Parameter-efficient fine-tuning (PEFT)

CS 4804: Introduction to AI

Fall 2025

https://tuvllms.github.io/ai-fall-2025/

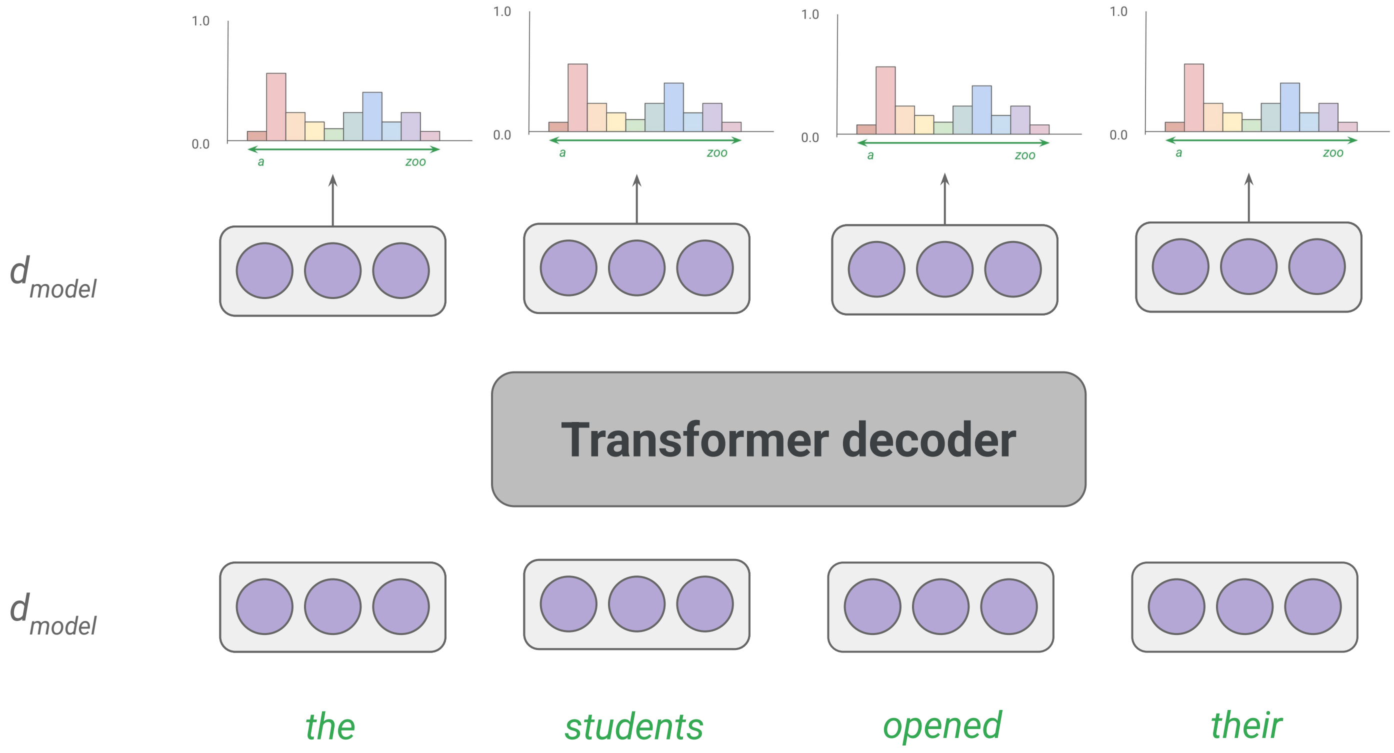
Tu Vu

VIRGINIA TECH.

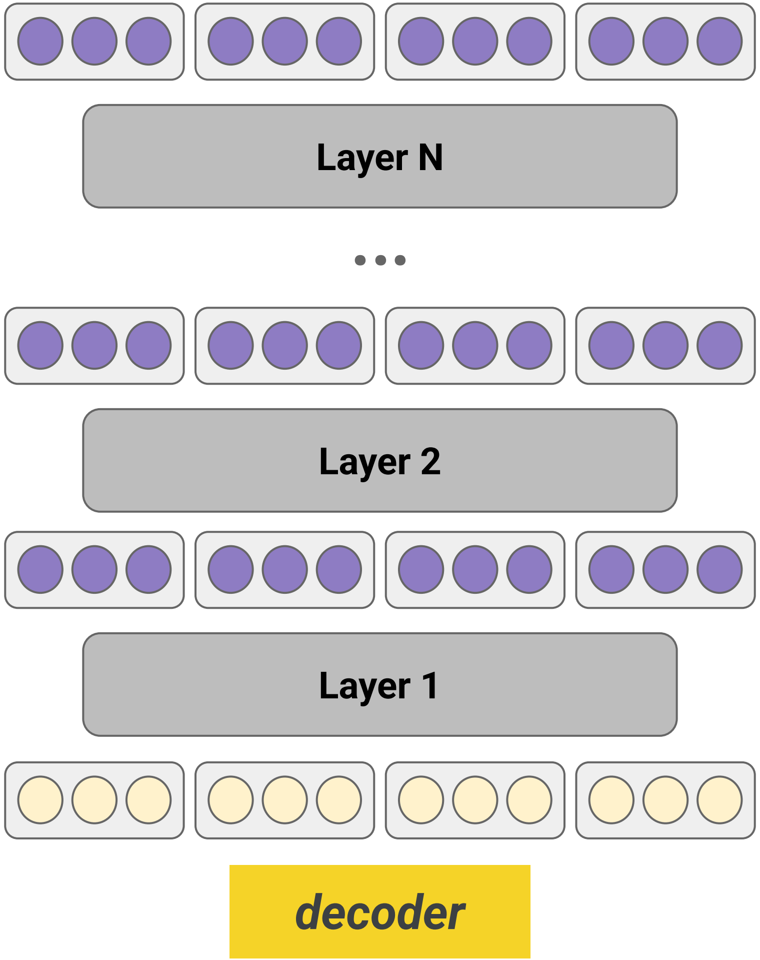
Logistics

- Feedback & grades for final project proposals released
- HW 2 released due 11/18
- Final presentations: 12/4 & 12/9

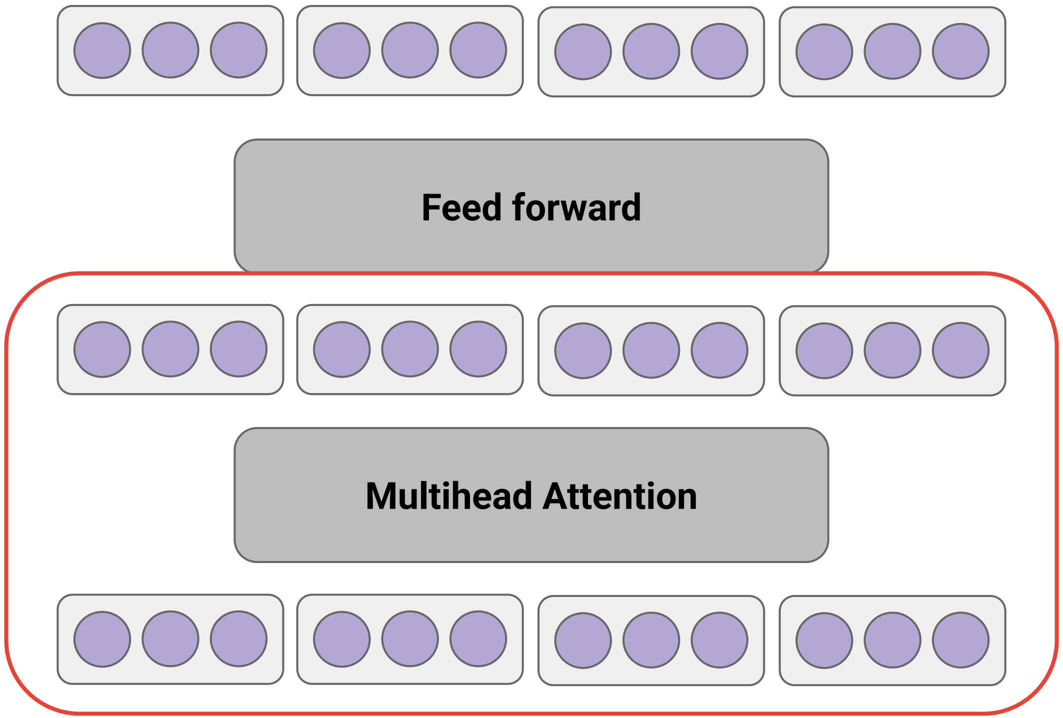
Decoder-only Transformer review



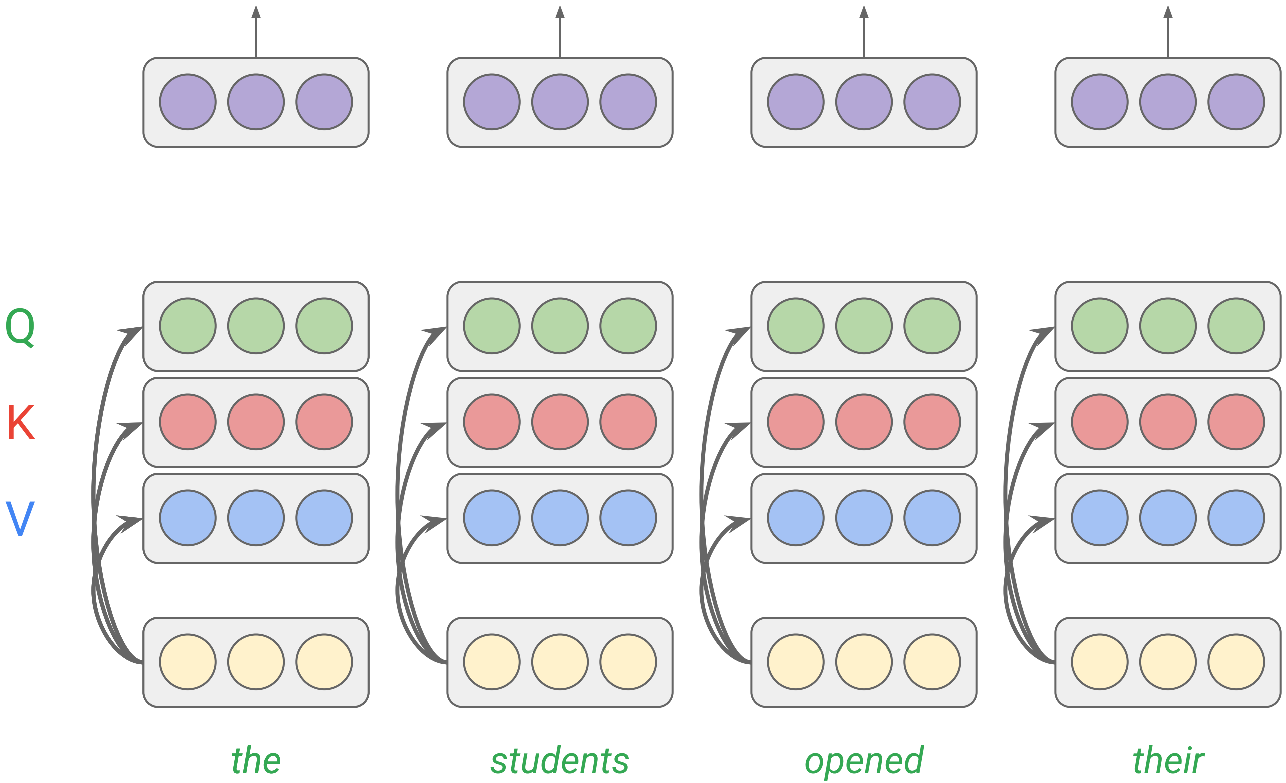
Transformer (N layers)



Transformer decoder



Attention



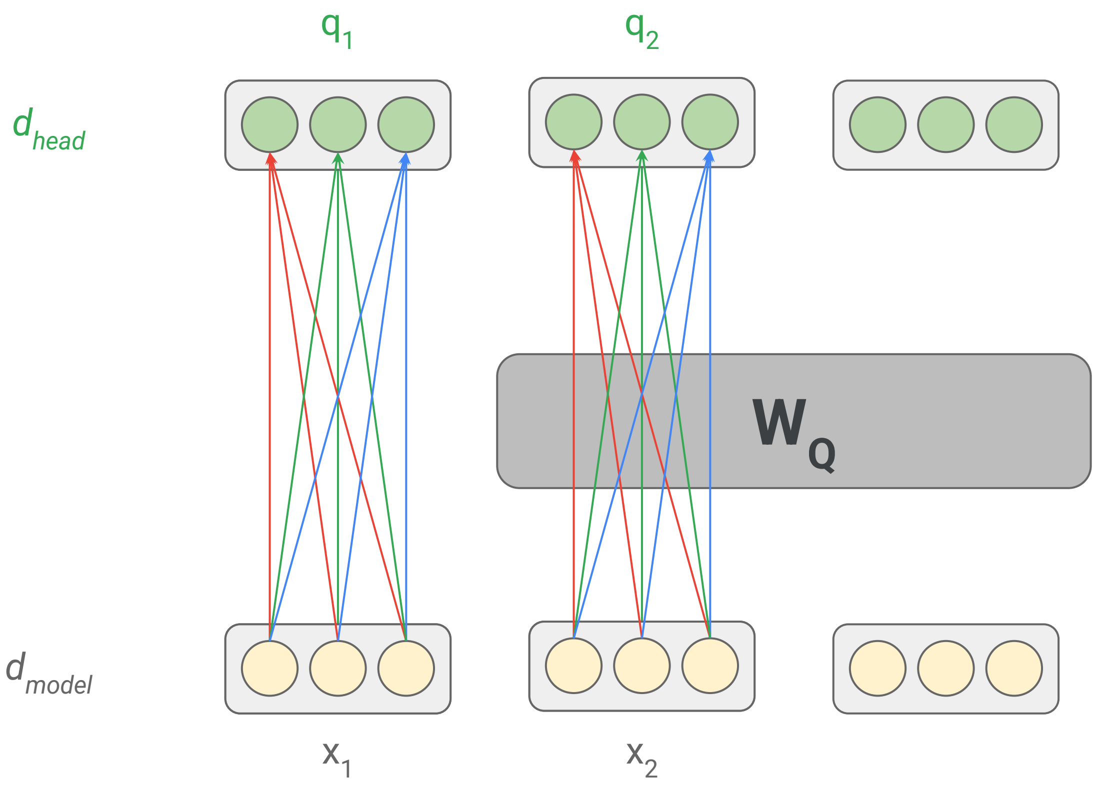
$$Q = X \cdot W_Q$$

$$K = X \cdot W_K$$

$$V = X \cdot W_V$$

linear projections

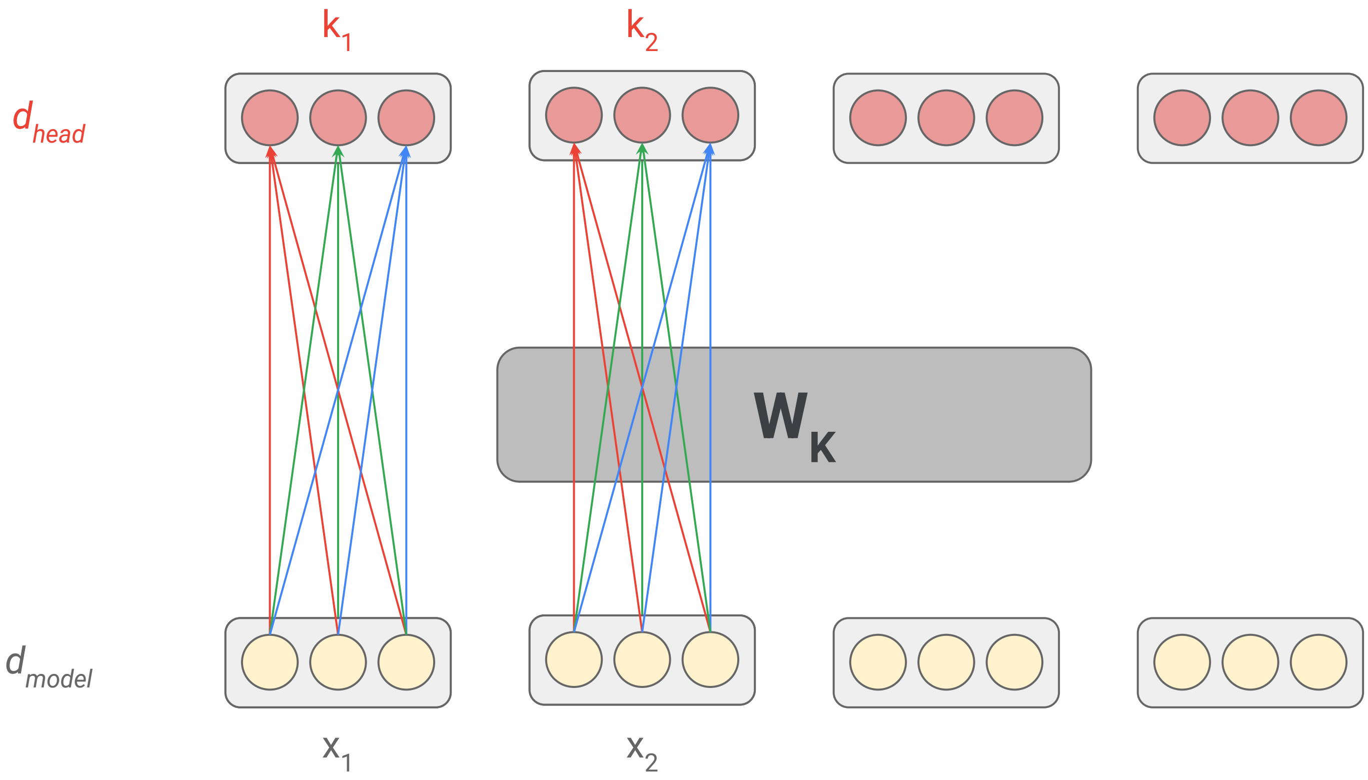
Query vectors



$$Q = X \cdot W_Q$$

linear projections

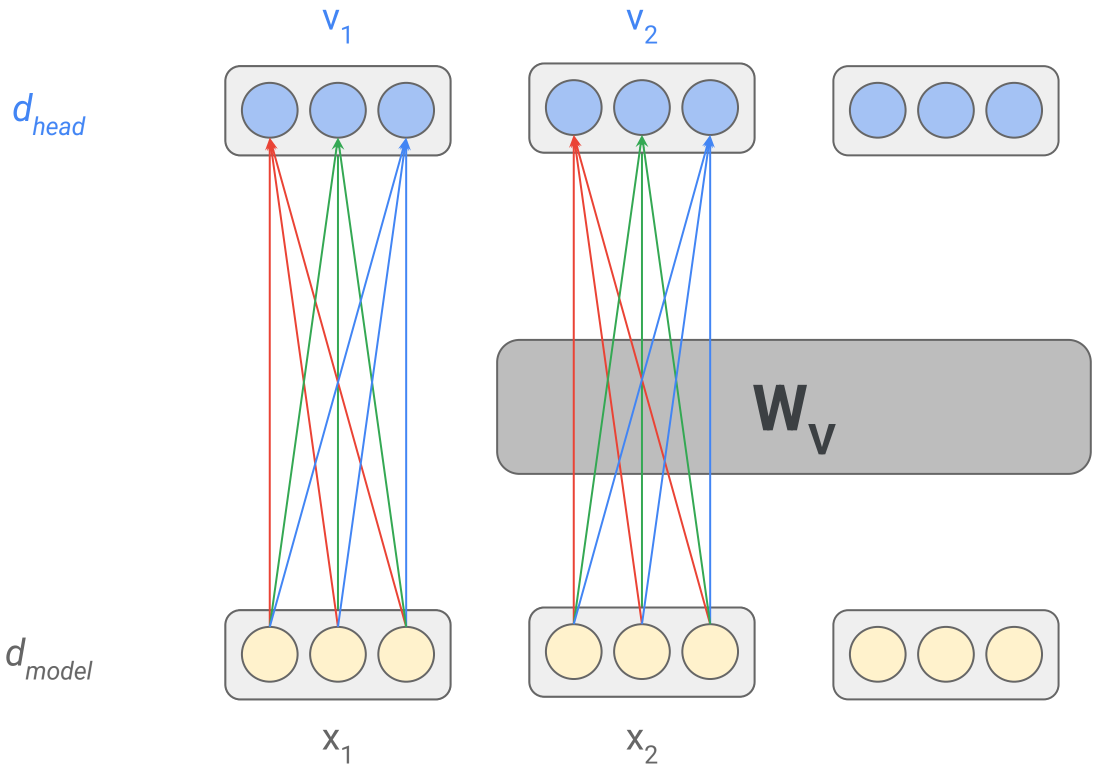
Key vectors



$$K = X \cdot W_K$$

linear projections

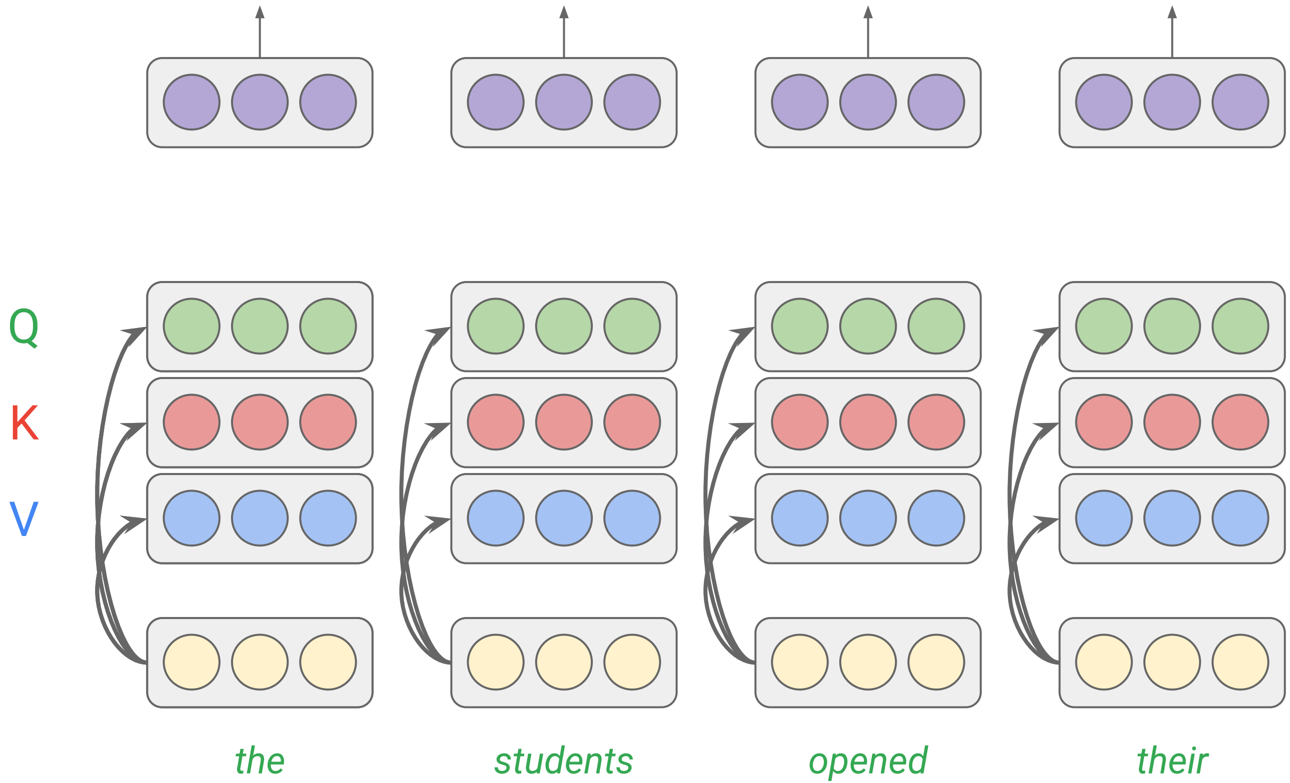
Value vectors



$$V = X \cdot W_V$$

linear projections

Attention (cont'd)



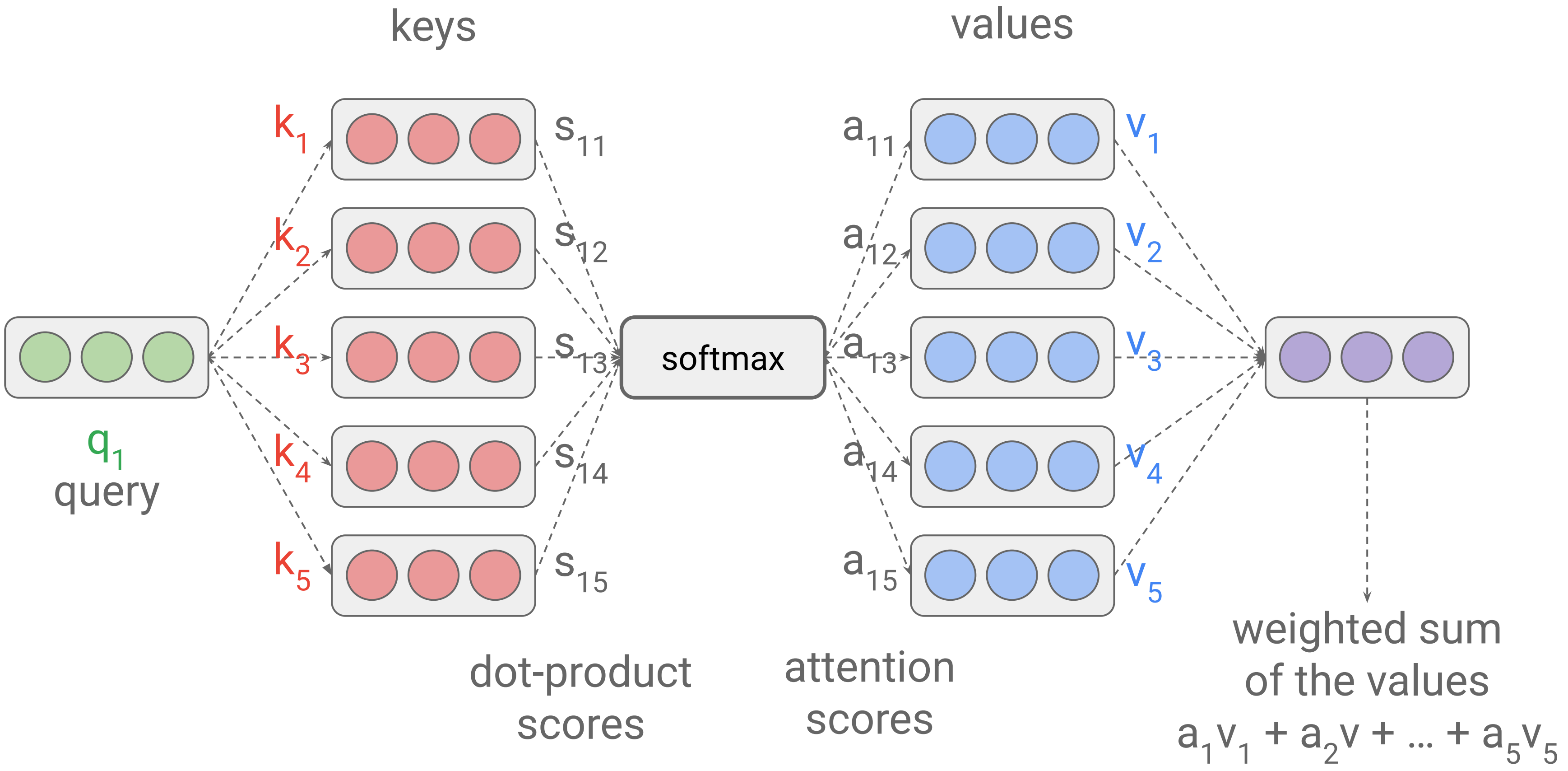
$$Q = X \cdot W_Q$$

$$K = X \cdot W_K$$

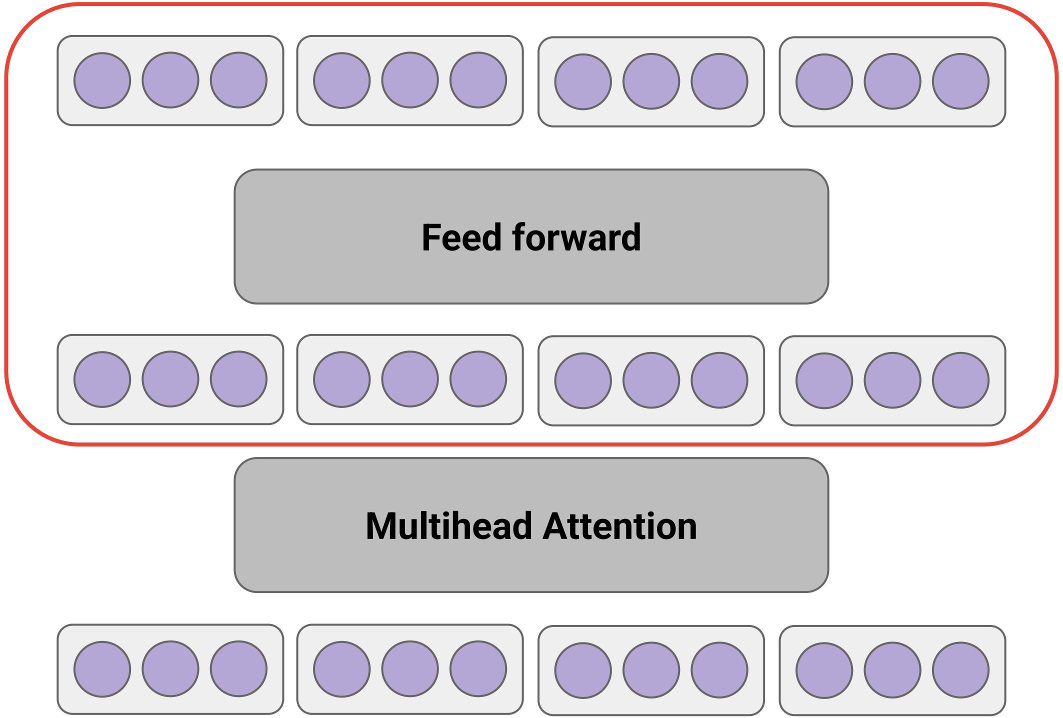
$$V = X \cdot W_V$$

linear projections

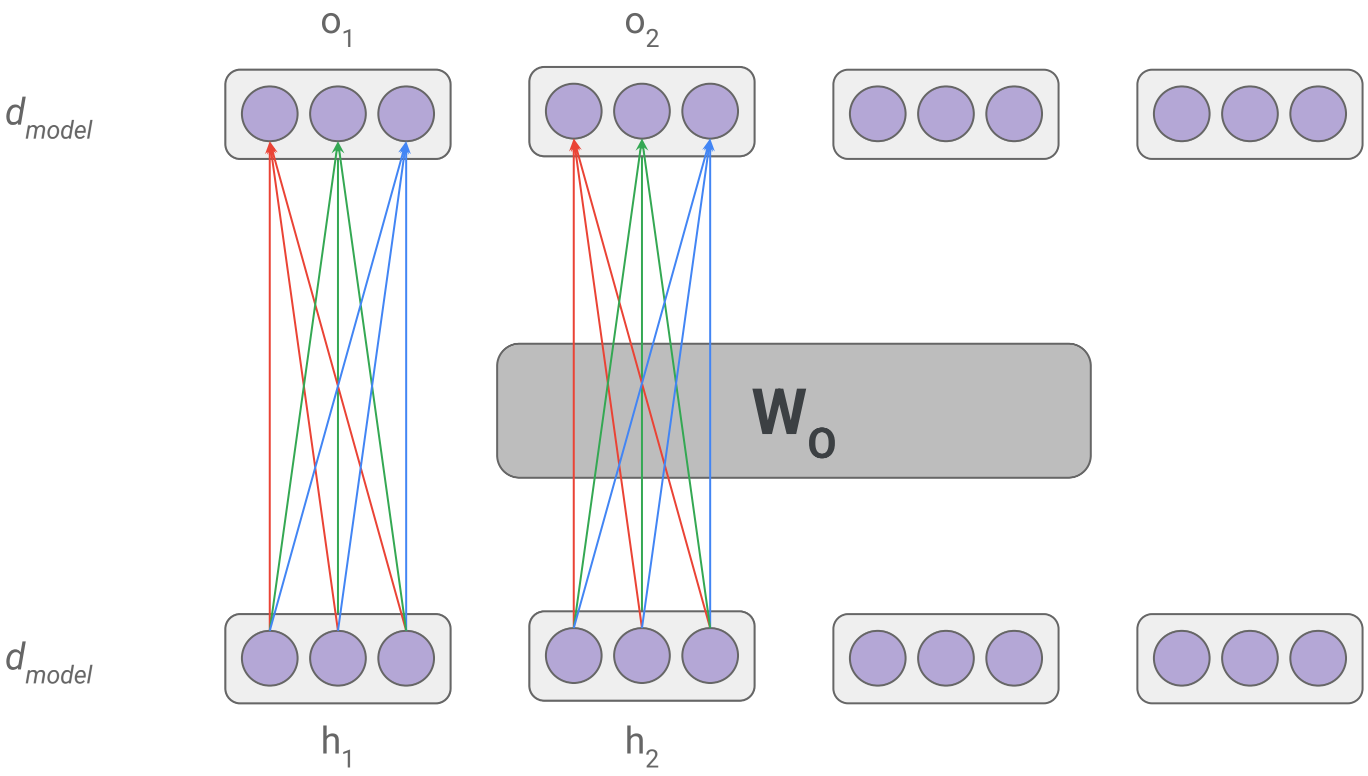
Attention (cont'd)



Transformer decoder



output vectors



$$O = H \cdot W_O$$

linear projections

Model parameters (weights)

- Weight matrices
  - E.g., W_Q , W_K, W_V, W_O
- Bias terms

Bias term

$$h = \sigma(\mathbf{W}x + b)$$

bias term

Updating model parameters

$$w_{t+1} = w_t - \eta \cdot \frac{\partial L}{\partial w_t}$$

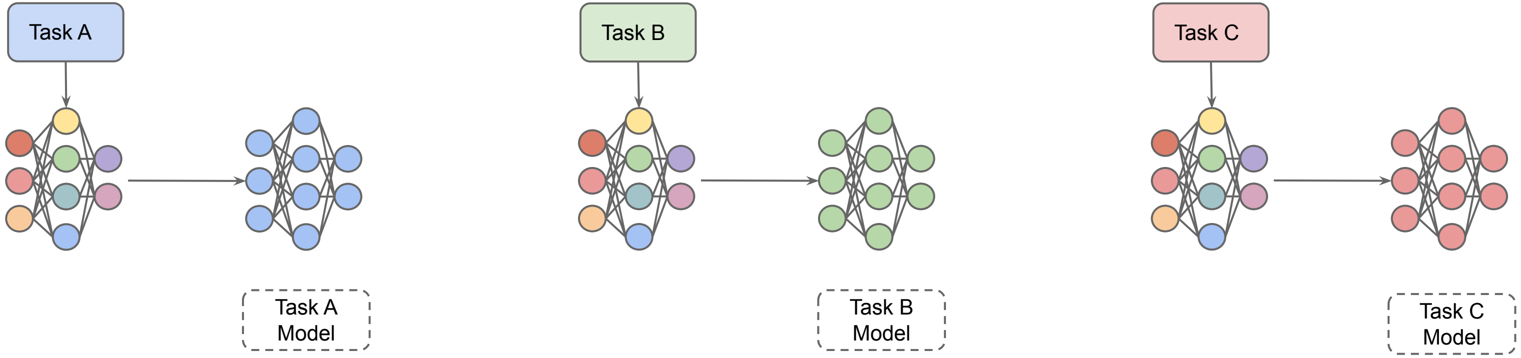
Where:

- w_t is the parameter at the current time step.
- w_{t+1} is the updated parameter after applying the gradient.
- η is the learning rate, which controls the step size.
- $\frac{\partial L}{\partial w_t}$ is the gradient of the loss function L with respect to the parameter w_t, representing how the loss changes as the parameter changes.

Updating model parameters (cont'd)

$$W' = W + \Delta W$$

Full model fine-tuning (Full FT)

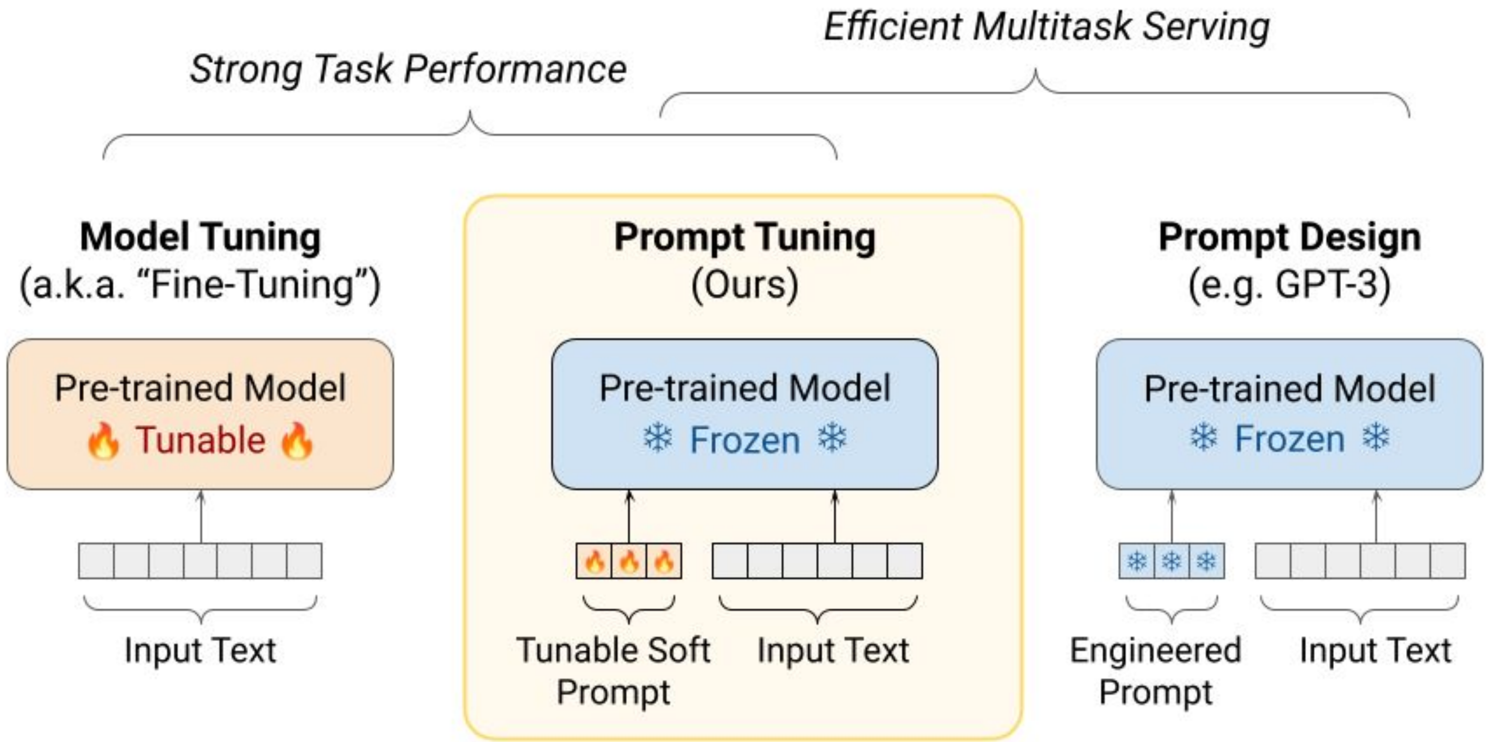


Limitations of full model tuning

The Power of Scale for Parameter-Efficient Prompt Tuning

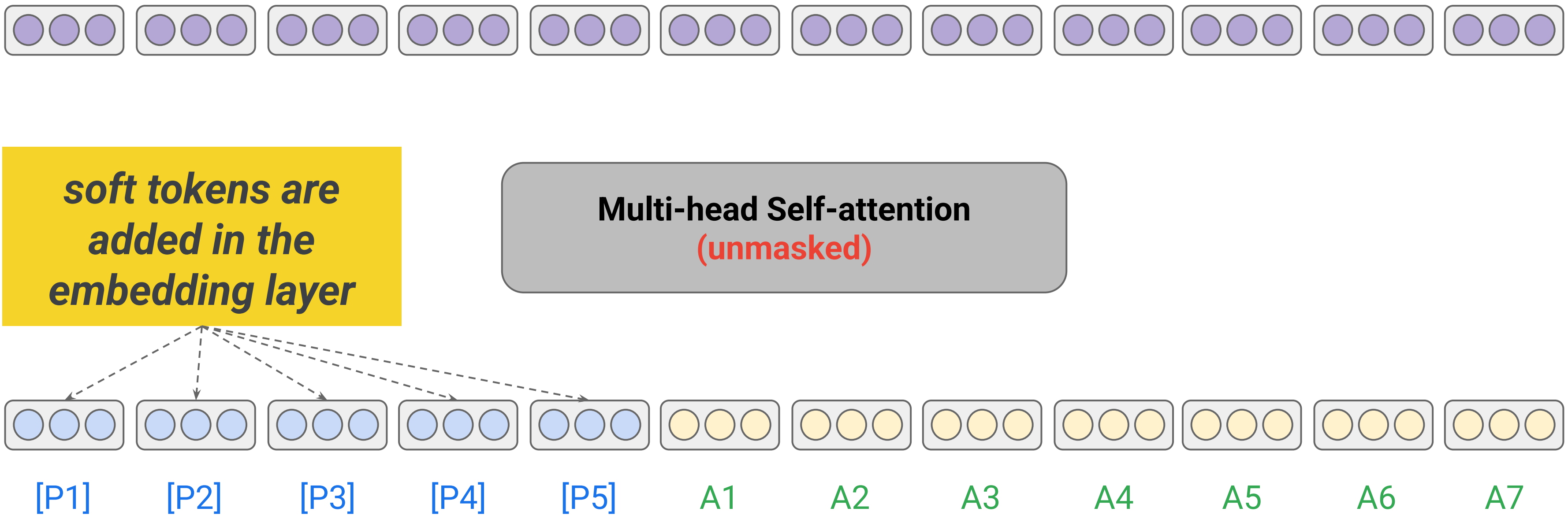
Brian Lester* **Rami Al-Rfou** **Noah Constant**
Google Research
{brianlester,rmyeid,nconstant}@google.com

Soft prompt tuning



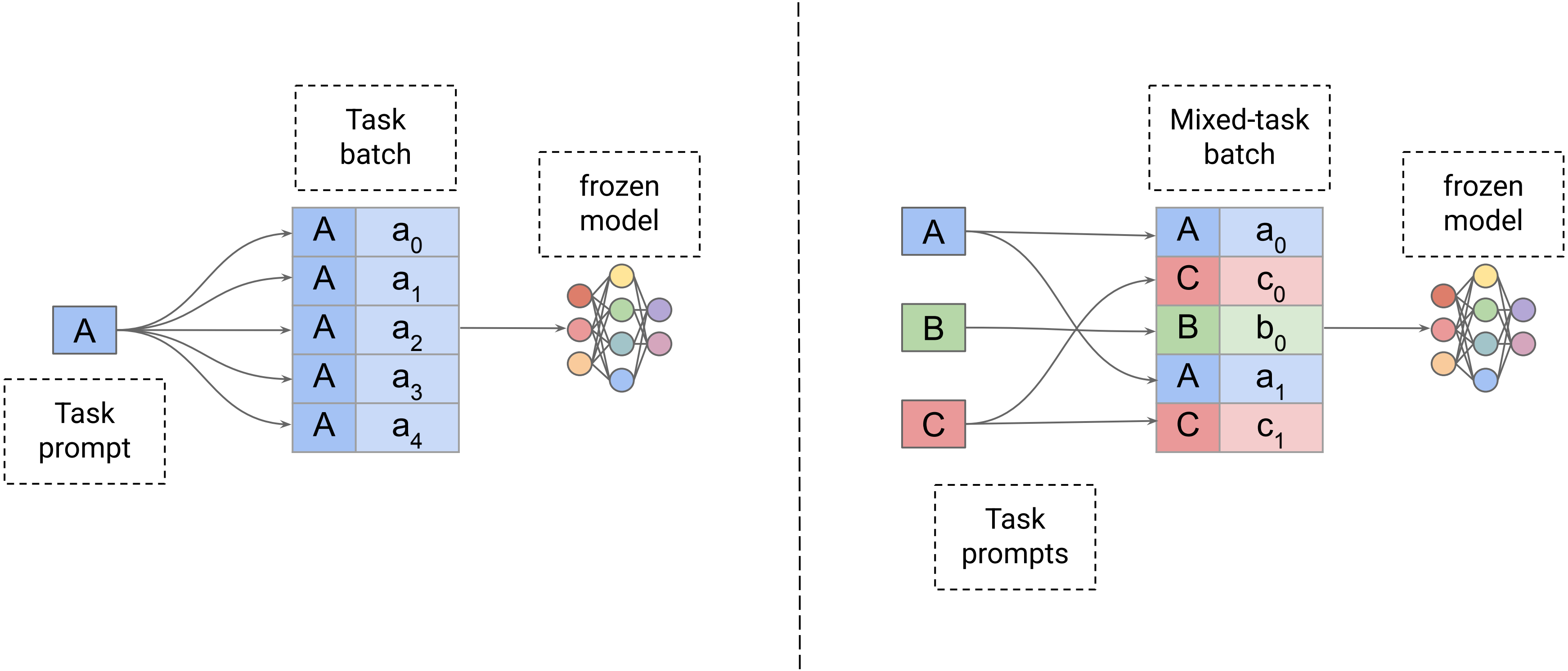
["The Power of Scale for Parameter-Efficient Prompt Tuning" by Lester et al. (2021)]

Soft prompt

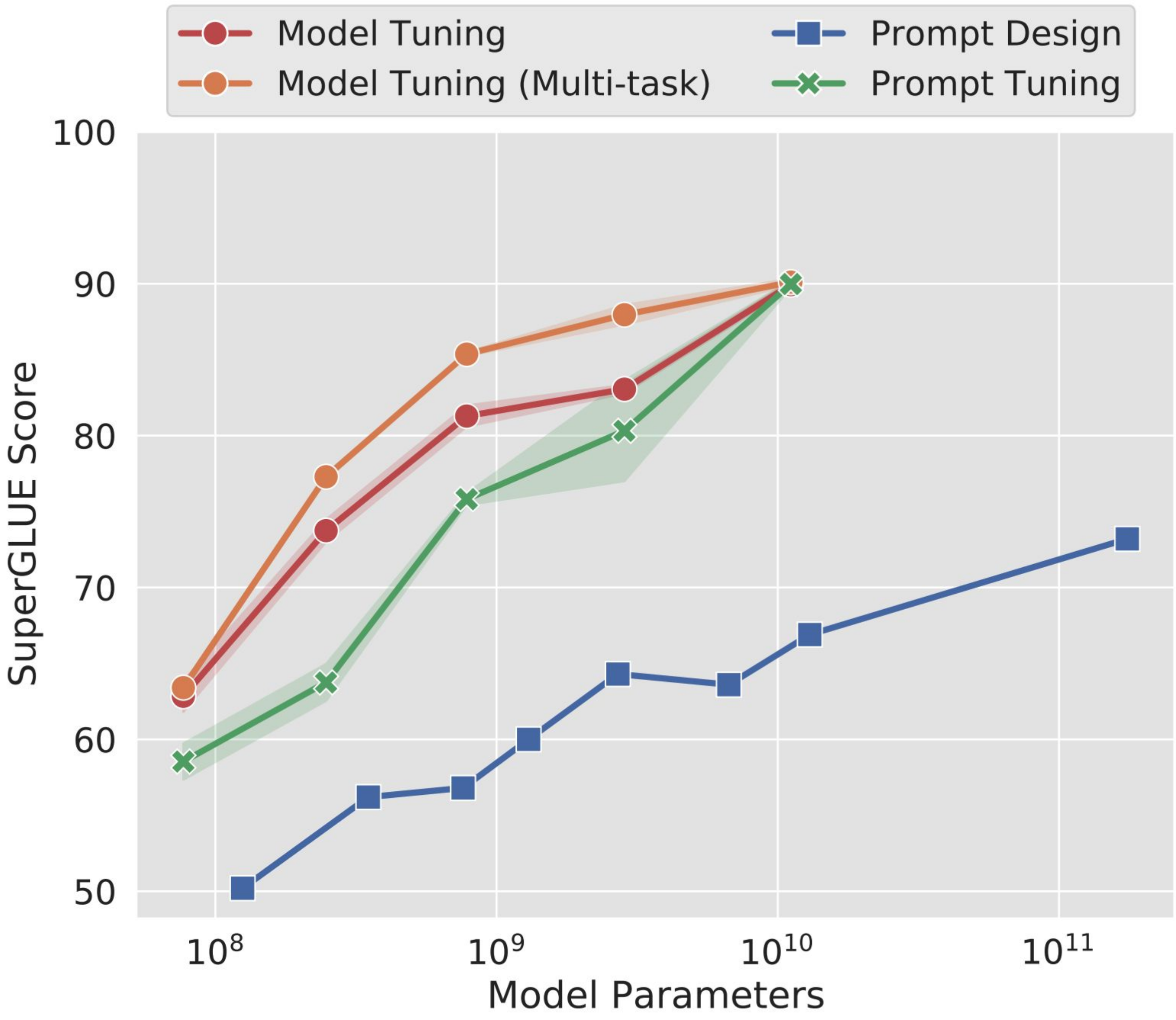

soft tokens are added in the embedding layer
Multi-head Self-attention
(unmasked)
[P1]
[P2]
[P3]
[P4]
[P5]
A1
A2
A3
A4
A5
A6
A7

Advantages of soft prompt tuning

Parameter-efficient tuning & mixed-task inference



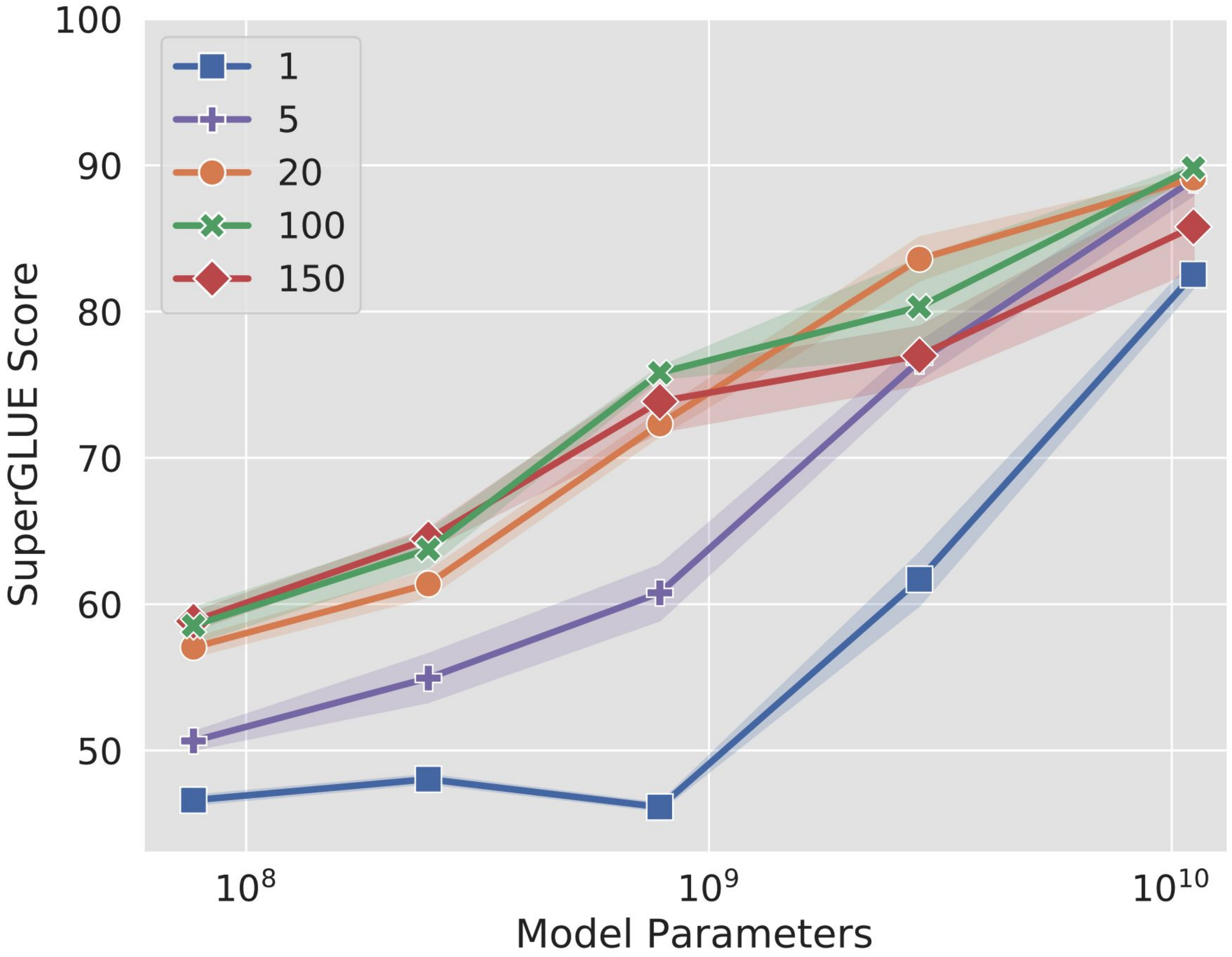
Improvement with Scale



Resilience to domain shift

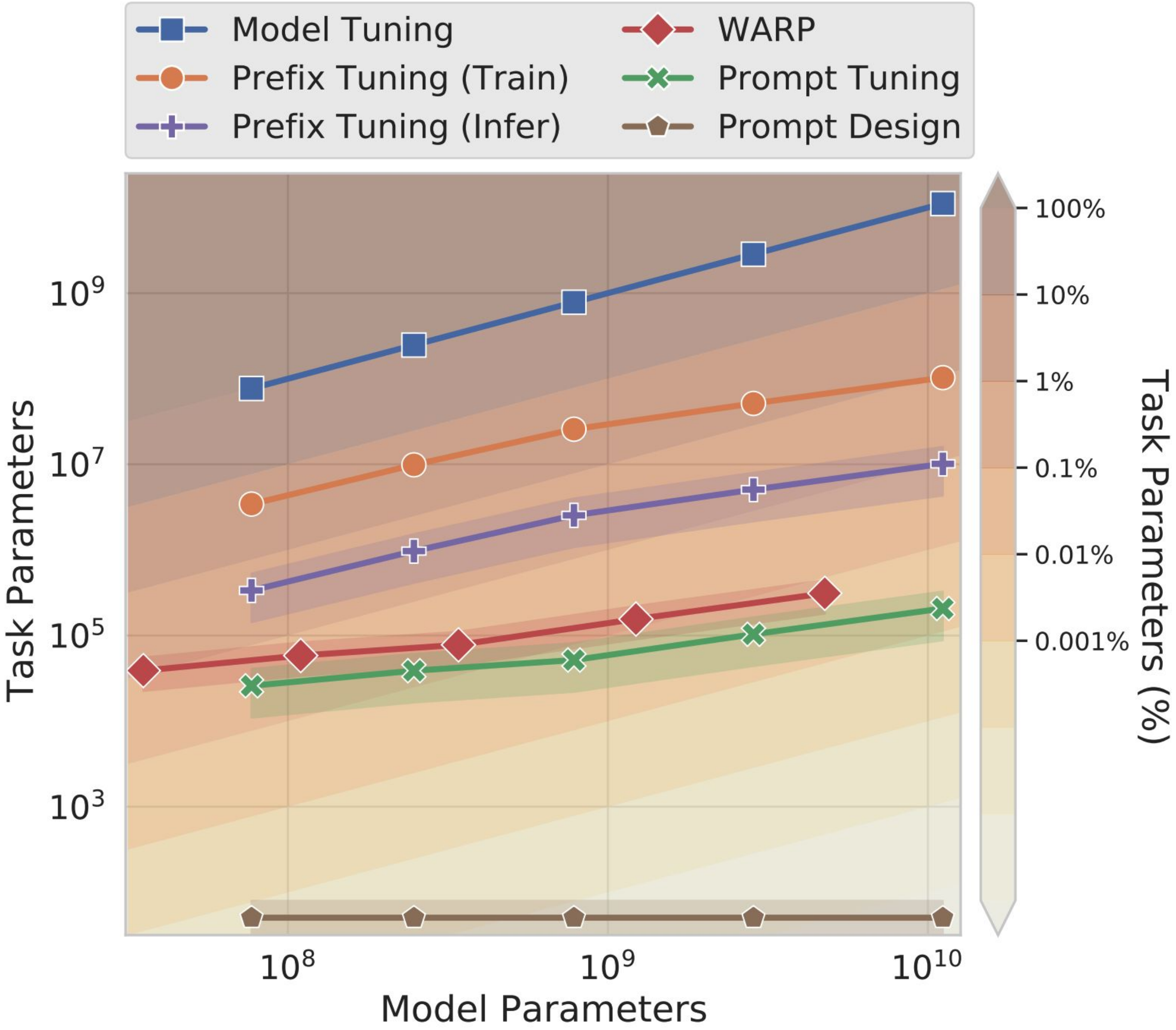
Train	**Eval**	**Tuning**	**Accuracy**	**F1**
QQP	MRPC	Model	73.1 ±0.9	81.2 ±2.1
		Prompt	**76.3** ±0.1	**84.3** ±0.3
MRPC	QQP	Model	74.9 ±1.3	**70.9** ±1.2
		Prompt	**75.4** ±0.8	69.7 ±0.3

Effect of prompt length



Effect of prompt initialization

Parameter usage



Interpretability

- the learned prompts taken as sequences show little interpretability

Limitations of soft prompt tuning

SPoT: Better Frozen Model Adaptation through Soft Prompt Transfer

Tu Vu[1,2★] **Brian Lester**[1] **Noah Constant**[1] **Rami Al-Rfou**[1] **Daniel Cer**[1]

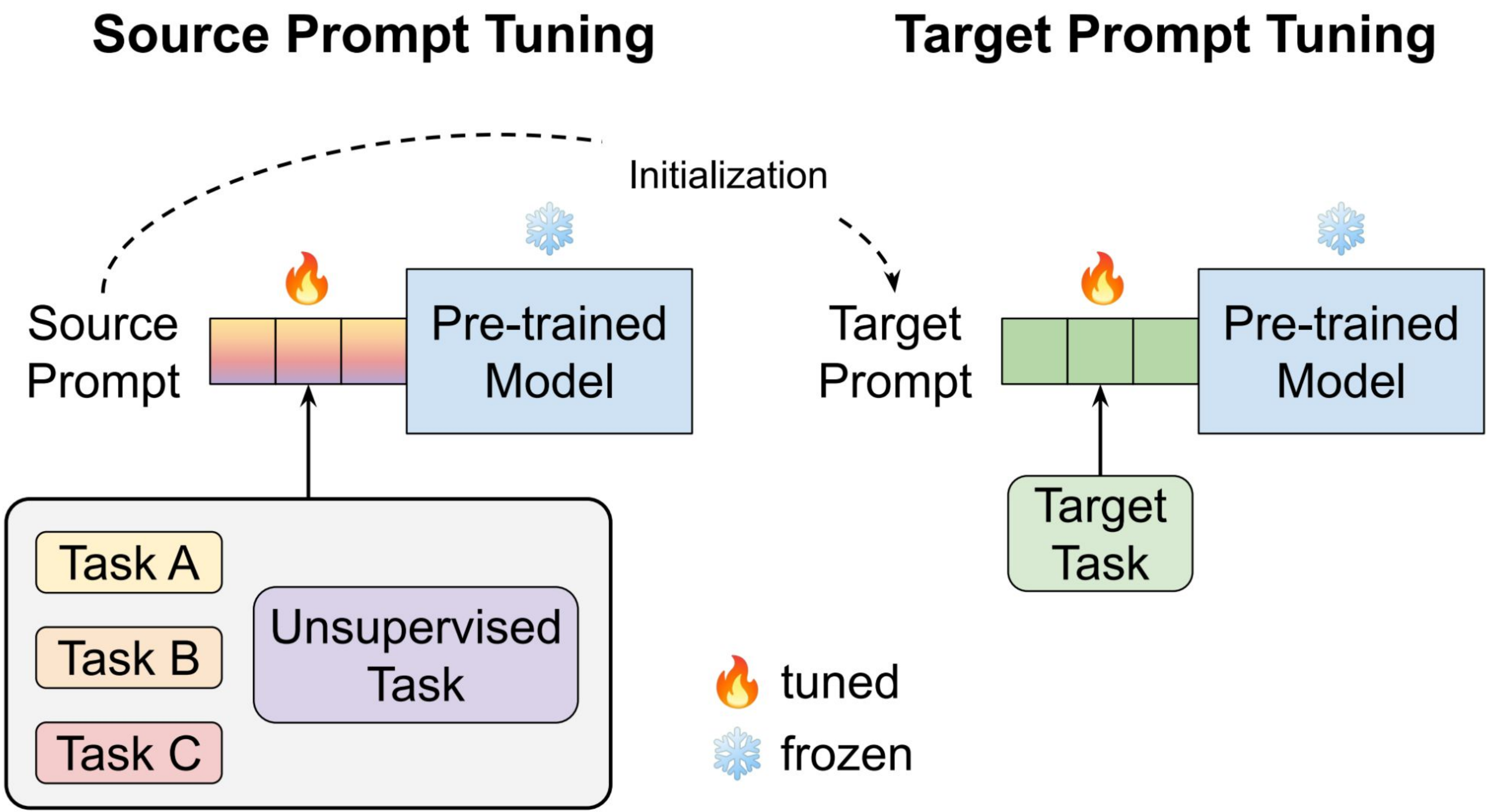
Google Research[1]

University of Massachusetts Amherst[2]

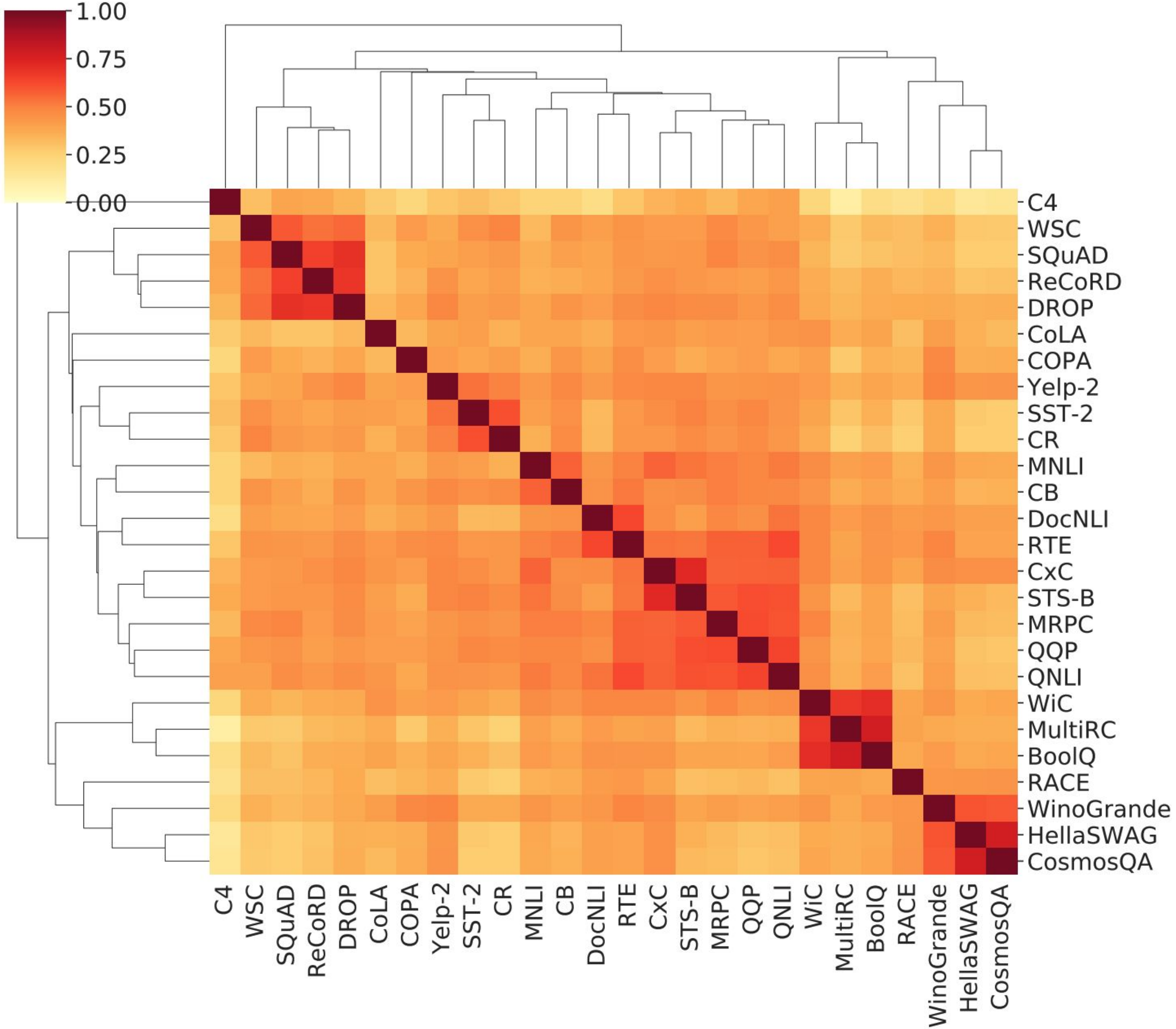
{ttvu,brianlester,nconstant,rmyeid,cer}@google.com

tuvu@cs.umass.edu

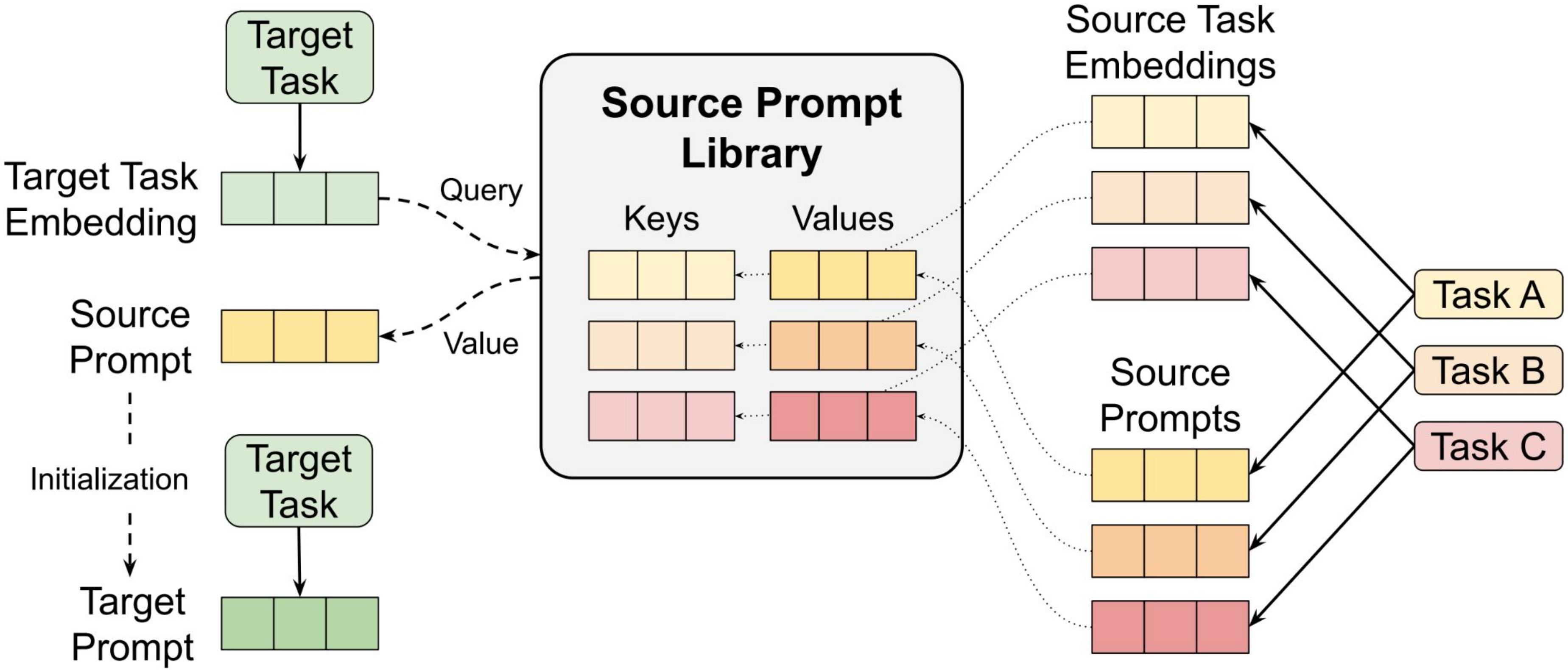
Generic SPoT



Prompt-based task embeddings capture task relationships



Targeted SPoT



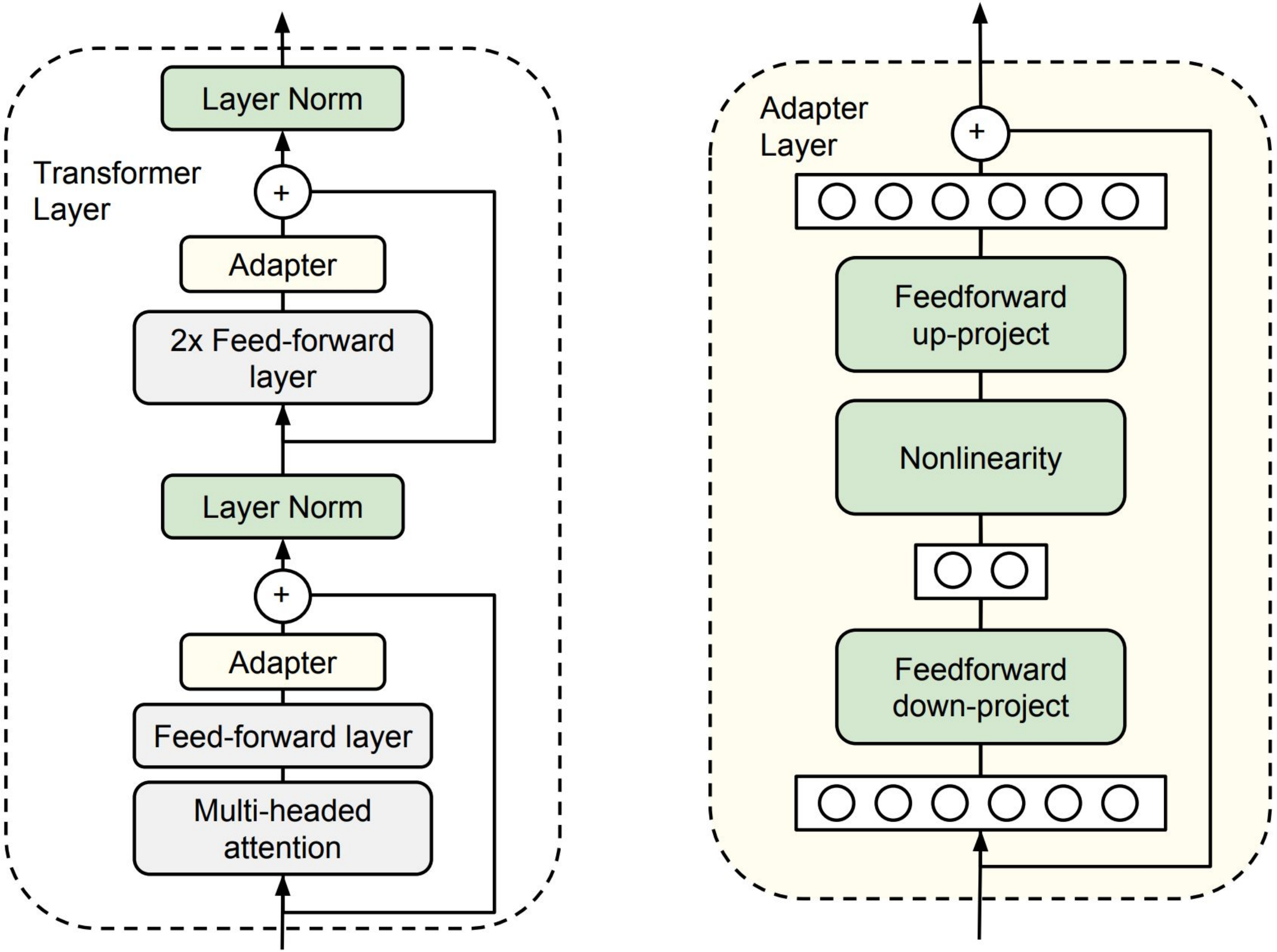
BitFit: Simple Parameter-efficient Fine-tuning for Transformer-based Masked Language-models

Elad Ben-Zaken[1] **Shauli Ravfogel**[1,2] **Yoav Goldberg**[1,2]
[1]Computer Science Department, Bar Ilan University
[2]Allen Institute for Artificial Intelligence
{benzakenelad, shauli.ravfogel, yoav.goldberg}@gmail.com

Parameter-Efficient Transfer Learning for NLP

Neil Houlsby [1] Andrei Giurgiu [1 *] Stanisław Jastrzębski [2 *] Bruna Morrone [1] Quentin de Laroussilhe [1]
Andrea Gesmundo [1] Mona Attariyan [1] Sylvain Gelly [1]

Adapters



Transformer
Layer
Layer Norm
+
Adapter
2x Feed-forward
layer
Layer Norm
+
Adapter
Feed-forward layer
Multi-headed
attention
Adapter
Layer
+
Feedforward
up-project
Nonlinearity
Feedforward
down-project

Prefix-Tuning: Optimizing Continuous Prompts for Generation

Xiang Lisa Li
Stanford University
xlisali@stanford.edu

Percy Liang
Stanford University
pliang@cs.stanford.edu

LoRA

Algebra review

- The rank of a matrix is the number of linearly independent rows or columns (whichever is smaller)

- A ***full-rank*** matrix refers to a matrix that does not have any constraints on its rank. In other words, it has the maximum possible rank, meaning all of its rows and columns are linearly independent.

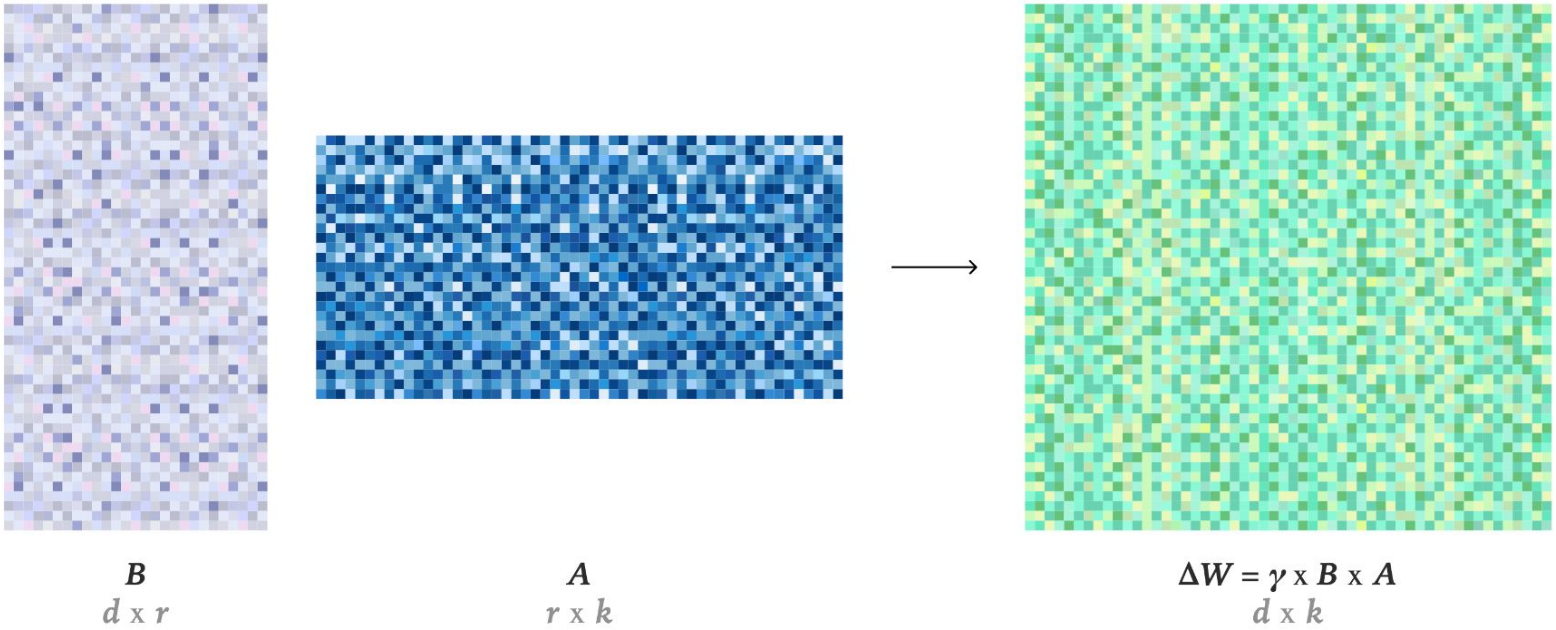
LoRA: Low-Rank Adaptation of Large Language Models

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

Weight changes during model adaptation have a low "intrinsic rank"

- The learned over-parametrized models in fact reside on a low intrinsic dimension
  - intrinsic dimension: the minimal number of variables needed to describe the essential variations in the data
- Many real-world high-dimensional datasets actually lie on or near a lower-dimensional manifold embedded in the high-dimensional space
- If a model or function resides in a low intrinsic dimension, then it may be possible to approximate it well with fewer parameters or a lower-dimensional representation, leading to improved generalization and efficiency

LoRA



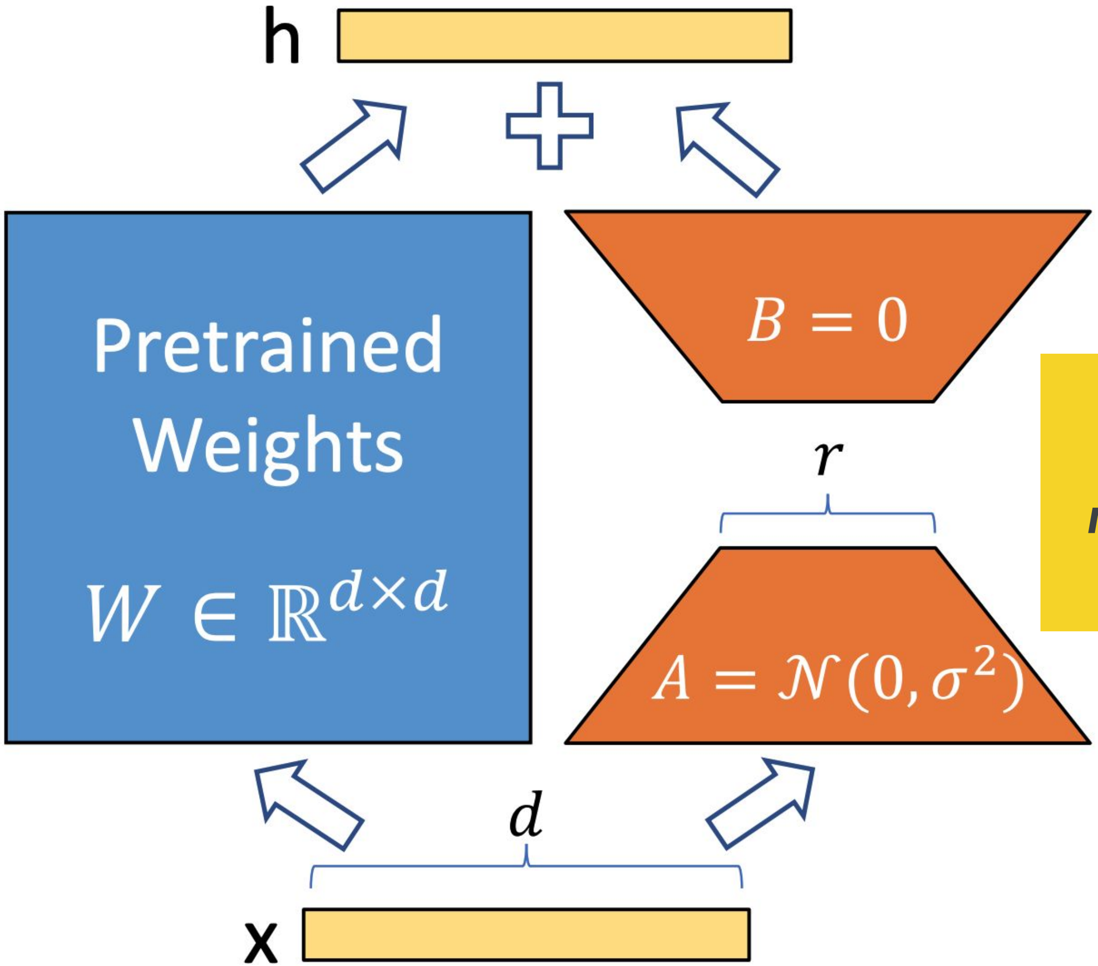
https://thinkingmachines.ai/blog/lora/

For a pre-trained weight matrix $W_0 \in \mathbb{R}^{d\times k}$, we constrain its update by representing the latter with a low-rank decomposition $W_0 + \Delta W = W_0 + BA$, where $B \in \mathbb{R}^{d\times r}$, $A \in \mathbb{R}^{r\times k}$, and the rank $r \ll \min(d, k)$. During training, W_0 is frozen and does not receive gradient updates, while A and B contain trainable parameters. Note both W_0 and $\Delta W = BA$ are multiplied with the same input, and their respective output vectors are summed coordinate-wise.

For $h = W_0 x$, our modified forward pass yields:

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

LoRA



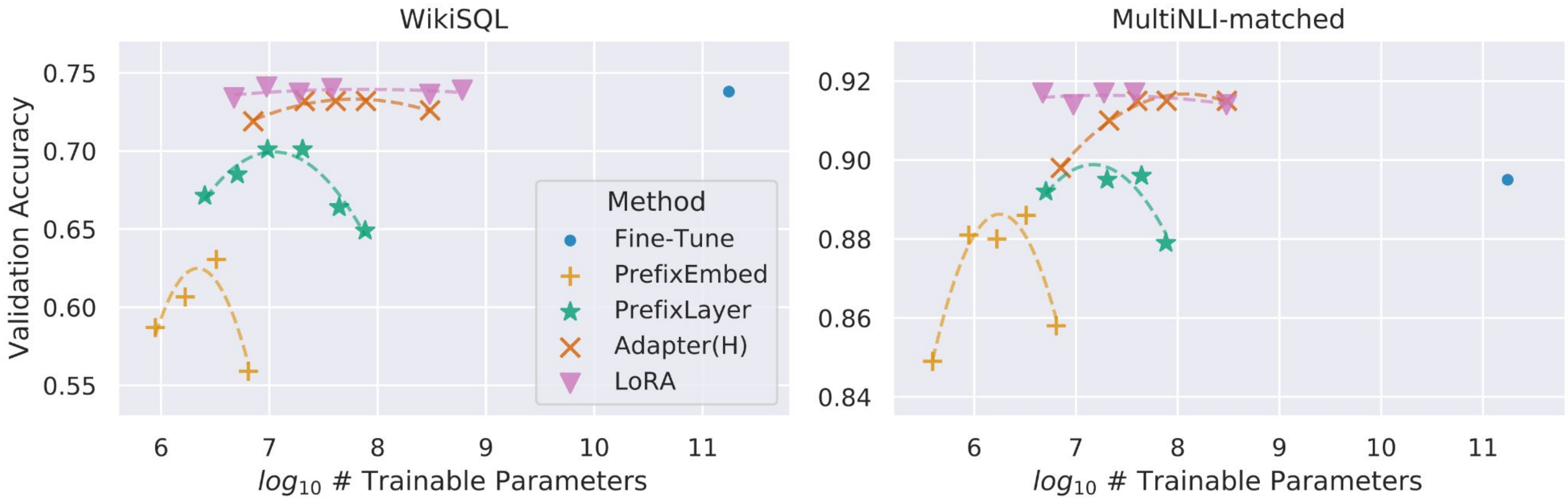
Only rank decomposition matrices A and B are trained

Advantages of LoRA

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**

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

LoRA exhibits better scalability and task performance



Given a limited parameter budget, which weight matrices should we apply LoRA to?

	# of Trainable Parameters = 18M						
Weight Type Rank r	W_q 8	W_k 8	W_v 8	W_o 8	W_q, W_k 4	W_q, W_v 4	W_q, W_k, W_v, W_o 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**

The effect of rank *r* on model performance

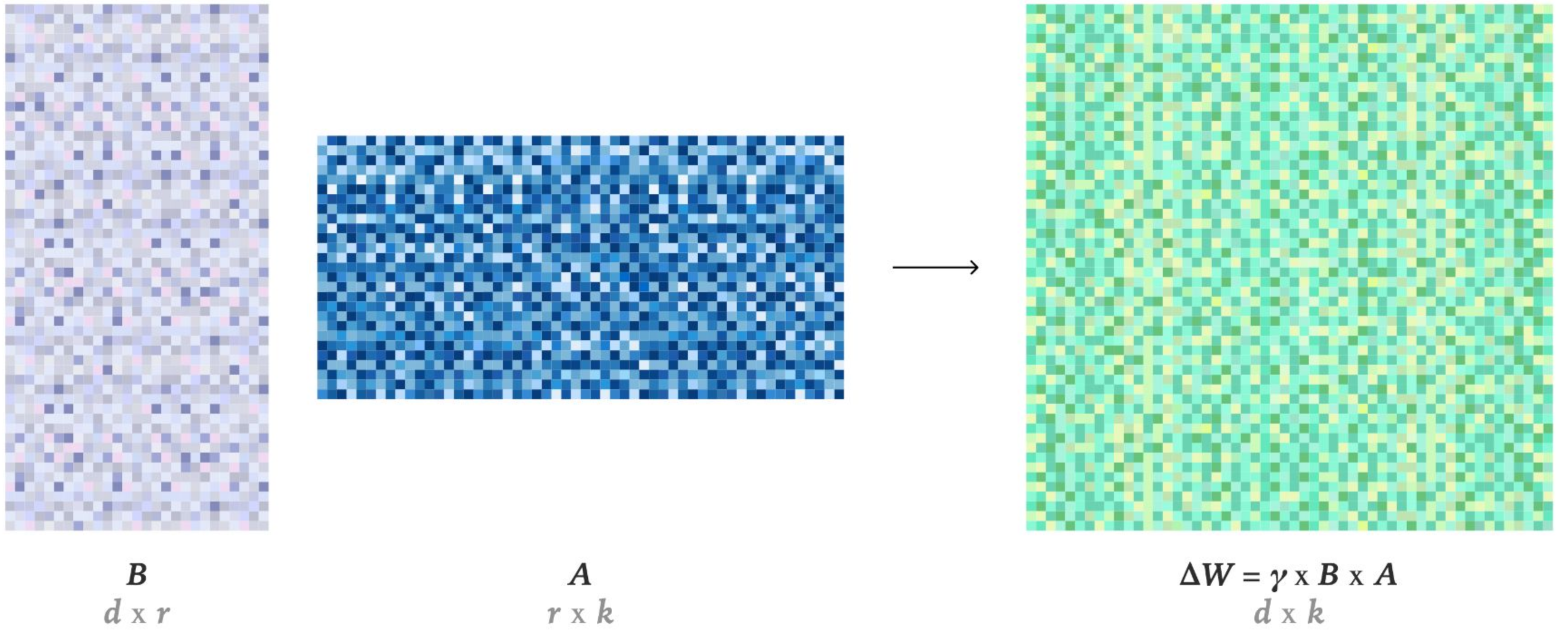
	Weight Type	$r = 1$	$r = 2$	$r = 4$	$r = 8$	$r = 64$
WikiSQL($\pm 0.5\%$)	W_q	68.8	69.6	70.5	70.4	70.0
	W_q, W_v	73.4	73.3	73.7	73.8	73.5
	W_q, W_k, W_v, W_o	74.1	73.7	74.0	74.0	73.9
MultiNLI ($\pm 0.1\%$)	W_q	90.7	90.9	91.1	90.7	90.7
	W_q, W_v	91.3	91.4	91.3	91.6	91.4
	W_q, W_k, W_v, W_o	91.2	91.7	91.7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

practical recommendations

of training examples

- < 20: LoRA is difficult to train
- 50: LoRA w/ careful settings can be better than full model fine-tuning; r=1 or 4
- O(100): e.g., 200-500, LoRA is recommended; r=1 or 4
- O(10K): should compare LoRA vs. full model fine-tuning
- Very large (>100K): LoRA can get decent quality to match full model fine-tuning when r is large, e.g., 128 or 512

LoRA Without Regret



https://thinkingmachines.ai/blog/lora/

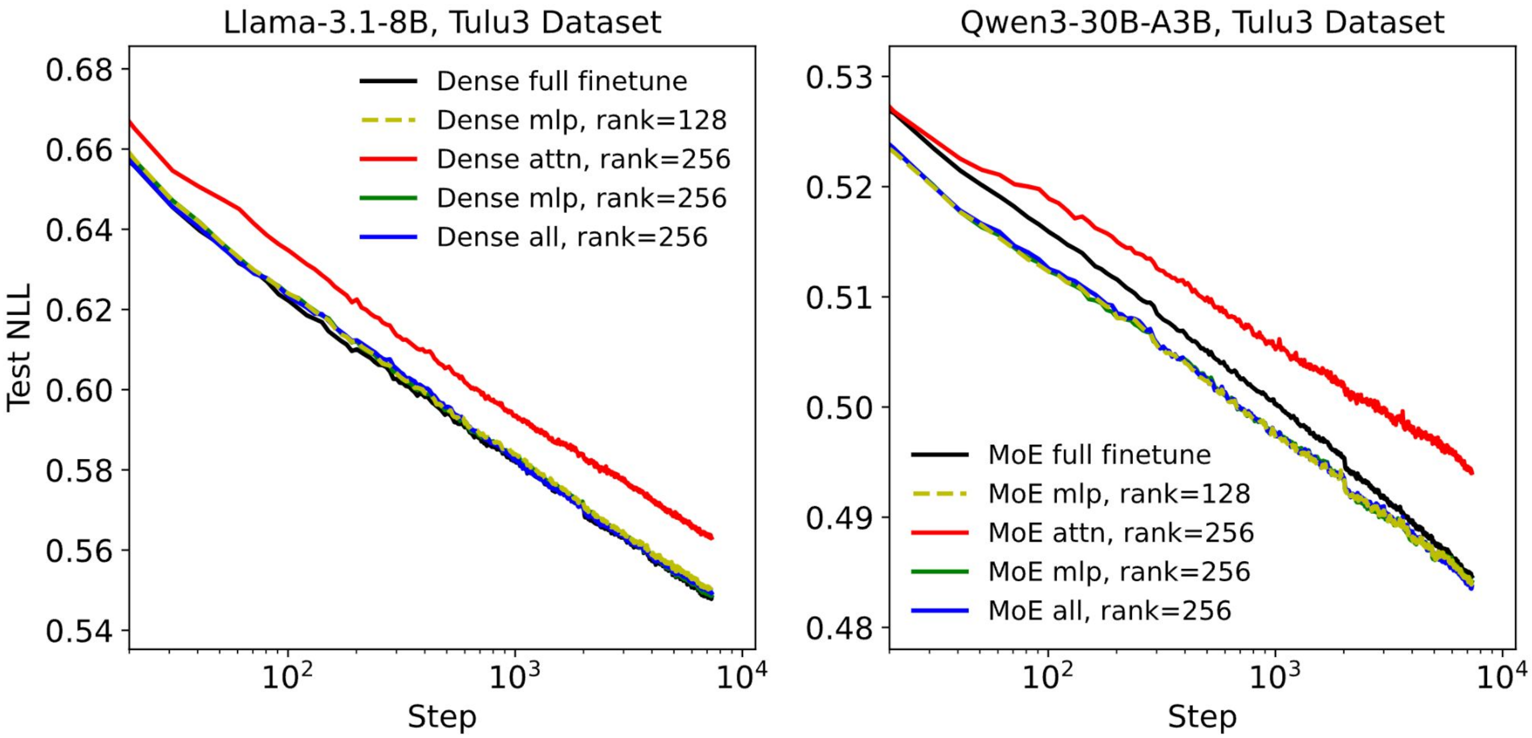
LoRA Without Regret

- LoRA can match full fine-tuning – both in sample efficiency and final performance – so long as two conditions hold:
  - You apply LoRA to all the weight matrices (especially MLP / MoE layers, not just attention).
  - The adapter has enough capacity relative to the amount of information to learn

  If the adapter rank is too small, LoRA becomes bottlenecked and cannot capture all the necessary updates (GPT-style models have a capacity of approximately 3.6 bits per parameter)

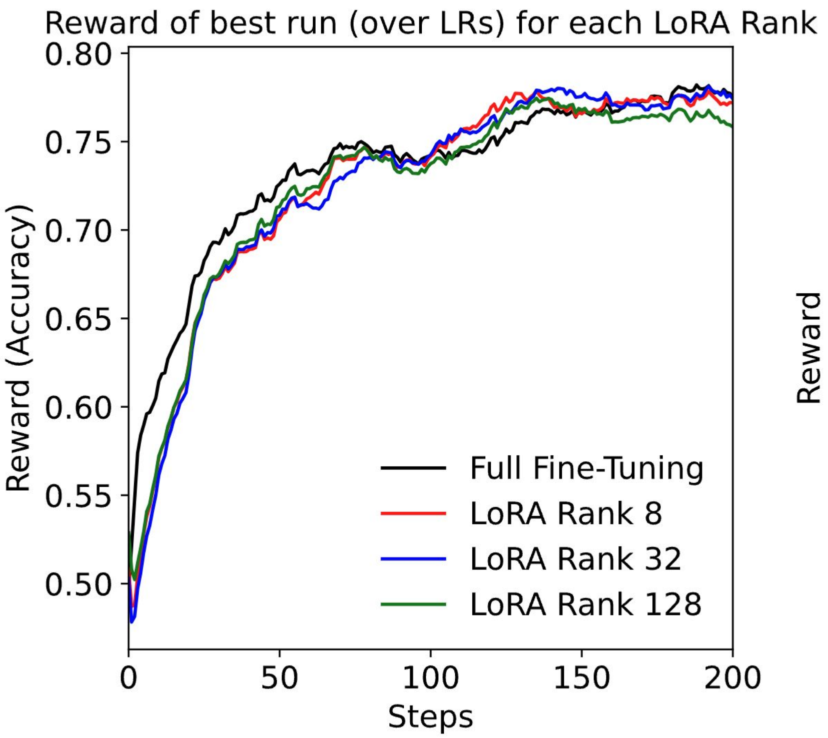
https://thinkingmachines.ai/blog/lora/

Attention-only LoRA significantly underperforms MLP-only LoRA



https://thinkingmachines.ai/blog/lora/

LoRA vs. Full Fine-tuning



https://thinkingmachines.ai/blog/lora/

LoRA Learns Less and Forgets Less

Dan Biderman[1,2], Jacob Portes[2], Jose Javier Gonzalez Ortiz[2], Mansheej Paul[2], Philip Greengard[1], Connor Jennings[2], Daniel King[2], Sam Havens[2], Vitaliy Chiley[2], Jonathan Frankle[2], Cody Blakeney[2], John P. Cunningham[1]
[1]**Columbia University** {db3236, pg2118, jpc2181}@columbia.edu
[2]**Databricks Mosaic Research** {jacob.portes, j.gonzalez, mansheej.paul, connor.jennings, daniel.king, sam.havens, vitaliy.chiley, jfrankle, cody.blakeney}@databricks.com

LoraHub: Efficient Cross-Task Generalization via Dynamic LoRA Composition

Chengsong Huang[†§*], Qian Liu[†*], Bill Yuchen Lin[◊*], Tianyu Pang[†], Chao Du[†], Min Lin[†]
[†]Sea AI Lab, Singapore
[§]Washington University in St. Louis, MO, USA
[◊]Allen Institute for AI, Seattle, WA, USA

Limitations of parameter-efficient tuning methods

Thank you!