comparison with LORA

Table 3: Experiments comparing 16-bit BrainFloat (BF16), 8-bit Integer (Int8), 4-bit Float (FP4), and 4-bit NormalFloat (NF4) on GLUE and Super-NaturalInstructions. QLoRA replicates 16-bit LoRA and full-finetuning.

Dataset	GLUE (Acc.)	Super-NaturalInstructions (RougeL)				
Model	RoBERTa-large	T5-80M	T5-250M	T5-780M	T5-3B	T5-11B
BF16	88.6	40.1	42.1	48.0	54.3	62.0
BF16 replication	88.6	40.0	42.2	47.3	54.9	-
LoRA BF16	88.8	40.5	42.6	47.1	55.4	60.7
QLoRA Int8	88.8	40.4	42.9	45.4	56.5	60.7
QLoRA FP4	88.6	40.3	42.4	47.5	55.6	60.9
QLoRA NF4 + DQ	-	40.4	42.7	47.7	55.3	60.9

Experiments - Data type and parameters

Table 4: Mean 5-shot MMLU test accuracy for LLaMA 7-65B models finetuned with adapters on Alpaca and FLAN v2 for different data types. Overall, NF4 with double quantization (DQ) matches BFloat16 performance, while FP4 is consistently one percentage point behind both.

	Mean 5-shot MMLU Accuracy								
LLaMA Size	7B		13B		33B		65B		Mean
Dataset	Alpaca	FLAN v2	Alpaca	FLAN v2	Alpaca	FLAN v2	Alpaca	FLAN v2	
BFloat16	38.4	45.6	47.2	50.6	57.7	60.5	61.8	62.5	53.0
Float4	37.2	44.0	47.3	50.0	55.9	58.5	61.3	63.3	52.2
NFloat4 + DQ	39.0	44.5	47.5	50.7	57.3	59.2	61.8	63.9	53.1

Experiments - Comparison with chatbot models

Table 6: Zero-shot Vicuna benchmark scores as a percentage of the score obtained by ChatGPT evaluated by GPT-4. We see that OASST1 models perform close to ChatGPT despite being trained on a very small dataset and having a fraction of the memory requirement of baseline models.

Model / Dataset	Params	Model bits	Memory	ChatGPT vs Sys	Sys vs ChatGPT	Mean	95% CI
GPT-4	-	-	-	119.4%	110.1%	**114.5%**	2.6%
Bard	-	-	-	93.2%	96.4%	94.8%	4.1%
Guanaco	65B	4-bit	41 GB	96.7%	101.9%	**99.3%**	4.4%
Alpaca	65B	4-bit	41 GB	63.0%	77.9%	70.7%	4.3%
FLAN v2	65B	4-bit	41 GB	37.0%	59.6%	48.4%	4.6%
Guanaco	33B	4-bit	21 GB	96.5%	99.2%	**97.8%**	4.4%
Open Assistant	33B	16-bit	66 GB	91.2%	98.7%	94.9%	4.5%
Alpaca	33B	4-bit	21 GB	67.2%	79.7%	73.6%	4.2%
FLAN v2	33B	4-bit	21 GB	26.3%	49.7%	38.0%	3.9%
Vicuna	13B	16-bit	26 GB	91.2%	98.7%	**94.9%**	4.5%
Guanaco	13B	4-bit	10 GB	87.3%	93.4%	90.4%	5.2%
Alpaca	13B	4-bit	10 GB	63.8%	76.7%	69.4%	4.2%
HH-RLHF	13B	4-bit	10 GB	55.5%	69.1%	62.5%	4.7%
Unnatural Instr.	13B	4-bit	10 GB	50.6%	69.8%	60.5%	4.2%
Chip2	13B	4-bit	10 GB	49.2%	69.3%	59.5%	4.7%
Longform	13B	4-bit	10 GB	44.9%	62.0%	53.6%	5.2%
Self-Instruct	13B	4-bit	10 GB	38.0%	60.5%	49.1%	4.6%
FLAN v2	13B	4-bit	10 GB	32.4%	61.2%	47.0%	3.6%
Guanaco	7B	4-bit	5 GB	84.1%	89.8%	**87.0%**	5.4%
Alpaca	7B	4-bit	5 GB	57.3%	71.2%	64.4%	5.0%
FLAN v2	7B	4-bit	5 GB	33.3%	56.1%	44.8%	4.0%

Based on our automated and human evaluations, we find that the top QLORA tuned model, Guanaco 65B, which we finetune on a variant of OASST1, is the best-performing open-source chatbot model and offers performance competitive to ChatGPT.

Experiments - Qualitative Analysis

Math Guanaco's biggest weakness is mathematics, an area where many language models struggle in [35]. When Guanaco shows its work, it tends to be accurate, e.g.,

> **User**: Lee mows one lawn and charges $33. Last week he mowed 16 lawns and three customers each gave him a $10 tip. How many dollars did Lee earn mowing lawns last week?
>
> **Guanaco**: `Lee earned $582 mowing lawns last week.`
>
> `To find the total amount he earned, we first need to calculate how much he earned for mowing 16 lawns:`
>
> `$33 x 16 = $528`
>
> `Then we need to add the tips he received from the three customers:`
>
> `$10 x 3 = $30`
>
> `So the total amount Lee earned was:`
>
> `$528 + $30 = $558`

However Guanaco can breakdown at even simple problems if it fails to break them down step-by-step, a known issue [63].

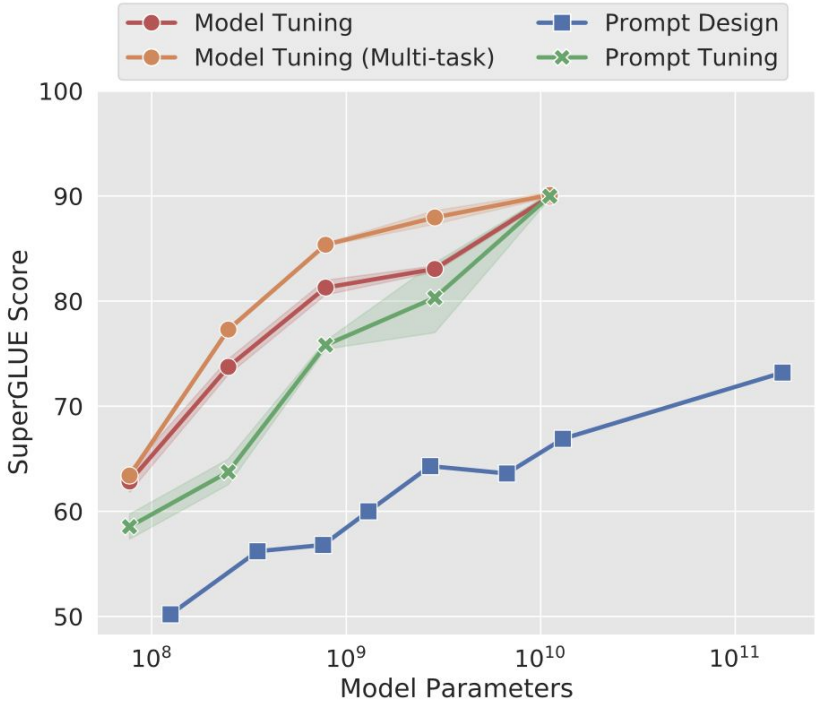
More qualitative analysis can be found in the paper.

The Power of Scale for Parameter-Efficient Prompt Tuning

Qihang Fang, Jiannan Wang

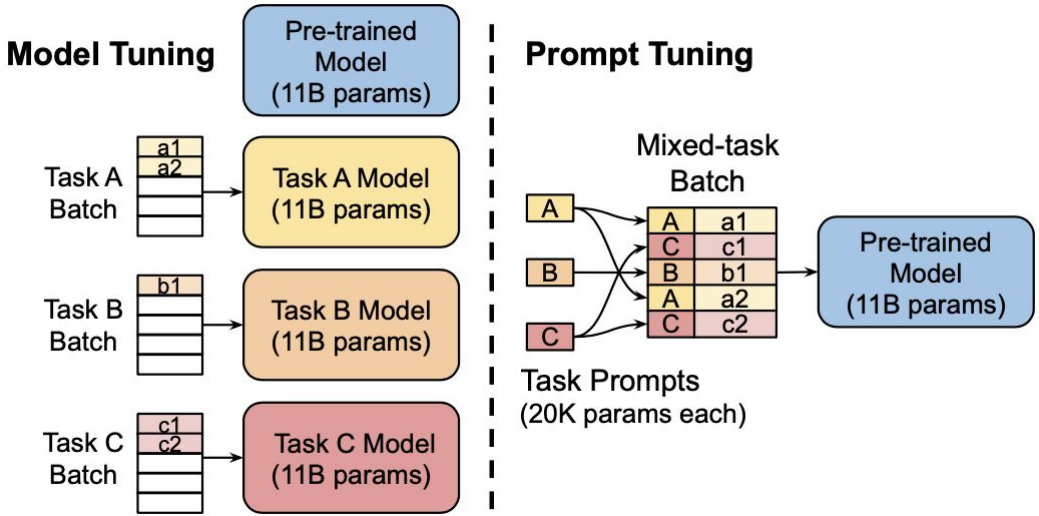
Introduction

- Model tuning
  - Separate copy of the model
- Prompt design
  - Prepend instructions and few example to input
- Prompt design lags far behind
  - Error-prone
  - Human involvement
  - Effectiveness



Prompt tuning

- Use a fixed prompt of special tokens, where only the embeddings of these prompt tokens can be updated
- Tuned Token: Soft prompt

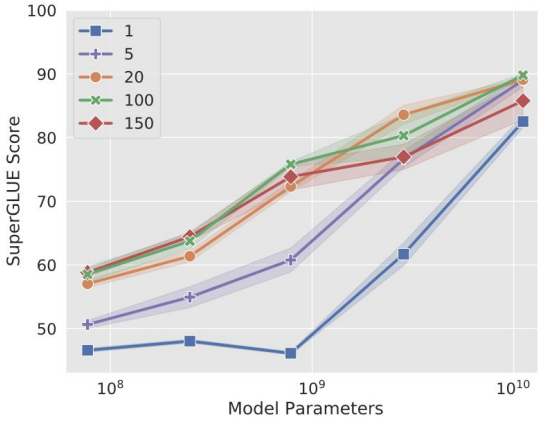


Prompt tuning

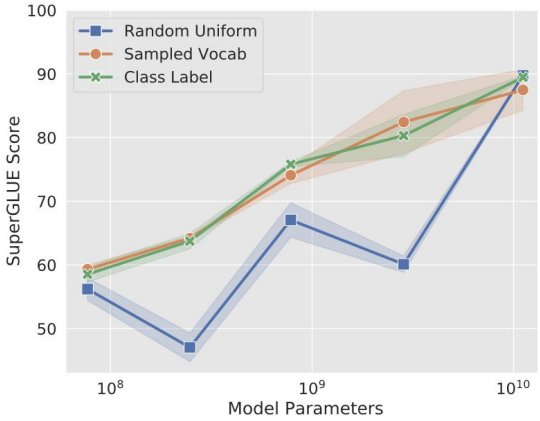
- Design Decisions
  - Initialization
    - Random
    - Sampled Vocabulory
    - Class Label (For Classification)
  - Length of the prompt: the shorter, the fewer
- Unlearn Span Corruption
  - T5 (mask span)
    - Input: Thank you <X> me to your party <Y> week
    - Output: <X> for inviting <Y> last <Z>
  - Span Corruption / Span Corruption+Sentinel / LM Adaptation

Ablation Study

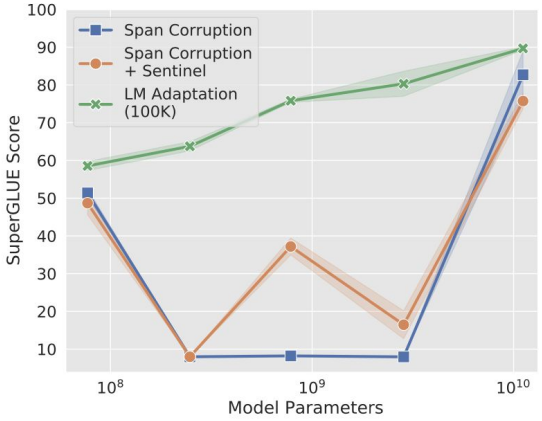
- Prompt Length
- Prompt initialization
- Pre-training objective



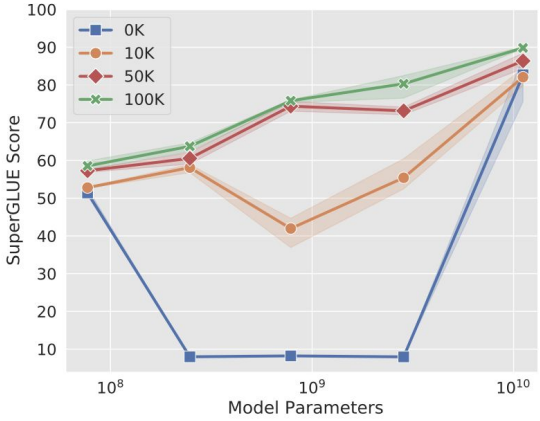
(a) Prompt length



(b) Prompt initialization



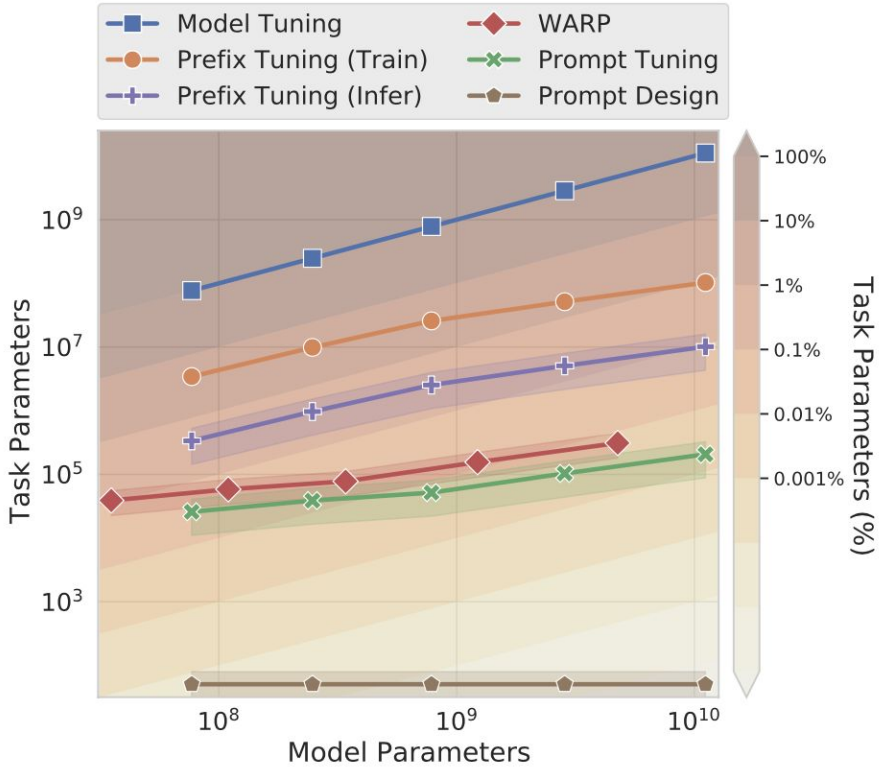
(c) Pre-training method



(d) LM adaptation steps

Comparison to Similar Approaches

-



Resilience to Domain Shift

- prompt representations indirectly modulate the representation of the input
  - reduce the model's ability to overfit
- Tends to give stronger zero-shot performance than model tuning

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

Train	Eval	Tuning	Accuracy	F1
QQP	MRPC	Model	73.1 ±0.9	81.2 ±2.1
		Prompt	**76.3** ±0.1	**84.3** ±0.3
MRPC	QQP	Model	74.9 ±1.3	**70.9** ±1.2
		Prompt	**75.4** ±0.8	69.7 ±0.3

Prompt Ensembling

- **Ensembles** of neural models trained from different initializations on the same data are widely observed to improve task performance
- Prompt tuning provides a more efficient way to **ensemble** multiple adaptations of a pre-trained language model

Dataset	Metric	Average	Best	Ensemble
BoolQ	acc.	91.1	91.3	**91.7**
CB	acc./F1	99.3 / 99.0	100.00 / 100.00	**100.0 / 100.0**
COPA	acc.	98.8	100.0	**100.0**
MultiRC	EM/F1$_a$	65.7 / 88.7	66.3 / 89.0	**67.1 / 89.4**
ReCoRD	EM/F1	92.7 / 93.4	92.9 / 93.5	**93.2 / 93.9**
RTE	acc.	92.6	**93.5**	**93.5**
WiC	acc.	76.2	76.6	**77.4**
WSC	acc.	95.8	**96.2**	**96.2**
SuperGLUE (dev)		90.5	91.0	**91.3**

- The ensemble beats the single-prompt average & matches the best individual prompt.

Interpretability

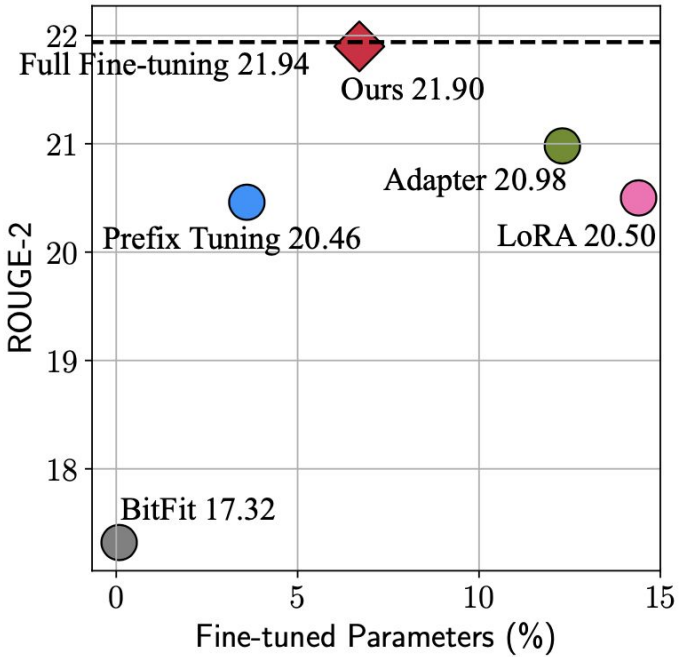
-

Towards a Unified View of Parameter-Efficient Transfer Learning

Qihang Fang, Jiannan Wang

Introduction

- Transfer Learning from pre-trained language models (PLMs) is now the prevalent paradigm in natural language processing



Introduction

- Adapter

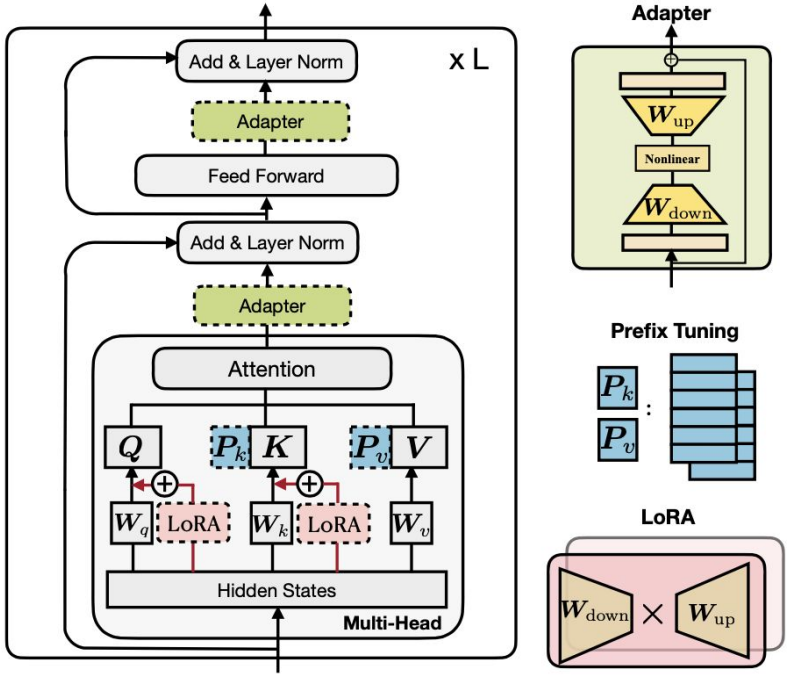
$$\boldsymbol{h} \leftarrow \boldsymbol{h} + f(\boldsymbol{h}\boldsymbol{W}_{\text{down}})\boldsymbol{W}_{\text{up}}$$

- LoRA

$$\boldsymbol{h} \leftarrow \boldsymbol{h} + s \cdot \boldsymbol{x}\boldsymbol{W}_{\text{down}}\boldsymbol{W}_{\text{up}}$$

- Prefix Tuning

$$\text{head}_i = \text{Attn}(\boldsymbol{x}\boldsymbol{W}_q^{(i)}, \text{concat}(\boldsymbol{P}_k^{(i)}, \boldsymbol{C}\boldsymbol{W}_k^{(i)}), \text{concat}(\boldsymbol{P}_v^{(i)}, \boldsymbol{C}\boldsymbol{W}_v^{(i)}))$$



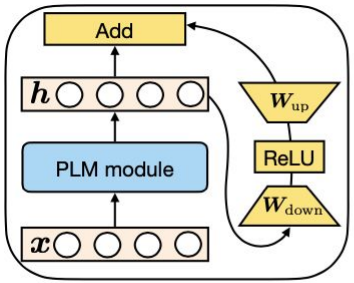
Question

- How are these methods connected?
- Do these methods share design elements that are essential for their effectiveness, and what are they?
- Can the effective ingredients of each method be transferred to others to yield more effective variants?

Prefix-Tuning vs Adapter

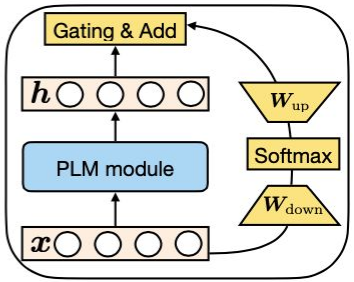
$$
\begin{aligned}
\text{head} &= \text{Attn}(\boldsymbol{x}\boldsymbol{W}_q, \text{concat}(\boldsymbol{P}_k, \boldsymbol{C}\boldsymbol{W}_k), \text{concat}(\boldsymbol{P}_v, \boldsymbol{C}\boldsymbol{W}_v)) \\
&= \text{softmax}\big(\boldsymbol{x}\boldsymbol{W}_q \text{concat}(\boldsymbol{P}_k, \boldsymbol{C}\boldsymbol{W}_k)^\top\big) \begin{bmatrix} \boldsymbol{P}_v \\ \boldsymbol{C}\boldsymbol{W}_v \end{bmatrix} \\
&= (1-\lambda(\boldsymbol{x}))\text{softmax}(\boldsymbol{x}\boldsymbol{W}_q\boldsymbol{W}_k^\top\boldsymbol{C}^\top)\boldsymbol{C}\boldsymbol{W}_v + \lambda(\boldsymbol{x})\text{softmax}(x\boldsymbol{W}_q\boldsymbol{P}_k^\top)\boldsymbol{P}_v \\
&= (1-\lambda(\boldsymbol{x}))\underbrace{\text{Attn}(\boldsymbol{x}\boldsymbol{W}_q, \boldsymbol{C}\boldsymbol{W}_k, \boldsymbol{C}\boldsymbol{W}_v)}_{\text{standard attention}} + \lambda(\boldsymbol{x})\underbrace{\text{Attn}(\boldsymbol{x}\boldsymbol{W}_q, \boldsymbol{P}_k, \boldsymbol{P}_v)}_{\text{independent of } \boldsymbol{C}},
\end{aligned}
$$

$$
\boldsymbol{h} \leftarrow (1-\lambda(\boldsymbol{x}))\boldsymbol{h} + \lambda(\boldsymbol{x})\Delta\boldsymbol{h}, \quad \Delta\boldsymbol{h} := \text{softmax}(\boldsymbol{x}\boldsymbol{W}_q\boldsymbol{P}_k^\top)\boldsymbol{P}_v
$$



(a) Adapter

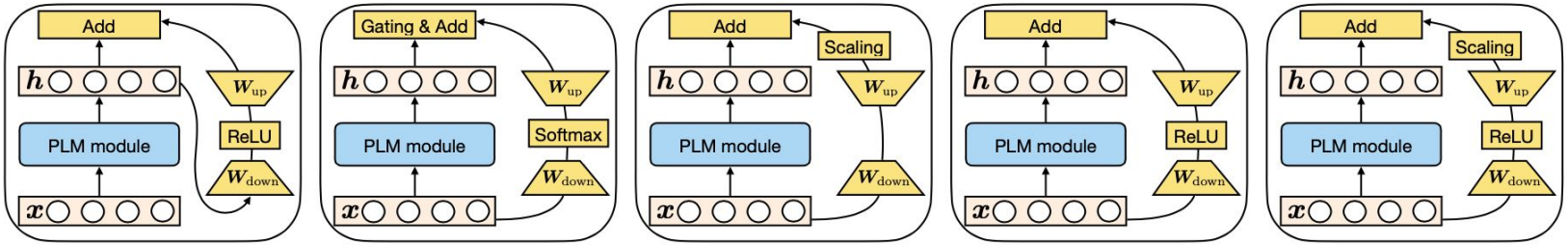
$$
\boldsymbol{h} \leftarrow (1-\lambda(\boldsymbol{x}))\boldsymbol{h} + \lambda(\boldsymbol{x})f(\boldsymbol{x}\boldsymbol{W}_1)\boldsymbol{W}_2
$$



(b) Prefix Tuning

Unified Framework

Method	Δh functional form	insertion form	modified representation	composition function
		Existing Methods		
Prefix Tuning	$\text{softmax}(xW_q P_k^\top)P_v$	parallel	head attn	$h \leftarrow (1-\lambda)h + \lambda\Delta h$
Adapter	$\text{ReLU}(hW_{\text{down}})W_{\text{up}}$	sequential	ffn/attn	$h \leftarrow h + \Delta h$
LoRA	$xW_{\text{down}}W_{\text{up}}$	parallel	attn key/val	$h \leftarrow h + s \cdot \Delta h$
		Proposed Variants		
Parallel adapter	$\text{ReLU}(hW_{\text{down}})W_{\text{up}}$	parallel	ffn/attn	$h \leftarrow h + \Delta h$
Muti-head parallel adapter	$\text{ReLU}(hW_{\text{down}})W_{\text{up}}$	parallel	head attn	$h \leftarrow h + \Delta h$
Scaled parallel adapter	$\text{ReLU}(hW_{\text{down}})W_{\text{up}}$	parallel	ffn/attn	$h \leftarrow h + s \cdot \Delta h$



(a) Adapter (b) Prefix Tuning (c) LoRA (d) Parallel Adapter (e) Scaled PA

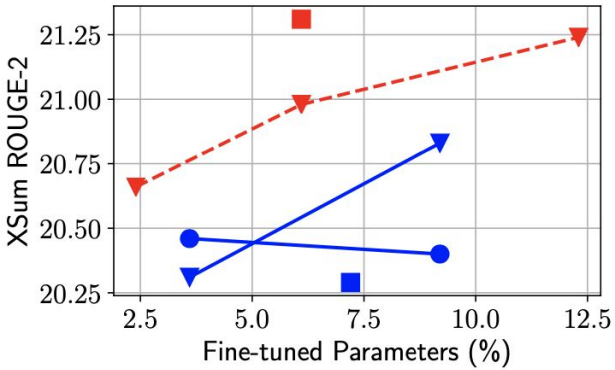
Which dimensions are important?

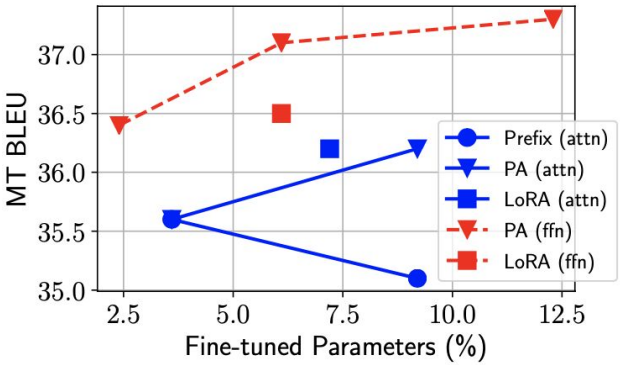
- **Which insertion form Sequential / Parallel?**
- Which modified representation Attention / FFN?
- Which composition function?

Method	# params	XSum (R-1/2/L)	MT (BLEU)
Prefix, l=200	3.6%	43.40/20.46/35.51	35.6
SA (attn), r=200	3.6%	42.01/19.30/34.40	35.3
SA (ffn), r=200	2.4%	43.21/19.98/35.08	35.6
PA (attn), r=200	3.6%	43.58/20.31/35.34	35.6
PA (ffn), r=200	2.4%	**43.93/20.66/35.63**	**36.4**

Which dimensions are important?

- Which insertion form Sequential / Parallel?
- **Which modified representation Attention / FFN?**
- Which composition function?





Method	# params	MT (BLEU)
PA (attn), r=200	3.6%	35.6
Prefix, l=200	3.6%	35.6
MH PA (attn), r=200	3.6%	35.8
Prefix, l=30	0.1%	35.2
-gating, l=30	0.1%	34.9
PA (ffn), r=30	0.1%	33.0
PA (attn), r=30	0.1%	33.7
MH PA (attn), r=30	0.1%	**35.3**

Which dimensions are important?

- Which insertion form Sequential / Parallel?
- Which modified representation Attention / FFN?
- **Which composition function?**

Method (# params)	XSum (R-1/2/LSum)
LoRA (6.1%), s=4	44.59/21.31/36.25
LoRA (6.1%), s=1	44.17/20.83/35.74
PA (6.1%)	44.35/20.98/35.98
Scaled PA (6.1%), s=4	**44.85/21.54/36.58**
Scaled PA (6.1%), trainable s	44.56/21.31/36.29

MAM Adapter

- An effective integration by transferring favorable design elements

Method	# params	XSum (R-1/2/L)	MT (BLEU)
Full fine-tuning†	100%	45.14/22.27/37.25	37.7
Full fine-tuning (our run)	100%	44.81/21.94/36.83	37.3
Bitfit (Ben Zaken et al., 2021)	0.1%	40.64/17.32/32.19	26.4
Prompt tuning (Lester et al., 2021)	0.1%	38.91/15.98/30.83	21.0
Prefix tuning (Li & Liang, 2021), l=200	3.6%	43.40/20.46/35.51	35.6
Pfeiffer adapter (Pfeiffer et al., 2021), r=600	7.2%	$44.03/20.89/35.89_{\pm.13/.10/.08}$	$36.9_{\pm.1}$
LoRA (ffn), r=102	7.2%	$44.53/21.29/36.28_{\pm.14/.07/.10}$	$36.8_{\pm.3}$
Parallel adapter (PA, ffn), r=1024	12.3%	$44.71/21.41/36.41_{\pm.16/.17/.16}$	$37.2_{\pm.1}$
PA (attn, r=30) + PA (ffn, r=512)	6.7%	$44.29/21.06/36.12_{\pm.31/.19/.18}$	$37.2_{\pm.1}$
Prefix tuning (attn, l=30) + LoRA (ffn, r=102)	6.7%	$44.84/21.71/36.77_{\pm.07/.05/.03}$	$37.0_{\pm.1}$
MAM Adapter (our variant, l=30, r=512)	6.7%	$\mathbf{45.06/21.90/36.87}_{\pm.08/.01/.04}$	$\mathbf{37.5}_{\pm.1}$

Summary

- Study three SOTA methods (LoRA, Adapter, Prefix Tuning)
- Propose a unified parameter efficient transfer learning framework
- Through comparative experiments, construct a new SOTA method

Discussion

- Can prompt tuning be used with other techniques (e.g., fine-tuning) to further improve the performance of language models?
- Can fine-tuning only the prompt-related parameters match or even exceed the performance of full fine-tuning with **a small amount of data**?
- Before QLoRA, 4-bit quantization can not be used in training process due to **the degradation of precision**. Will 8-bit quantization and 16 bit quantization meet the same issue?

That’s ALL
Q&A