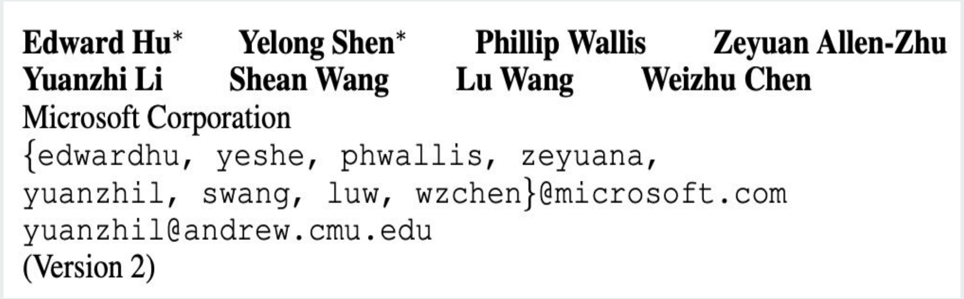
LoRA: Low-Rank Adaptation of Large Language Models



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)

Presented by Neel Kothari and Tanish Patwa

Introduction

- The major downside of fine-tuning LLMs for downstream tasks: the new model contains as many parameters as in the original model.
- As larger models are trained every few months, this changes from a mere “inconvenience” to a critical deployment challenge
- General solution: Selective adaptation of task specific parameters
- Problem: Increased inference latency due to increase in model depth or reduction of usable input sequence length

Suggested solution:

LoRA (Low Rank Adaptation Approach)

- LoRA allows us to train some dense layers in a neural network indirectly by optimizing rank decomposition matrices of the change in dense layers during adaptation instead, while keeping the pre-trained weights frozen.
- Hypothesis: change in weights during model adaptation also has a low “intrinsic rank”

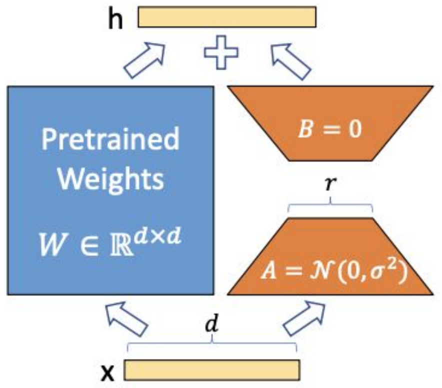

h
Pretrained Weights
$W \in \mathbb{R}^{d \times d}$
$B = 0$
r
$A = \mathcal{N}(0, \sigma^2)$
d
x

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

Key advantages

- Transferable: We can freeze the shared model and efficiently switch tasks by replacing the matrices A and B, reducing the storage requirement and task-switching overhead significantly.
- LoRA lowers the hardware barrier up to 3 times when using adaptive optimizers.
- Trainable matrices merged with the frozen weights when deployed, *introducing no inference latency* compared to a fully fine-tuned model.
- LoRA is orthogonal to many prior methods and can be combined with many of them, such as prefix-tuning.

Problem Statement

$$\max_{\Phi} \sum_{(x,y)\in\mathcal{Z}} \sum_{t=1}^{|y|} \log\left(P_{\Phi}(y_t|x, y_{<t})\right) \quad (1)$$

$$\max_{\Theta} \sum_{(x,y)\in\mathcal{Z}} \sum_{t=1}^{|y|} \log\left(p_{\Phi_0+\Delta\Phi(\Theta)}(y_t|x, y_{<t})\right) \quad (2)$$

Drawback of (1) - For full fine-tuning, for *each* downstream task, we learn a *different* set of parameters ΔΦ whose dimension |ΔΦ| equals |Φ0|

Thus a more parameter-efficient approach is used, where the task-specific parameter increment ΔΦ = ΔΦ(Θ) is further encoded by a much smaller-sized set of parameters Θ with |Θ| ≪ |Φ0|. The task of finding ΔΦ thus becomes optimizing over Θ

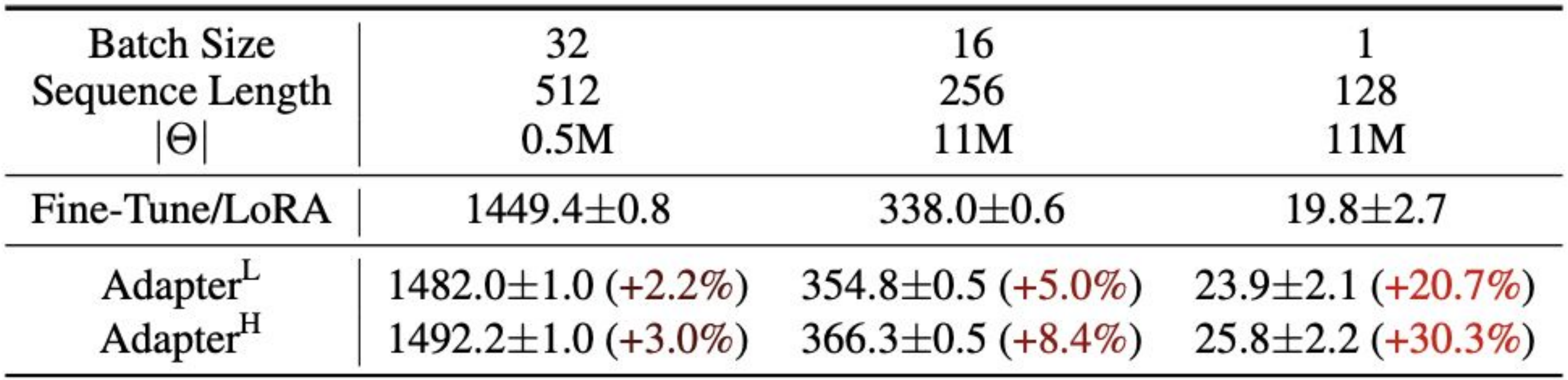
Aren’t Existing Solutions Good Enough ?

Adapter layers Introduce Inference Latency -

- Processed sequentially
- Each layer’s output depends on completing the adapter’s operations before passing to the next transformer layer. While individual operations within the adapter can be parallelized, the adapter’s output must be fully computed before the main model can continue.
- Creates bottleneck

Optimizing Prompt - Prefix Tuning

- During training, the model processes both the prefix and the original input sequence as a single extended input. The trainable prefix embeddings are optimized alongside the task, which allows it to "learn" how to shift the model's attention and responses appropriately.
- performance changes non-monotonically



Batch Size	32	16	1
Sequence Length	512	256	128
$\|\Theta\|$	0.5M	11M	11M
Fine-Tune/LoRA	1449.4±0.8	338.0±0.6	19.8±2.7
Adapter^{L}	1482.0±1.0 (+2.2%)	354.8±0.5 (+5.0%)	23.9±2.1 (+20.7%)
Adapter^{H}	1492.2±1.0 (+3.0%)	366.3±0.5 (+8.4%)	25.8±2.2 (+30.3%)

Table 1: Infernece latency of a single forward pass in GPT-2 medium measured in milliseconds,

Methodology

Given: $W_0 \in \mathbb{R}^{d \times k}$

The weight update is represented as: $\Delta W = BA$ where

- $B \in \mathbb{R}^{d \times r}$
- $A \in \mathbb{R}^{r \times k}$
- $r \ll \min(d, k)$

The updated output can be expressed as: $h = W_0 x + BAx$

The rank r of the matrix refers to the number of linearly independent components, establishing a low-rank representation of the weight updates. This approach enables efficient updates while capturing essential variations in the weight matrix.

Initialization

$$A \rightarrow \text{Gaussian}, \quad B \rightarrow 0$$

Training

Scale ΔW by

$$h = W_0 x + \frac{\alpha}{r}(BA)x$$

Adam optimizer

Scaling Factor - $\frac{\text{lora\_alpha}}{\text{lora\_rank}}$

Rank-Stabilized LoRA (rsLoRA) - $\frac{\text{lora\_alpha}}{\sqrt{\text{lora\_rank}}}$

Advantages

1. No Additional Inference latency
   a. Since LoRA keeps the original weight matrix frozen during adaptation, the model can easily revert back to its original state by simply ignoring the low-rank updates.
   b. Because LoRA introduces only a small number of additional parameters it allows for rapid re-adaptation to new tasks or changes in the dataset without the need for extensive retraining.
2. Practical Benefits
   a. LoRA provides a **25% speedup** during training on GPT-3 (175 billion parameters) compared to full fine-tuning.
   b. For a large Transformer like GPT-3 (175 billion parameters), VRAM consumption is reduced from **1.2 TB to 350 GB**.

Empirical Experiments

Baselines and Abbreviations:

- **Fine-Tuning (FT)** : During fine-tuning, the model is initialized to the pre-trained weights and biases, and all model parameters undergo gradient updates
- **FTTop2:** adapts just the last two layers
- Bias-only or **BitFit** is a baseline where we only train the bias vectors while freezing everything else.

- Prefix-embedding tuning (**PreEmbed**) inserts special tokens among the input tokens. These special tokens have trainable word embeddings and are generally not in the model's vocabulary.

- Prefix-layer tuning (**PreLayer**): is an extension to prefix-embedding tuning + learning the activations after every Transformer layer

- **Adapter tuning:** inserts adapter layers between the self- attention module (and the MLP module) and the subsequent residual connection.
- **AdapterH**: There are two fully connected layers with biases in an adapter layer with a nonlinearity in between.
- **AdapterL**: adapter layer applied only after the MLP module and after a LayerNorm
- **AdapterD**: AdapterDrop, which drops some adapter layers
- **LoRA:** adds trainable pairs of rank decomposition matrices in parallel to existing weight matrices.

Models experimented with

RoBERTA BASE/LARGE: GLUE Benchmark

DeBERTA XXL: GLUE Benchmark

GPT-2 MEDIUM/LARGE: E2G NLG Challenge, DART, WebNLG

GPT-3 175B: WikiSQL, MNLI-m, SAMSum

Dataset Details

GLUE Benchmark: includes MNLI, SST-2, MRPC, CoLA, QNLI, QQP, RTE, and STS-B

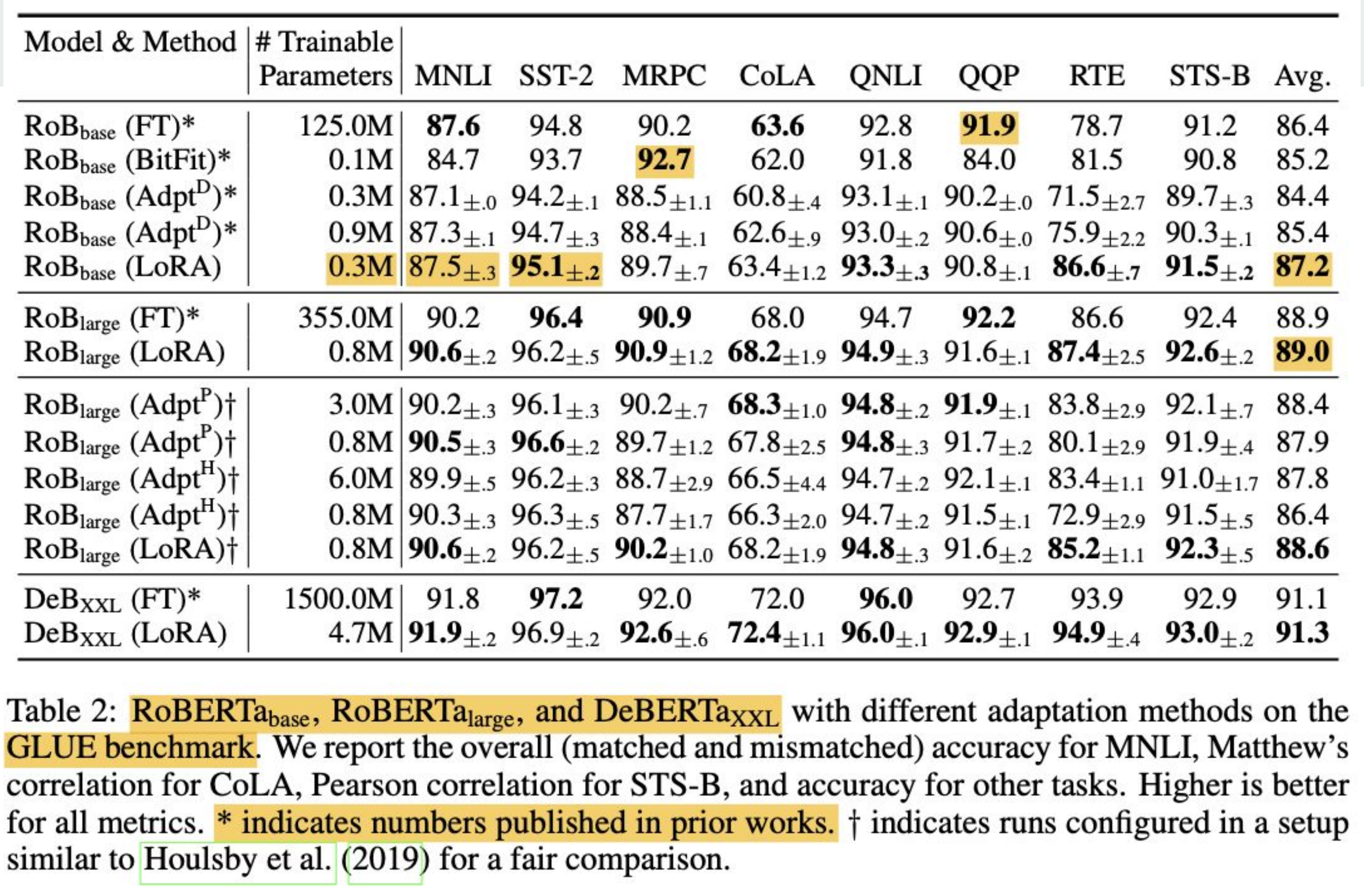
WikiSQL: table schema+question and SQL query pairs.

SAMSum: Conversation and abstractive summary pairs

E2E NLG Challenge: Contains key-value pairs and human-written reference texts

DART: ENTITY — RELATION — ENTITY triples

WebNLG: SUBJECT — PROPERTY — OBJECT triples



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

Table 2: RoBERTa$_{\text{base}}$, RoBERTa$_{\text{large}}$, and DeBERTa$_{\text{XXL}}$ with different adaptation methods on the GLUE benchmark. We report the overall (matched and mismatched) accuracy for MNLI, Matthew's correlation for CoLA, Pearson correlation for STS-B, and accuracy for other tasks. Higher is better for all metrics. * indicates numbers published in prior works. † indicates runs configured in a setup similar to Houlsby et al. (2019) for a fair comparison.

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$^{\text{Top2}}$)*	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	$\mathbf{70.4}_{\pm.1}$	$\mathbf{8.89}_{\pm.02}$	$\mathbf{46.8}_{\pm.2}$	$\mathbf{72.0}_{\pm.2}$	$2.47_{\pm.02}$

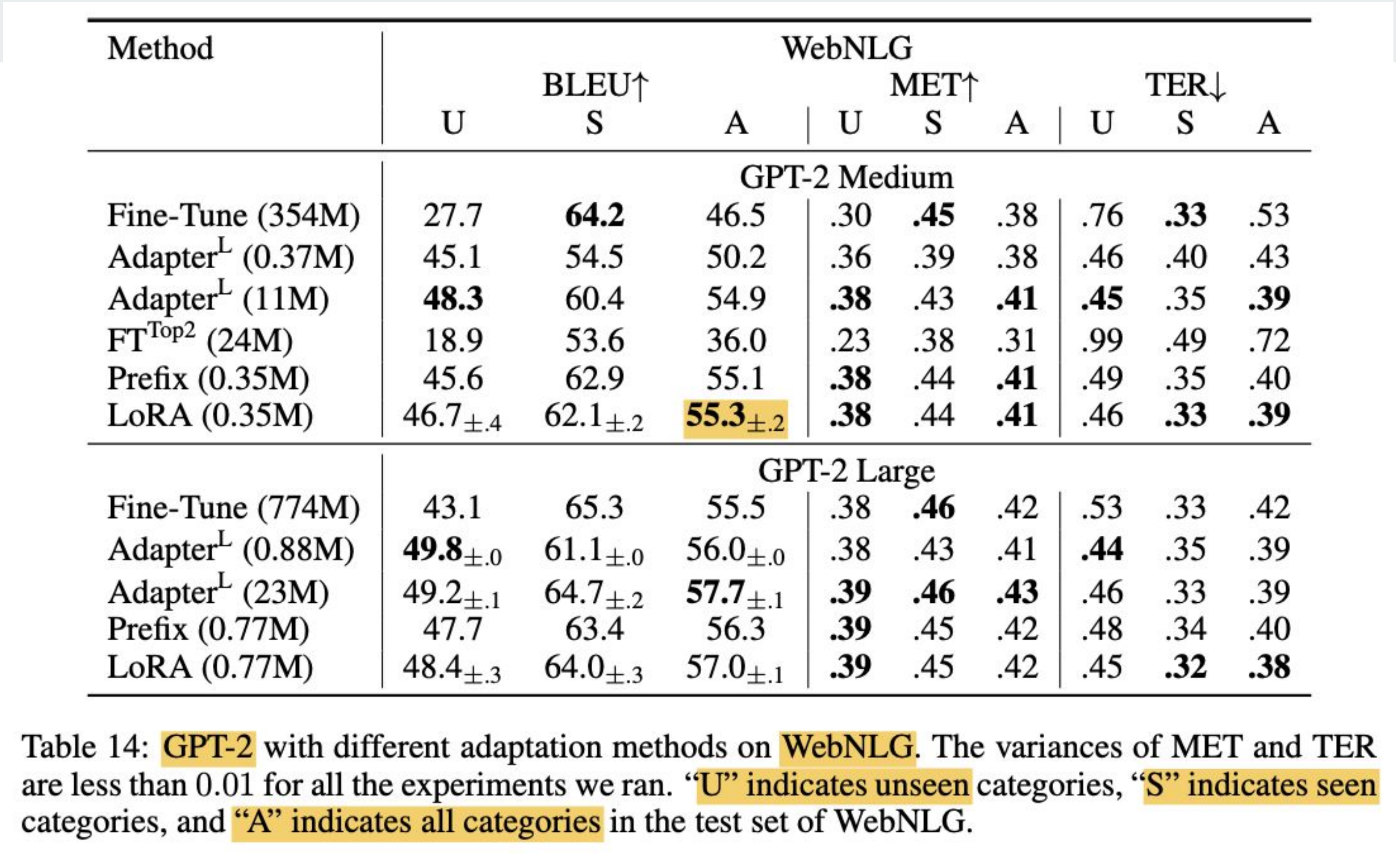
Table 3: GPT-2 medium (M) and large (L) with different adaptation methods on the E2E NLG Challenge. For all metrics, higher is better. LoRA outperforms several baselines with comparable or fewer trainable parameters. Confidence intervals are shown for experiments we ran. * indicates numbers published in prior works.

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 (Adapter^{H})	7.1M	71.9	89.8	53.0/28.9/44.8
GPT-3 (Adapter^{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

Table 4: Performance of different adaptation methods on GPT-3 175B. We report the logical form validation accuracy on WikiSQL, validation accuracy on MultiNLI-matched, and Rouge-1/2/L on SAMSum. LoRA performs better than prior approaches, including full fine-tuning. The results on WikiSQL have a fluctuation around $\pm0.5\%$, MNLI-m around $\pm0.1\%$, and SAMSum around $\pm0.2/\pm0.2/\pm0.1$ for the three metrics.

Method	# Trainable Parameters	DART BLEU↑	DART MET↑	DART TER↓
		GPT-2 Medium		
Fine-Tune	354M	46.2	**0.39**	**0.46**
Adapter^{L}	0.37M	42.4	0.36	0.48
Adapter^{L}	11M	45.2	0.38	**0.46**
FT^{Top2}	24M	41.0	0.34	0.56
PrefLayer	0.35M	46.4	0.38	**0.46**
LoRA	0.35M	**47.1**$_{\pm.2}$	**0.39**	**0.46**
		GPT-2 Large		
Fine-Tune	774M	47.0	**0.39**	0.46
Adapter^{L}	0.88M	45.7$_{\pm.1}$	0.38	0.46
Adapter^{L}	23M	47.1$_{\pm.1}$	**0.39**	**0.45**
PrefLayer	0.77M	46.7	0.38	**0.45**
LoRA	0.77M	**47.5**$_{\pm.1}$	**0.39**	**0.45**

Table 13: GPT-2 with different adaptation methods on DART. The variances of MET and TER are less than 0.01 for all adaption approaches.



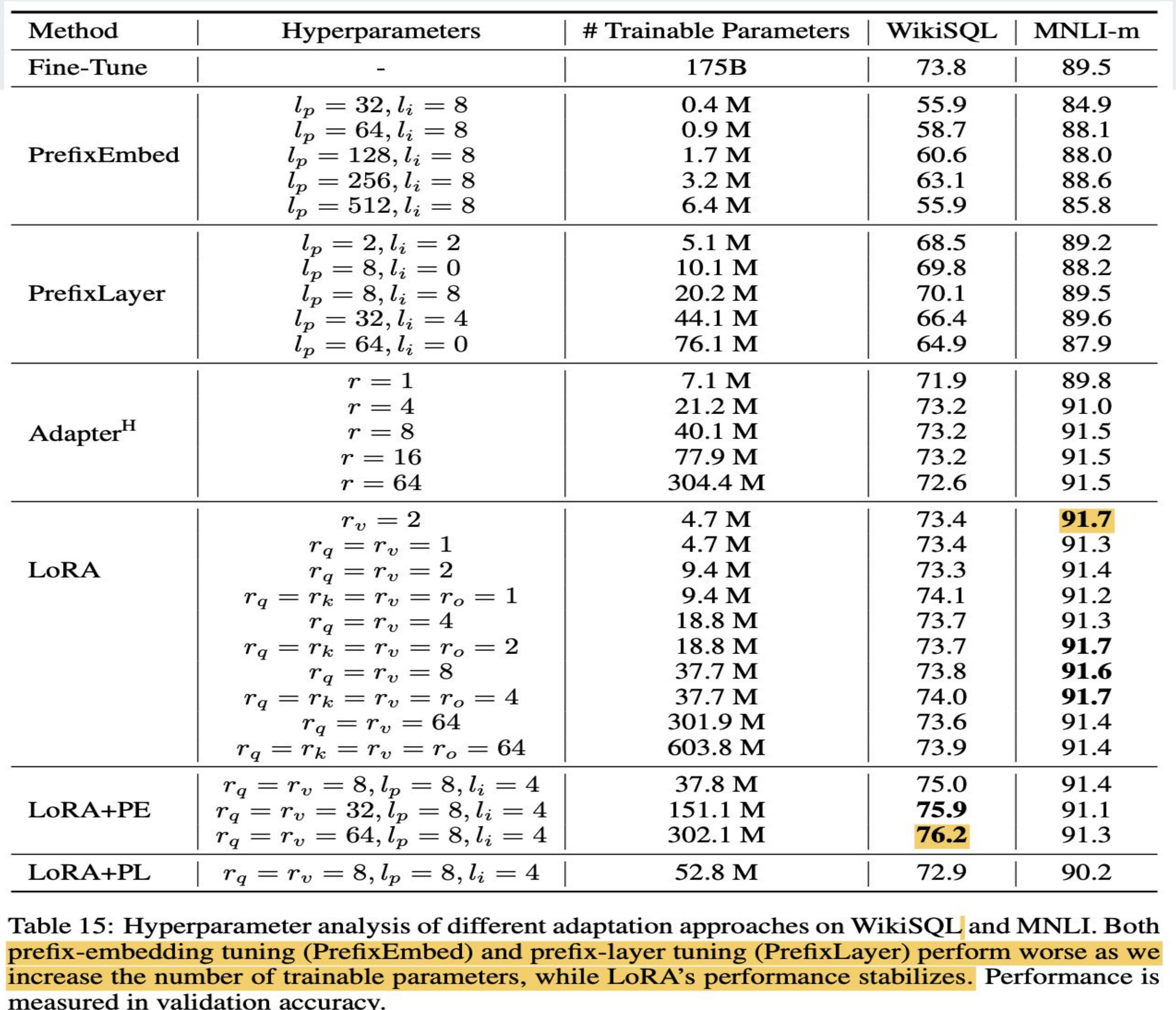
Method	WebNLG								
	BLEU↑			MET↑			TER↓		
	U	S	A	U	S	A	U	S	A
	GPT-2 Medium								
Fine-Tune (354M)	27.7	**64.2**	46.5	.30	**.45**	.38	.76	**.33**	.53
Adapter^{L} (0.37M)	45.1	54.5	50.2	.36	.39	.38	.46	.40	.43
Adapter^{L} (11M)	**48.3**	60.4	54.9	**.38**	.43	**.41**	**.45**	.35	**.39**
FT$^{\text{Top2}}$ (24M)	18.9	53.6	36.0	.23	.38	.31	.99	.49	.72
Prefix (0.35M)	45.6	62.9	55.1	**.38**	.44	**.41**	.49	.35	.40
LoRA (0.35M)	$46.7_{\pm.4}$	$62.1_{\pm.2}$	$\mathbf{55.3}_{\pm.2}$	**.38**	.44	**.41**	.46	**.33**	**.39**
	GPT-2 Large								
Fine-Tune (774M)	43.1	65.3	55.5	.38	**.46**	.42	.53	.33	.42
Adapter^{L} (0.88M)	$\mathbf{49.8}_{\pm.0}$	$61.1_{\pm.0}$	$56.0_{\pm.0}$	.38	.43	.41	**.44**	.35	.39
Adapter^{L} (23M)	$49.2_{\pm.1}$	$64.7_{\pm.2}$	$\mathbf{57.7}_{\pm.1}$	**.39**	**.46**	**.43**	.46	.33	.39
Prefix (0.77M)	47.7	63.4	56.3	**.39**	.45	.42	.48	.34	.40
LoRA (0.77M)	$48.4_{\pm.3}$	$64.0_{\pm.3}$	$57.0_{\pm.1}$	**.39**	.45	.42	.45	**.32**	**.38**

Table 14: GPT-2 with different adaptation methods on WebNLG. The variances of MET and TER are less than 0.01 for all the experiments we ran. "U" indicates unseen categories, "S" indicates seen categories, and "A" indicates all categories in the test set of WebNLG.

Prefix tuning+LoRA

- LoRA+PrefixEmbed (LoRA+PE) combines LoRA with prefix-embedding tuning, where we tokens are inserted whose embeddings are treated as trainable parameters.

- LoRA+PrefixLayer (LoRA+PL) combines LoRA with prefix-layer tuning. Here, instead of letting the hidden representations of the inserted tokens evolve naturally, they are replaced after every Transformer block with an input agnostic vector. Thus, both the embeddings and subsequent Transformer block activations are treated as trainable parameters.



Method	Hyperparameters	# Trainable Parameters	WikiSQL	MNLI-m
Fine-Tune	-	175B	73.8	89.5
PrefixEmbed	$l_p = 32, l_i = 8$	0.4 M	55.9	84.9
	$l_p = 64, l_i = 8$	0.9 M	58.7	88.1
	$l_p = 128, l_i = 8$	1.7 M	60.6	88.0
	$l_p = 256, l_i = 8$	3.2 M	63.1	88.6
	$l_p = 512, l_i = 8$	6.4 M	55.9	85.8
PrefixLayer	$l_p = 2, l_i = 2$	5.1 M	68.5	89.2
	$l_p = 8, l_i = 0$	10.1 M	69.8	88.2
	$l_p = 8, l_i = 8$	20.2 M	70.1	89.5
	$l_p = 32, l_i = 4$	44.1 M	66.4	89.6
	$l_p = 64, l_i = 0$	76.1 M	64.9	87.9
Adapter^{H}	$r = 1$	7.1 M	71.9	89.8
	$r = 4$	21.2 M	73.2	91.0
	$r = 8$	40.1 M	73.2	91.5
	$r = 16$	77.9 M	73.2	91.5
	$r = 64$	304.4 M	72.6	91.5
LoRA	$r_v = 2$	4.7 M	73.4	**91.7**
	$r_q = r_v = 1$	4.7 M	73.4	91.3
	$r_q = r_v = 2$	9.4 M	73.3	91.4
	$r_q = r_k = r_v = r_o = 1$	9.4 M	74.1	91.2
	$r_q = r_v = 4$	18.8 M	73.7	91.3
	$r_q = r_k = r_v = r_o = 2$	18.8 M	73.7	**91.7**
	$r_q = r_v = 8$	37.7 M	73.8	**91.6**
	$r_q = r_k = r_v = r_o = 4$	37.7 M	74.0	**91.7**
	$r_q = r_v = 64$	301.9 M	73.6	91.4
	$r_q = r_k = r_v = r_o = 64$	603.8 M	73.9	91.4
LoRA+PE	$r_q = r_v = 8, l_p = 8, l_i = 4$	37.8 M	75.0	91.4
	$r_q = r_v = 32, l_p = 8, l_i = 4$	151.1 M	**75.9**	91.1
	$r_q = r_v = 64, l_p = 8, l_i = 4$	302.1 M	**76.2**	91.3
LoRA+PL	$r_q = r_v = 8, l_p = 8, l_i = 4$	52.8 M	72.9	90.2

Table 15: Hyperparameter analysis of different adaptation approaches on WikiSQL and MNLI. Both prefix-embedding tuning (PrefixEmbed) and prefix-layer tuning (PrefixLayer) perform worse as we increase the number of trainable parameters, while LoRA's performance stabilizes. Performance is measured in validation accuracy.

INFERENCE LATENCY INTRODUCED BY ADAPTER LAYERS

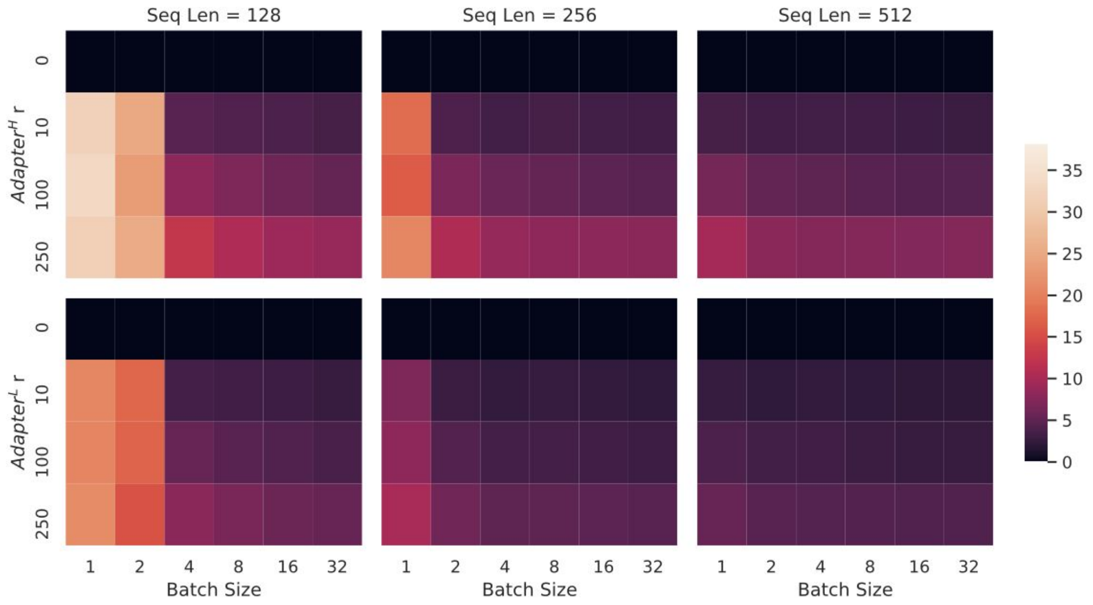
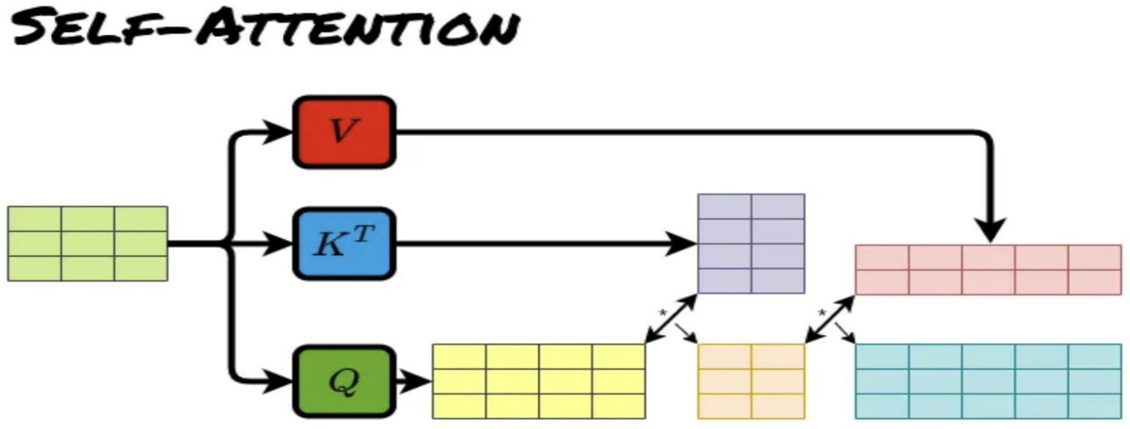


Figure 5: Percentage slow-down of inference latency compared to the no-adapter ($r = 0$) baseline. The top row shows the result for Adapter$^{\text{H}}$ and the bottom row Adapter$^{\text{L}}$. Larger batch size and sequence length help to mitigate the latency, but the slow-down can be as high as over 30% in an online, short-sequence-length scenario. We tweak the colormap for better visibility.

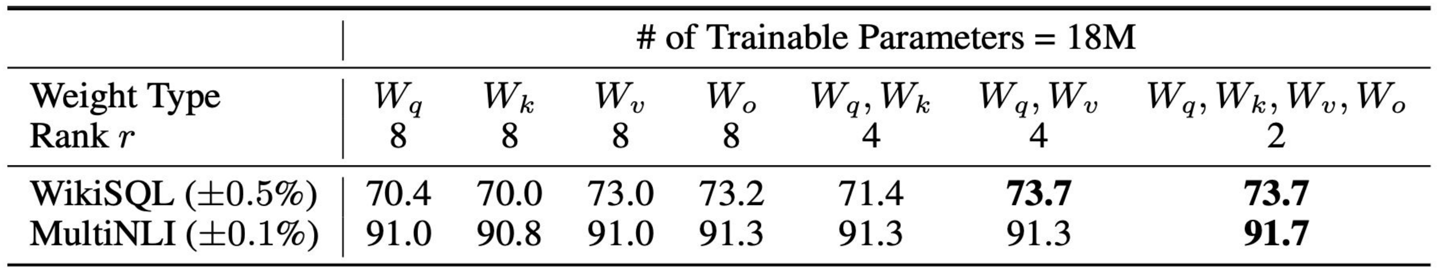
Transformers



github.com/tensorops/TransformerX
soran-ghaderi.github.io
github.com/soran-ghaderi

Understanding Low- Rank Updates

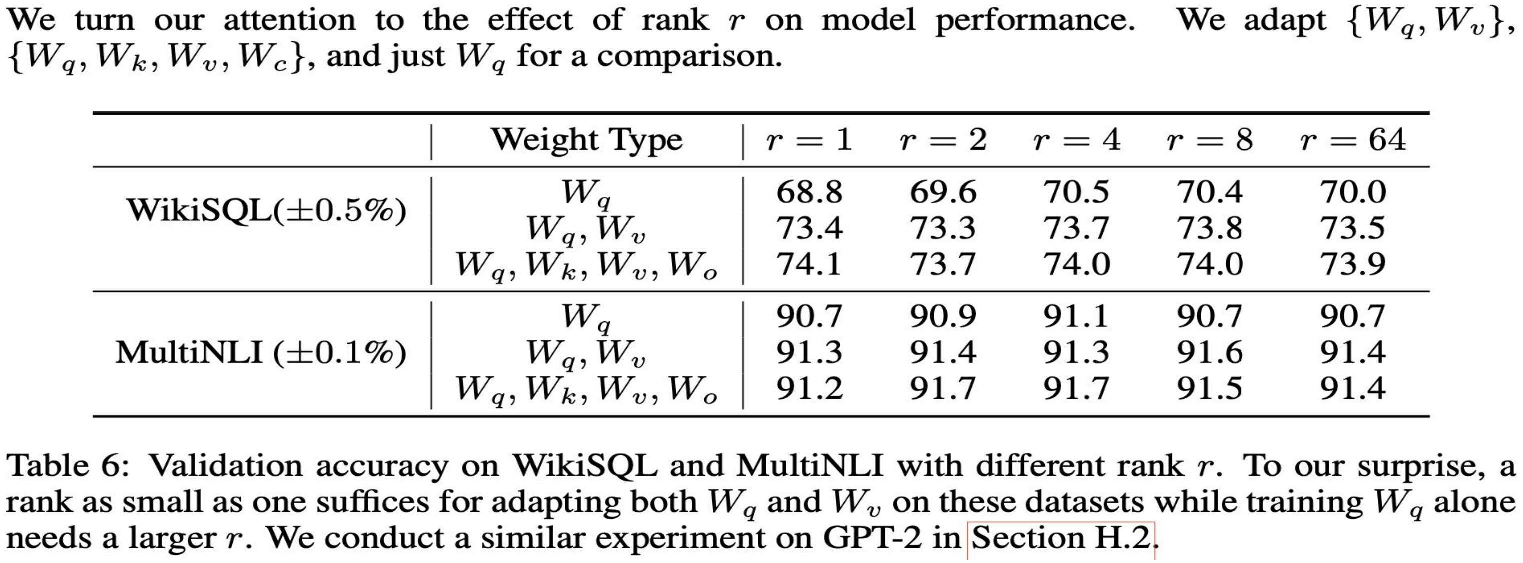
Given a parameter budget constraint, *which subset of weight matrices* in a pre-trained Transformer should we adapt to maximize downstream performance?



	# 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.

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



We turn our attention to the effect of rank r on model performance. We adapt $\{W_q, W_v\}$, $\{W_q, W_k, W_v, W_c\}$, and just W_q for a comparison.

	Weight Type	$r = 1$	$r = 2$	$r = 4$	$r = 8$	$r = 64$
WikiSQL($\pm0.5\%$)	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 ($\pm0.1\%$)	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.

Rank r	val_loss	BLEU	NIST	METEOR	ROUGE_L	CIDEr
1	1.23	68.72	8.7215	0.4565	0.7052	2.4329
2	1.21	69.17	8.7413	0.4590	0.7052	2.4639
4	1.18	**70.38**	**8.8439**	**0.4689**	0.7186	**2.5349**
8	1.17	69.57	8.7457	0.4636	**0.7196**	2.5196
16	**1.16**	69.61	8.7483	0.4629	0.7177	2.4985
32	**1.16**	69.33	8.7736	0.4642	0.7105	2.5255
64	**1.16**	69.24	8.7174	0.4651	0.7180	2.5070
128	**1.16**	68.73	8.6718	0.4628	0.7127	2.5030
256	**1.16**	68.92	8.6982	0.4629	0.7128	2.5012
512	**1.16**	68.78	8.6857	0.4637	0.7128	2.5025
1024	1.17	69.37	8.7495	0.4659	0.7149	2.5090

Table 18: Validation loss and test set metrics on E2E NLG Challenge achieved by LoRA with different rank r using GPT-2 Medium. Unlike on GPT-3 where $r = 1$ suffices for many tasks, here the performance peaks at $r = 16$ for validation loss and $r = 4$ for BLEU, suggesting the GPT-2 Medium has a similar intrinsic rank for adaptation compared to GPT-3 175B. Note that some of our hyperparameters are tuned on $r = 4$, which matches the parameter count of another baseline, and thus might not be optimal for other choices of r.

Subspace Similarity between different r

$$\phi(A_{r=8}, A_{r=64}, i, j) = \frac{||U_{A_{r=8}}^{i\top} U_{A_{r=64}}^{j}||_F^2}{\min(i, j)} \in [0, 1] \quad (4)$$

where $U_{A_{r=8}}^{i}$ represents the columns of $U_{A_{r=8}}$ corresponding to the top-i singular vectors.

- Right singular matrices of both the matrices.
- We calculate Forbius Norm - to quantify sub space similarity
- Higher value means higher similarity

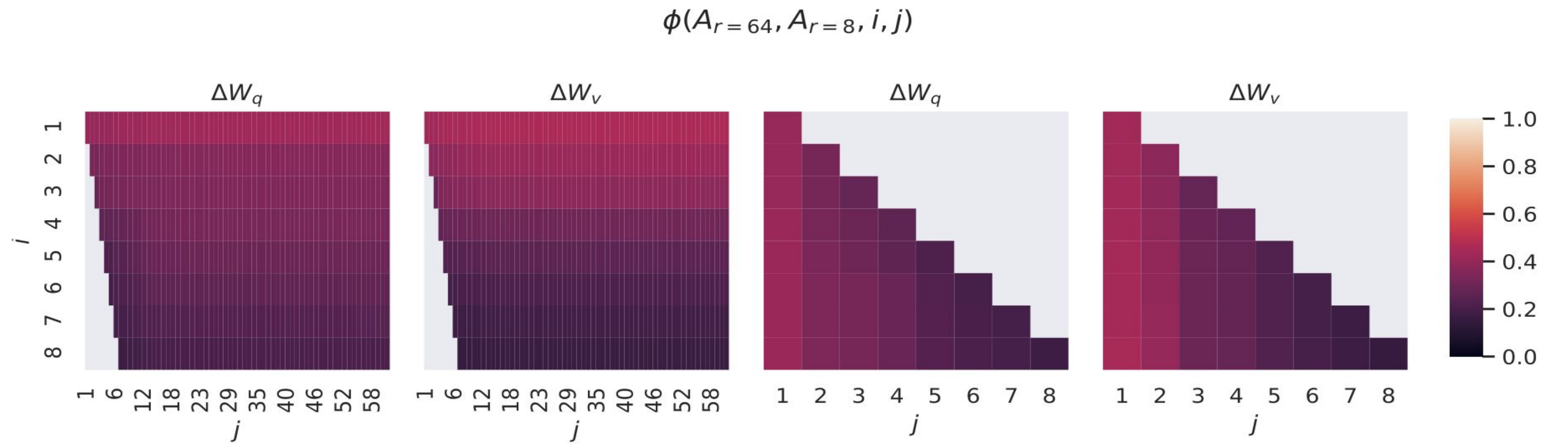


Figure 3: Subspace similarity between column vectors of $A_{r=8}$ and $A_{r=64}$ for both ΔW_q and ΔW_v. The third and the fourth figures zoom in on the lower-left triangle in the first two figures. The top directions in $r = 8$ are included in $r = 64$, and vice versa.

We make an *important observation* from Figure 3.

> Directions corresponding to the top singular vector overlap significantly between $A_{r=8}$ and $A_{r=64}$, while others do not. Specifically, ΔW_v (resp. ΔW_q) of $A_{r=8}$ and ΔW_v (resp. ΔW_q) of $A_{r=64}$ share a subspace of dimension 1 with normalized similarity > 0.5, providing an explanation of why $r = 1$ performs quite well in our downstream tasks for GPT-3.

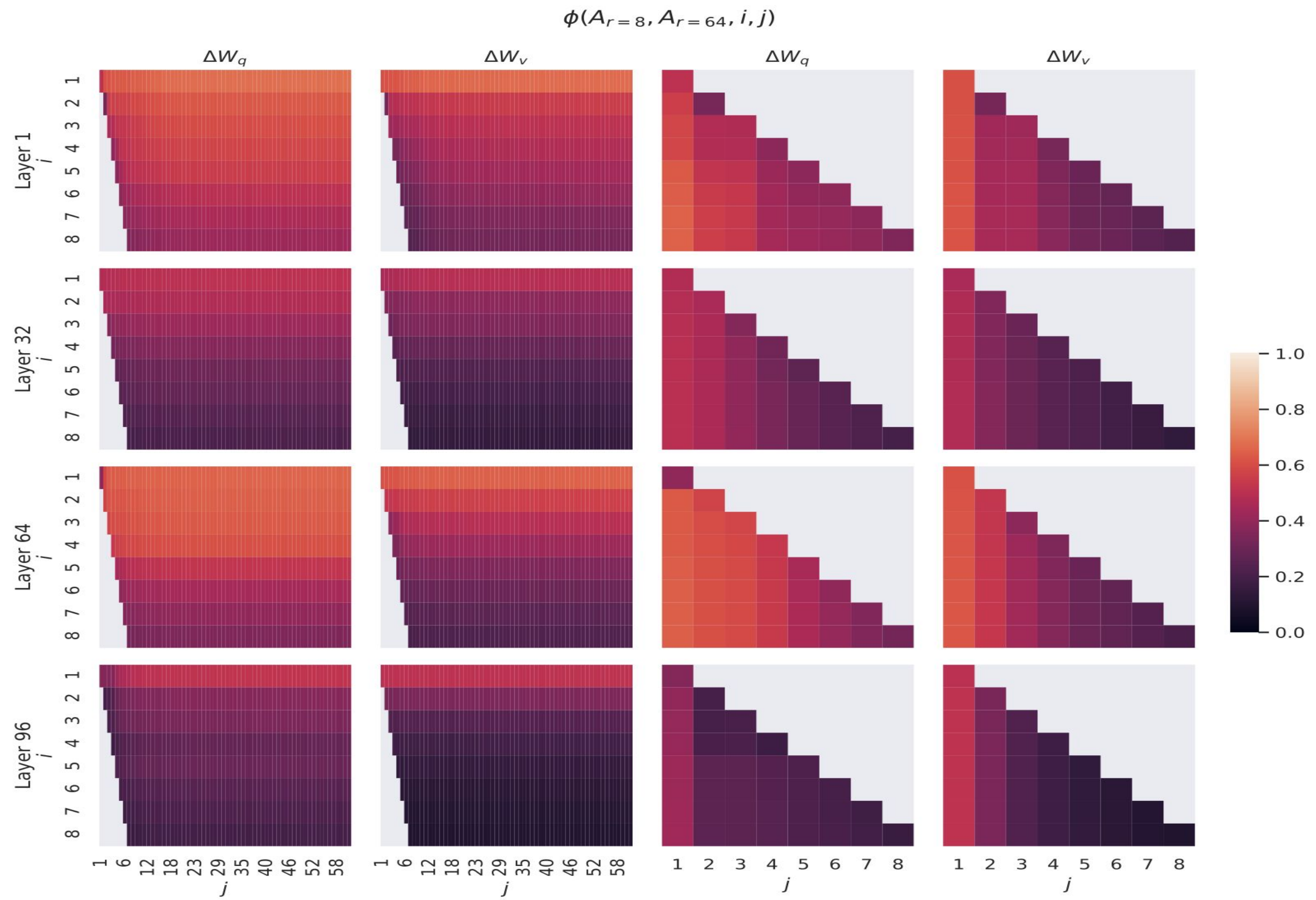


Figure 6: Normalized subspace similarity between the column vectors of $A_{r=8}$ and $A_{r=64}$ for both ΔW_q and ΔW_v from the 1st, 32nd, 64th, and 96th layers in a 96-layer Transformer.

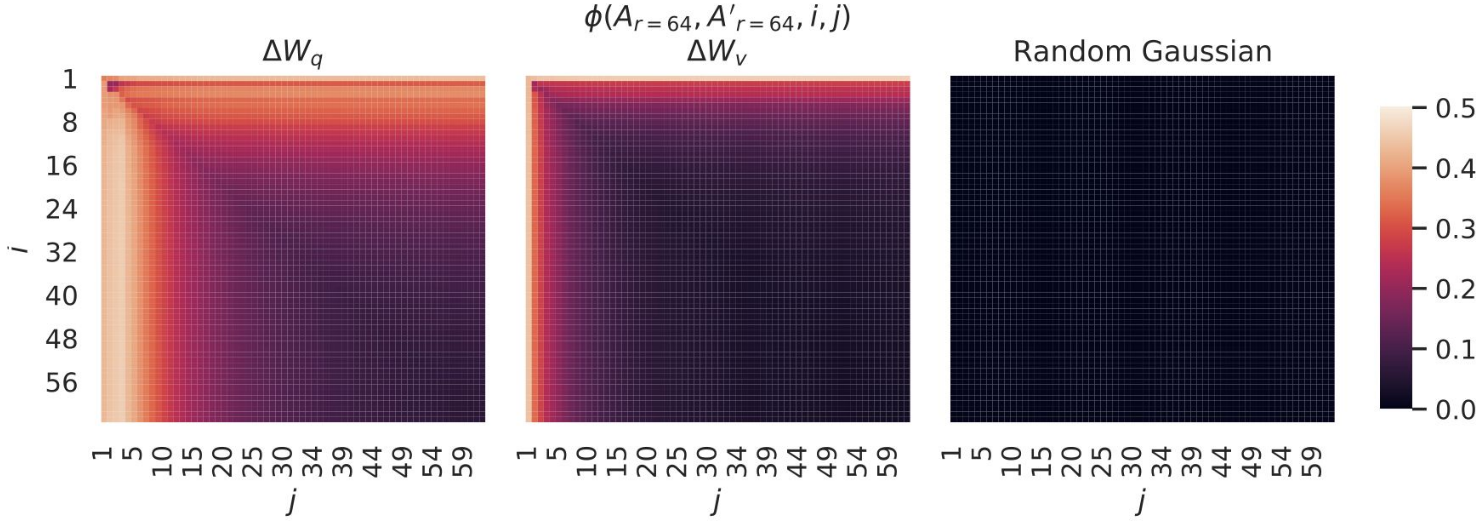


Figure 4: **Left and Middle:** Normalized subspace similarity between the column vectors of $A_{r=64}$ from two random seeds, for both ΔW_q and ΔW_v in the 48-th layer. **Right:** the same heat-map between the column vectors of two random Gaussian matrices. See Section H.1 for other layers.

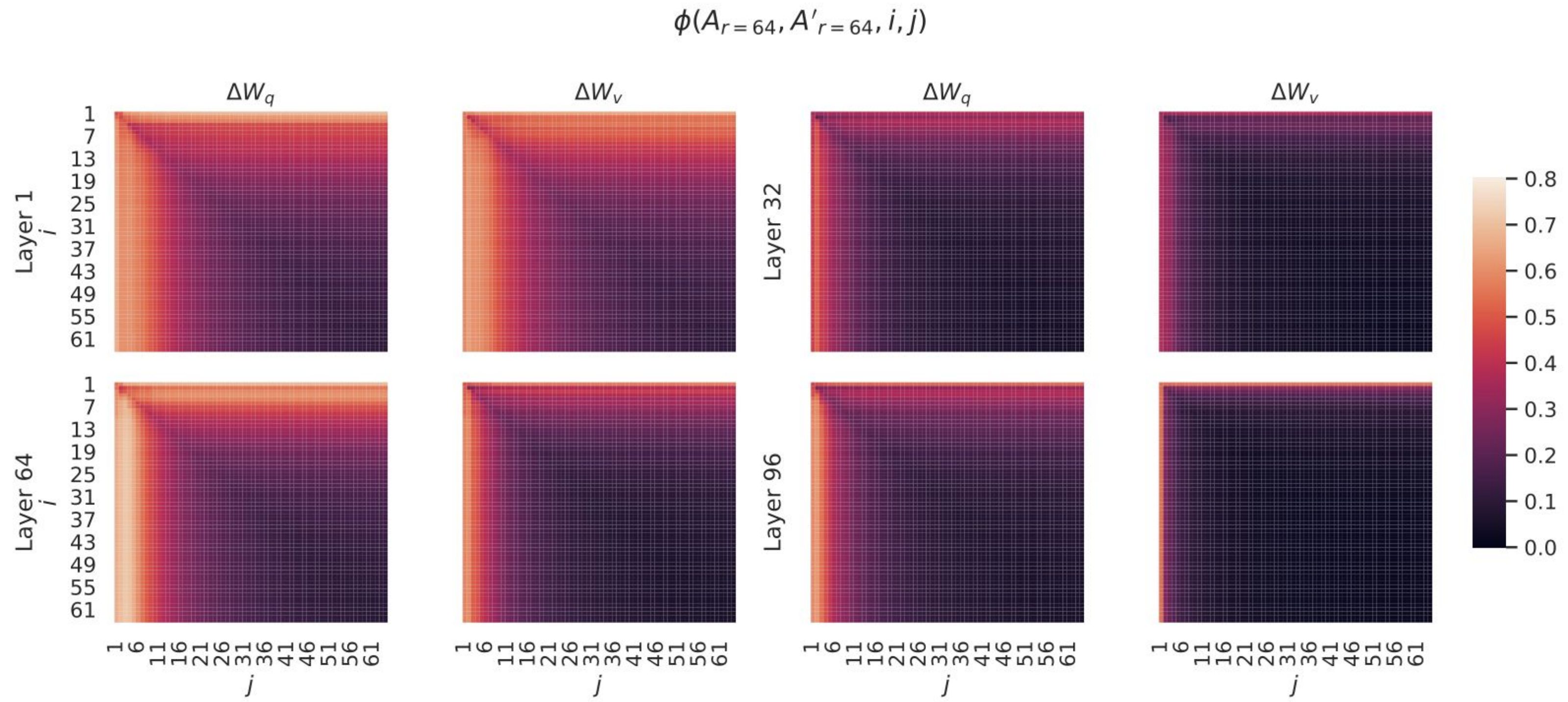


Figure 7: Normalized subspace similarity between the column vectors of $A_{r=64}$ from two randomly seeded runs, for both ΔW_q and ΔW_v from the 1st, 32nd, 64th, and 96th layers in a 96-layer Transformer.

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

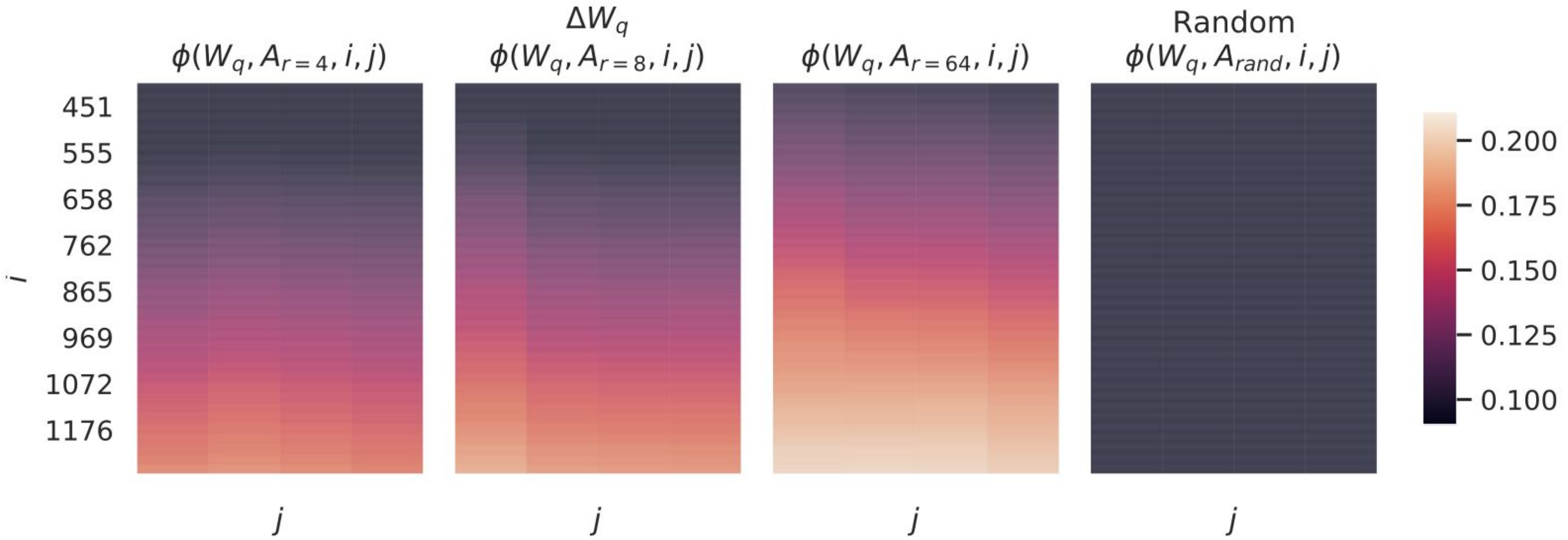


Figure 8: Normalized subspace similarity between the singular directions of W_q and those of ΔW_q with varying r and a random baseline. ΔW_q amplifies directions that are important but not emphasized in W. ΔW with a larger r tends to pick up more directions that are already emphasized in W.

$$\frac{||\Delta W||_F}{||U^\top W V^\top||_F}$$

where:

- $||\Delta W||_F$ is the Frobenius norm of ΔW, which represents the overall magnitude or "energy" of ΔW.
- U and V are the left and right singular matrices from the Singular Value Decomposition (SVD) of ΔW, so we can write $\Delta W = U \Sigma V^\top$.
- $||U^\top W V^\top||_F$ represents the Frobenius norm of the projection of W onto the subspace spanned by the singular vectors of ΔW. This projection aligns W with the directions emphasized by ΔW.

To answer these questions, we project W onto the r-dimensional subspace of ΔW by computing $U^\top W V^\top$, with U/V being the left/right singular-vector matrix of ΔW. Then, we compare the Frobenius norm between $\|U^\top W V^\top\|_F$ and $\|W\|_F$. As a comparison, we also compute $\|U^\top W V^\top\|_F$ by replacing U, V with the top r singular vectors of W or a random matrix.

	$r = 4$			$r = 64$		
	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.

Amplification Factor

- For r=4, the amplification factor can be as large as **20**.
  - This means that **four feature directions in each layer** (from the pre-trained model WW) need to be amplified significantly to achieve the desired performance on the task.
  - Amplification Factor=6.91 / 0.32 = 21.59

- The amplification factor is only around **2** for r=64r=64.
  - Indicates that **most directions** learned in ΔWΔW with r=64r=64 are **not heavily amplified**.
  - Amplification Factor=3.57 / 1.90 = 1.88

Different types of LoRA

- QLoRA
  - Uses **quantization-aware low-rank adaptation**, combining quantization with LoRA for memory efficiency without sacrificing accuracy.
- AdaLoRA
  - Introduces **adaptive rank selection**, adjusting ranks dynamically during training for different layers or tasks.
- LoRA ++
  - An **enhanced version of LoRA** that introduces regularization techniques for better generalization and stability.
- DoRA
  - DoRA identifies and amplifies specific directions in the model's feature space that are most relevant to the target domain.

In a Nutshell

- Training on just 1% of parameters imposes a ceiling on what can be learned, no matter the quality or size of the dataset.

- This restricted capacity means the fine-tuned portion has limited "intelligence" or adaptability, essentially acting like a small model .

- When attempting to add genuinely new knowledge (e.g., a new language), the limited parameter capacity in LoRA leads to poor generalization and increased hallucination. Essentially, the model uses the limited, fine-tuned knowledge, resulting in inaccuracies or shallow understanding.

- LoRA is best for "steering" the model's existing knowledge in new directions rather than teaching it entirely new, complex information.

Thank You