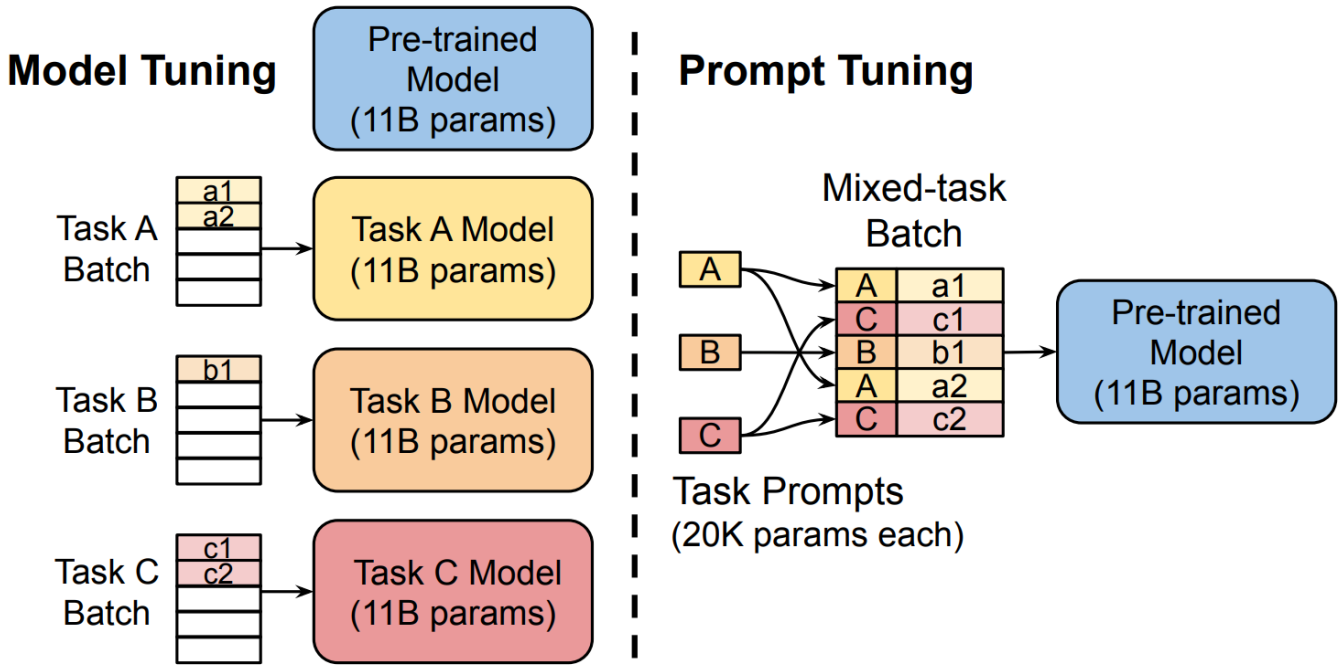
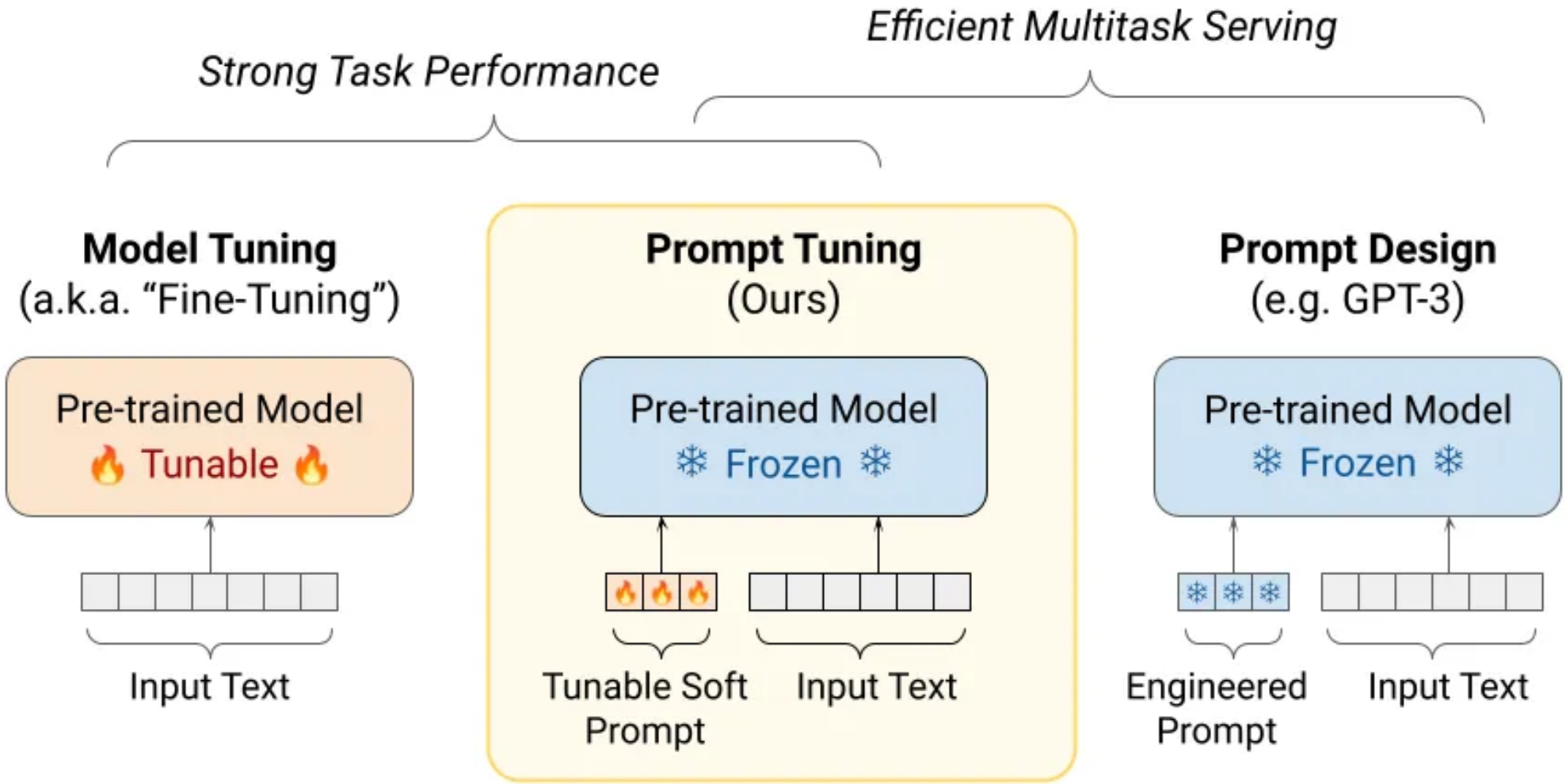
The Power of Scale for Parameter-Efficient Prompt Tuning

Brian Lester* Rami Al-Rfou Noah Constant

History

- The idea of Freezing
- Model Tuning (Fine-tuning): GPT and BERT (update all parameters)
- Prompt Design: GPT-3 (freeze model, adapt via handcrafted prompts)



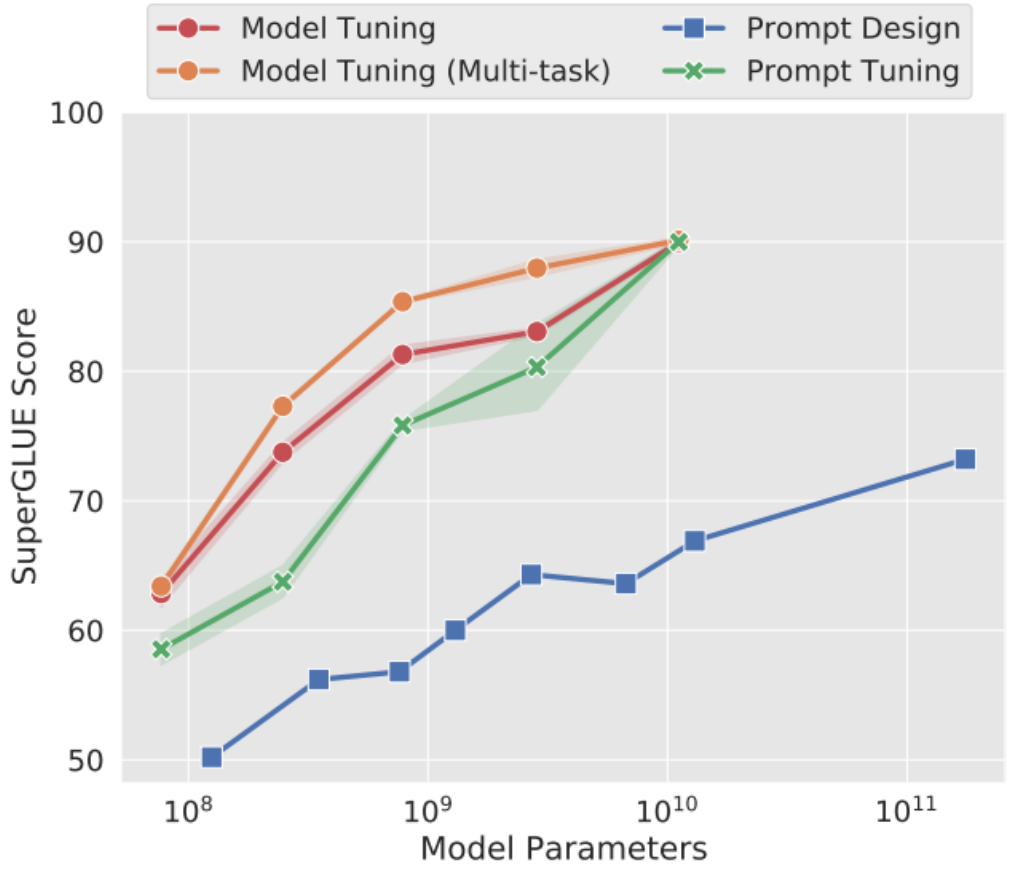

Efficient Multitask Serving
Strong Task Performance
Model Tuning
(a.k.a. "Fine-Tuning")
Pre-trained Model
🔥 Tunable 🔥
Input Text
Prompt Tuning
(Ours)
Pre-trained Model
❄ Frozen ❄
Tunable Soft Prompt
Input Text
Prompt Design
(e.g. GPT-3)
Pre-trained Model
❄ Frozen ❄
Engineered Prompt
Input Text

Prompt Tuning

- Freeze the entire pre-trained model, do not change its weights
- Learn only k tunable tokens per downstream task
- Prepending those tokens to the input text forming a soft prompt.

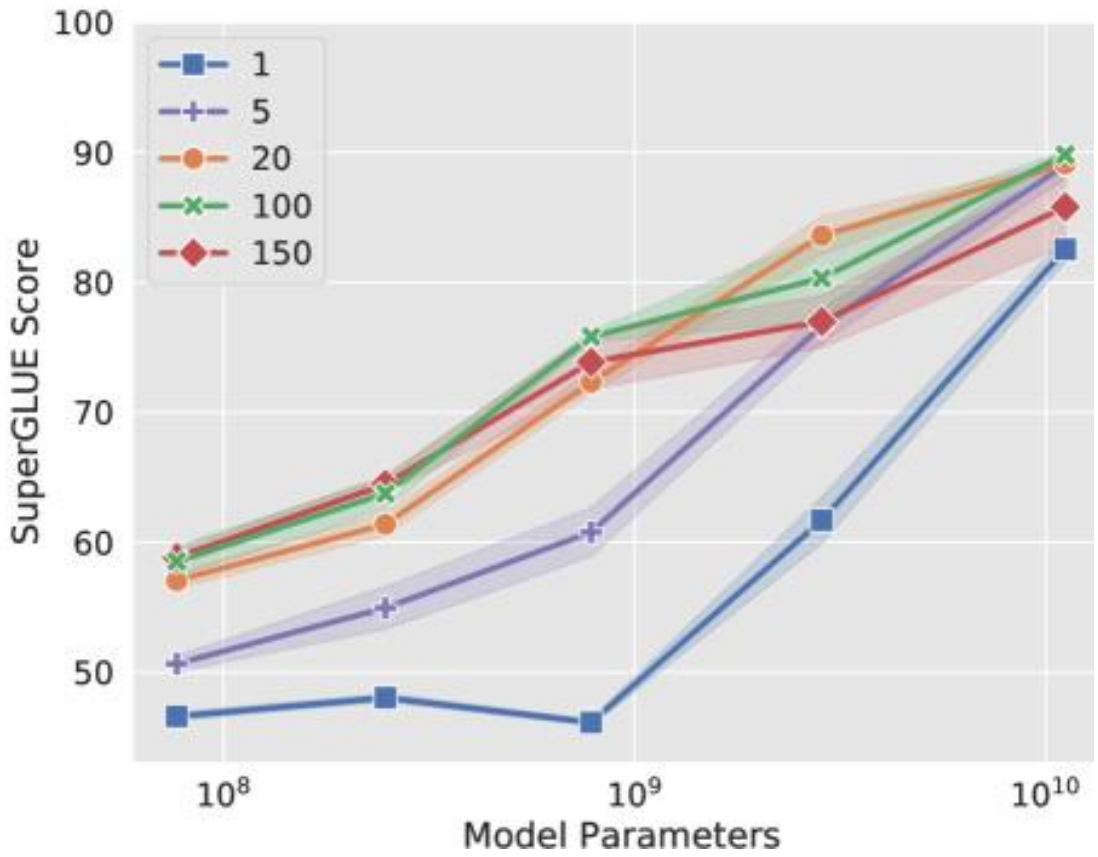
Example Token

- [P1, P2, …, P20, Input_1, Input_2, … Input_n], matrix of 20 * 768 (vector size = embedding dimension of model.



Ablation-Prompt Length

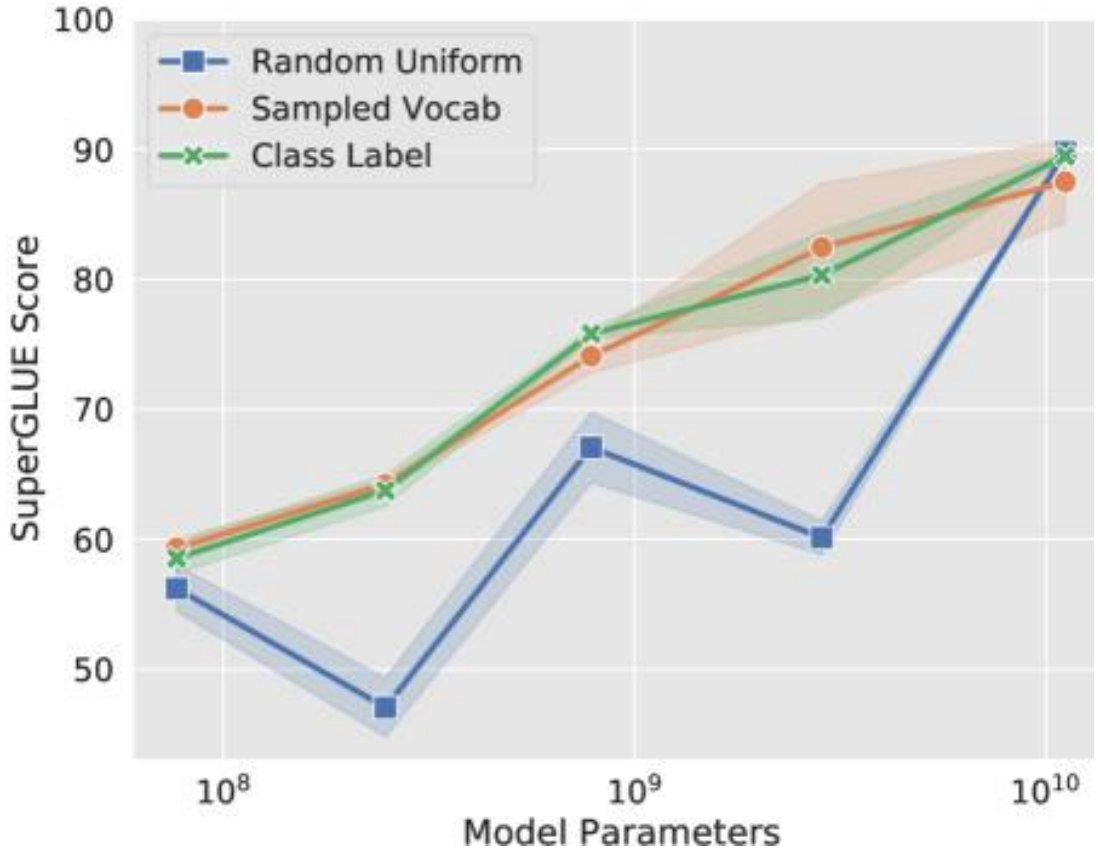
- Increasing prompt length beyond 1 token generally gives large improvement for most model sizes.
- XXL model still does reasonably well even with single token prompt.



WashU

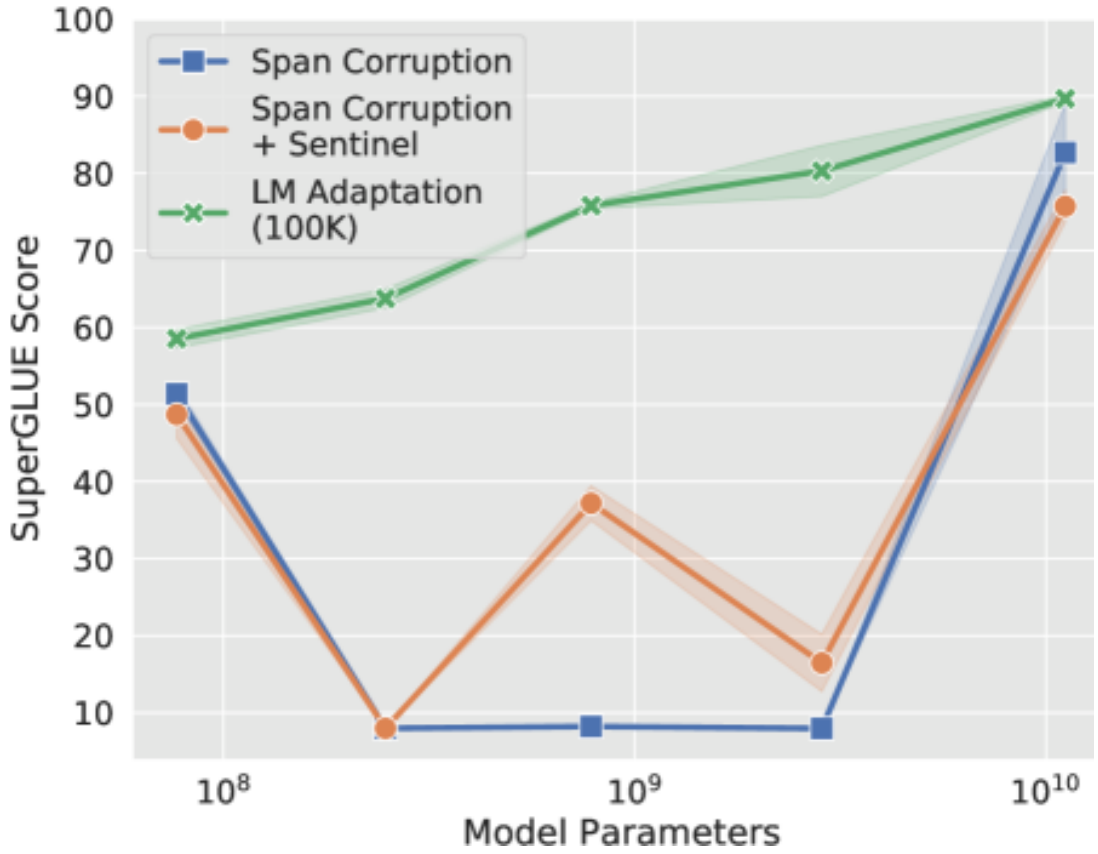
Ablation-Prompt Initialization

- Class Label initialization tends to perform best, especially in smaller and midsize models.
- Sampling vocab is middle ground.
- Differences in initialization diminish for XXL model.



Ablation: Pre-training Objective

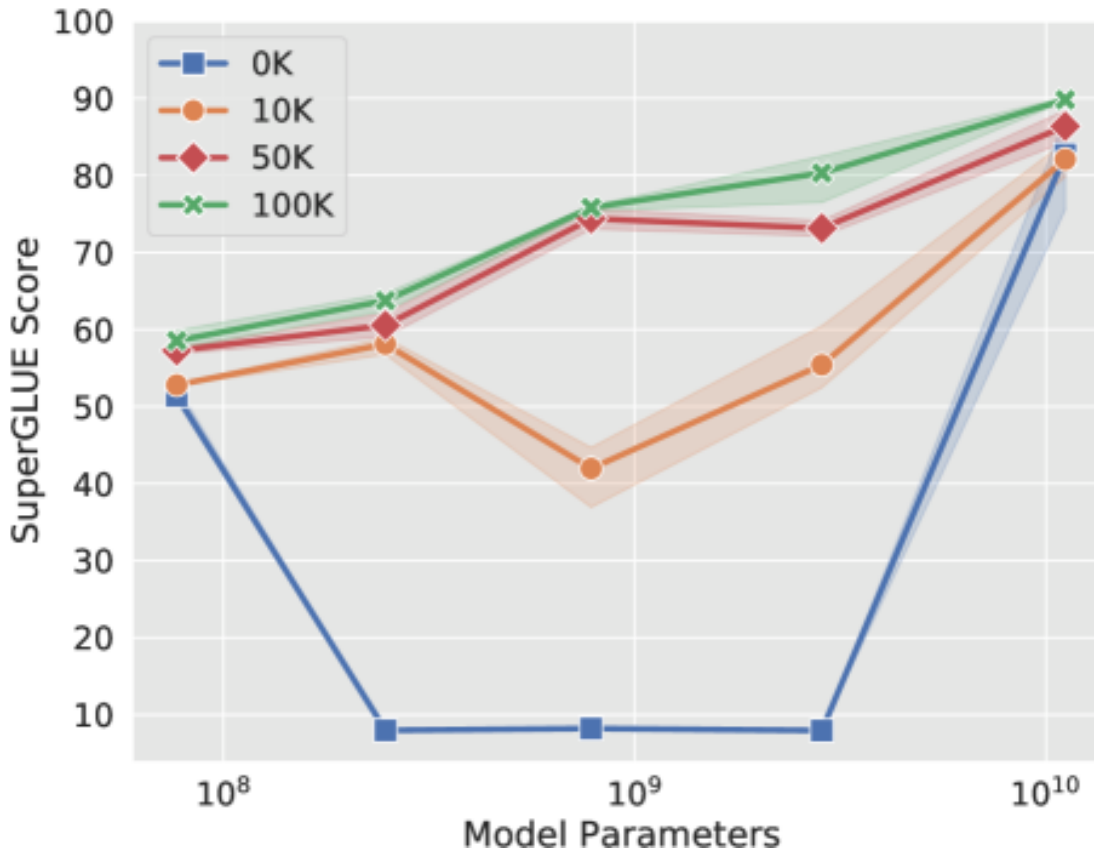
- LM adaptions give higher performance across model sizes.
- Adding sentinel to downstream targets while using span corruption yields little benefit.
- XXL models are more robust. Even non-ideal settings produce good results.



WashU

Ablation-LM Adaptation Steps

- More adaptation steps equals better performance, but up to a point.
- There is a diminishing return, which the XXL model gets smaller gains.



Domain Transfer

- Prompt tuning tends to outperform model tuning under domain shift, particularly when the shift is large.

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

Zero-shot Domain Transfer in Question Answering
Training: SQuAD
Testing: Out of domain datasets from MRQA 2019.

Zero-shot Domain Transfer in Paraphrase Detection
Training: Transfer Tested between community Q&A pairs and MRPC, which is news sentence pairs.
Examine both transfer directions.

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

- Ensembling gives gains over both average and the best individual prompt. Less storage cost than ensembling full models.

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**

Interpretability

- Nearest neighbor analyses Learned prompt tokens cluster semantically.
- Class label initialization tends to preserve class label embeddings.

Summary

- Scaling Effect
  - Small models: prompt tuning lags behind full model tuning.
  - Large models (billions of parameters): performance gap disappears, prompt tuning matches full fine-tuning.
- Efficiency
- Performance
  - Prompt-tuned small models can rival much larger GPT-3 models.

Summary - Takeaway

- Good with domain shifts
- Boosted performance through prompt resembling
- Potential interpretability of task behavior

Parameter-Efficient Transfer Learning for NLP

Neil Houlsby Andrei Giurgiu Stanisław Jastrzębski

Bruna Morrone Quentin de Laroussilhe Andrea Gesmundo

Mona Attariyan Sylvain Gelly

Supplementary Material for Parameter-Efficient Transfer Learning for NLP

Neil Houlsby Andrei Giurgiu Stanisław Jastrzębski

Bruna Morrone Quentin de Laroussilhe Andrea Gesmundo

Mona Attariyan Sylvain Gelly

- ❑Background & Motivation
- ❑Method: Adapters
- ❑Experiments & Results
- ❑Contributions & Limitations
- ❑Conclusion

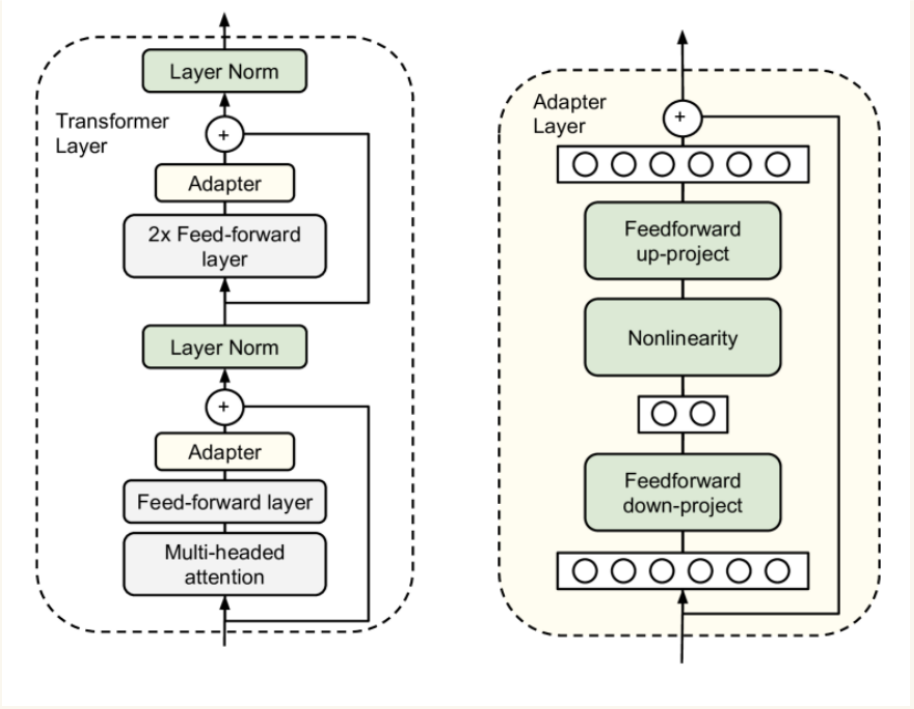
Background & Motivation

- Pretrained models (e.g., BERT) → strong performance but expensive to fine-tune
- Each new task requires training & storing the full model
- Need parameter-efficient transfer learning alternatives
- Goal: reduce cost while retaining accuracy

Method: Adapters

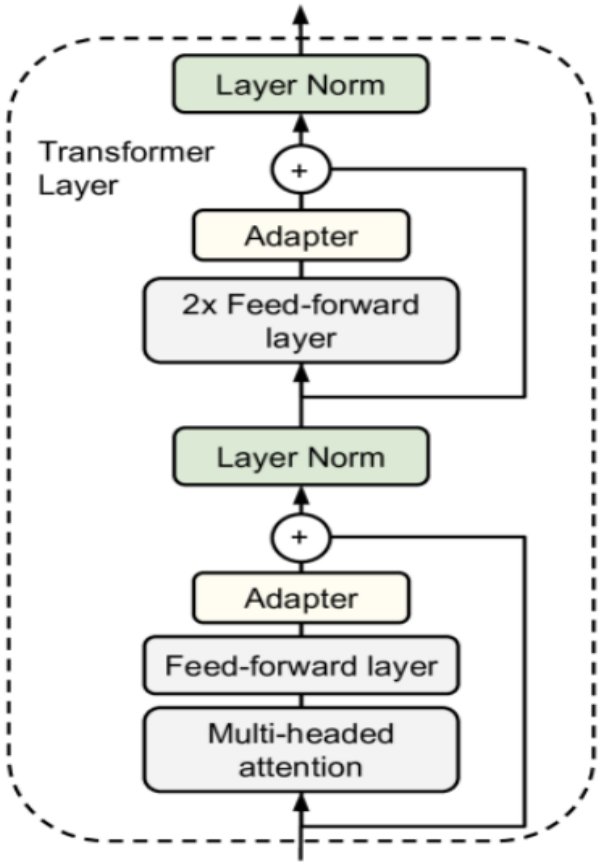
Part 1： Where to Insert Adapters

Part 2： *How Adapters Work*



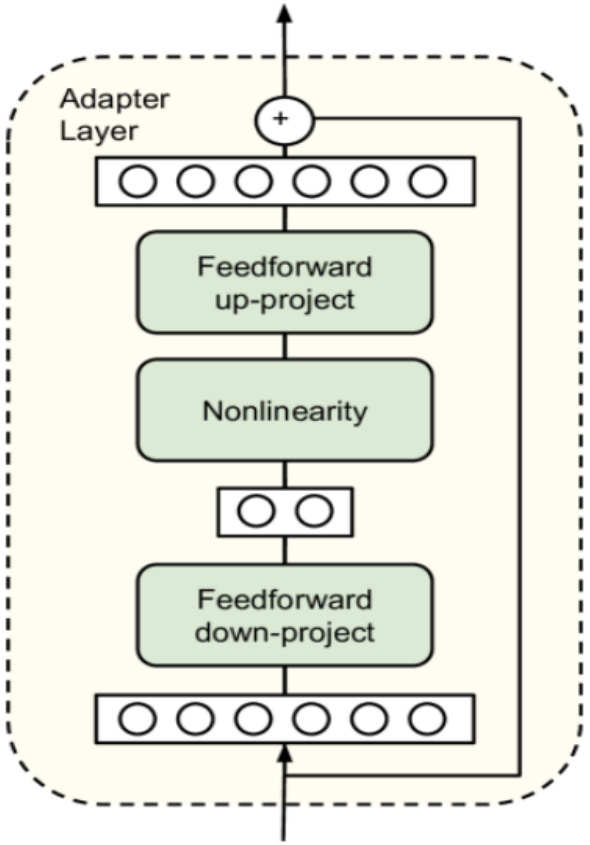
WashU

Method (Part 1): Where Adapters are Inserted



- Inserted after attention + feed-forward layers
- Only adapters + layer norms + classifier are trained
- Pretrained model parameters frozen

Method (Part 2): Adapter Architecture



- Down-project: reduce dimensionality (bottleneck)
- Nonlinearity
- Up-project: restore dimensionality
- Very few parameters compared to full layer

Intuition

- Pretrained model provides general language knowledge
- Adapters act as small task-specific “plugins”
- Enable efficient transfer across many different tasks

Experiments & Results

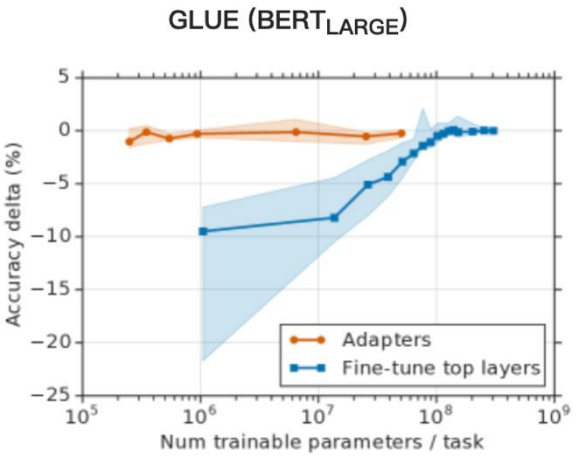
GLUE Benchmark

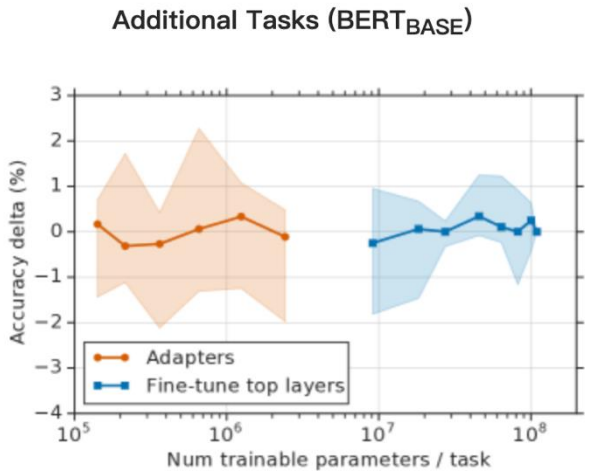
- Adapters achieve accuracy within 0.4% of full fine-tuning
- Require only 2–3% trainable parameters per task

	\pbox3cmTotal num									
params	\pbox3cmTrained									
params / task	CoLA	SST	MRPC	STS-B	QQP	$MNLI_m$	$MNLI_{mm}$	QNLI	RTE	Total	
$BERT_{LARGE}$	9.0 ×	100%	60.5	94.9	89.3	87.6	72.1	86.7	85.9	91.1	7(
Adapters (8-256)	1.3 ×	3.6%	59.5	94.0	89.5	86.9	71.8	84.9	85.1	90.7	7:
Adapters (64)	1.2 ×	2.1%	56.9	94.2	89.6	87.3	71.8	85.3	84.6	91.4	6{

WashU

Experiments & Results

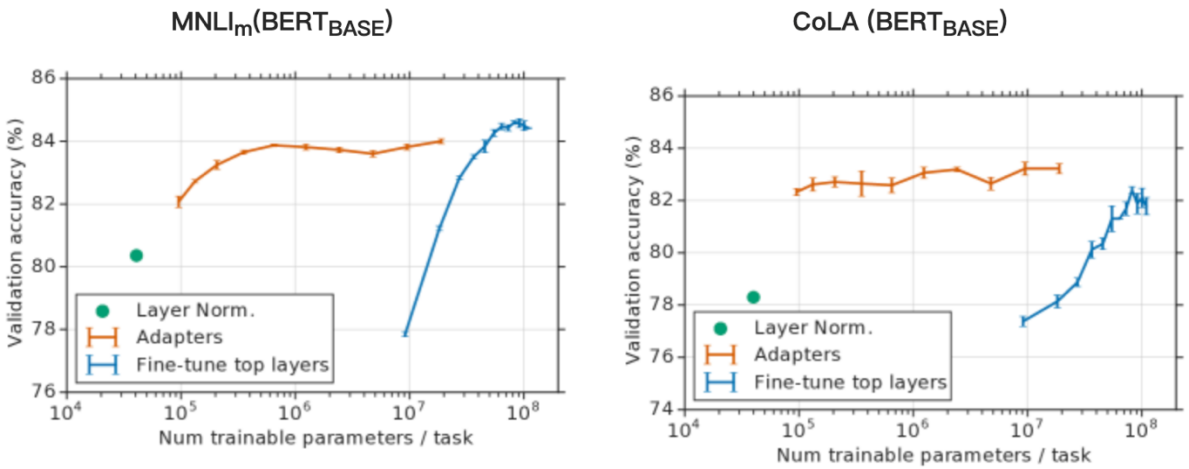




Efficiency

- Adapters: near full accuracy with far fewer parameters
- Top-layers fine-tuning: poor when parameters are limited
- Adapters clearly more parameter-efficient

Experiments & Results



Robustness

- Tested adapter placement across Transformer layers
- Performance stable regardless of where adapters are inserted
- Shows method is robust and flexible

Experiments & Results

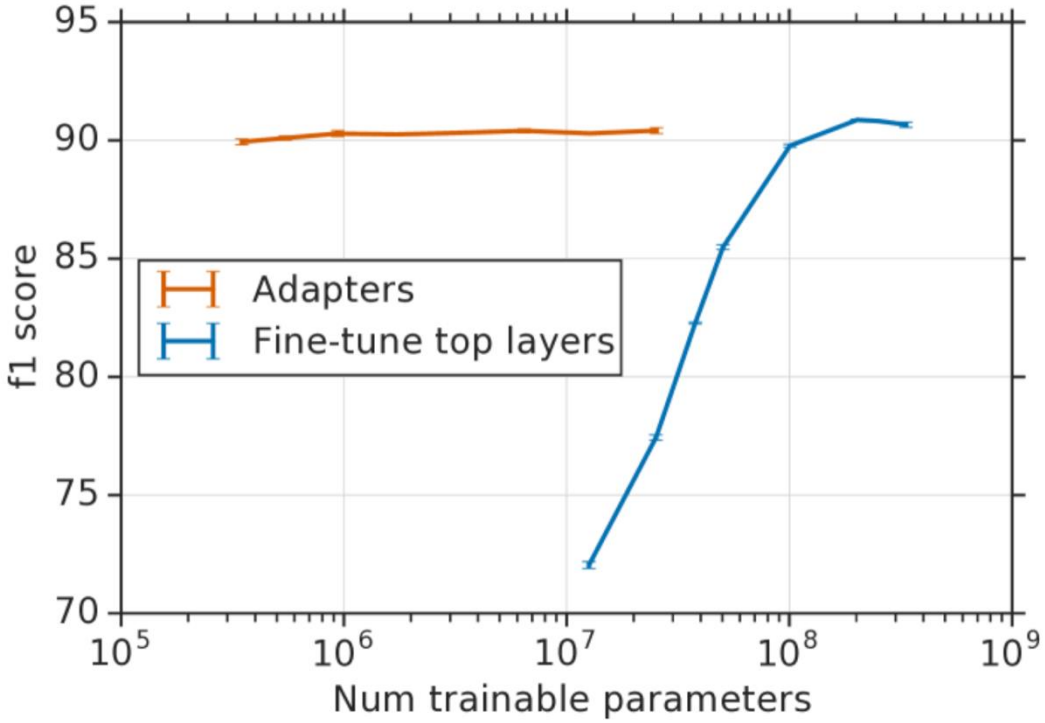
Parameter/Performance Trade-off

- Investigated different adapter sizes (bottleneck dimension m)
- m = reduced hidden size in adapter (e.g., 64, 128, 256)
- Trade-off: smaller m → fewer parameters, but potential drop in performance
- Results: even with small m (e.g., 64), adapters achieve strong accuracy
- Demonstrates adapters remain effective across parameter scales

Experiments & Results

SQuAD Extractive Question Answering

- Tested adapters on the **SQuAD v1.1 dataset** (reading comprehension)
- Compared with full fine-tuning
- Adapters achieve near-identical F1 and EM scores while training ~3% parameters
- Confirms effectiveness beyond GLUE → generalizable to QA tasks



WashU

Experiments & Results

Analysis & Discussion

- **Placement**: inserting adapters in both attention & feed-forward layers works best
- **Training cost**: faster to train than full fine-tuning due to fewer parameters
- **Robustness**: stable performance across tasks and adapter configurations
- **Limitation**: inference still runs full model; each task requires an adapter

Contributions & Limitations

- First to systematically study **adapters** for NLP
- Showed **near full fine-tuning accuracy** with only ~3% parameters
- Inference still requires the full model
- Each task needs a separate adapter

Summary

- Adapters = small & efficient
- 2–3% parameters ≈ full fine-tuning
- Efficient / stable / flexible
- Great for multi-task transfer

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)

WashU

Agenda

- Topic & Challenge
- Existing Solution
- LoRA-Proposed Solution
- Experimental Step&Key Result
- Related Work
- Conclusion&Future Work

WashU

1. Topic & Challenge

- **Topic**: Adapting large-scale pre-trained language models (e.g., GPT-3) to downstream tasks.

- **Challenge**: Full fine-tuning(Traditional way to adapt) requires retraining all parameters (175B for GPT-3), which is costly to store and deploy.

2. Existing Solution and Limitation

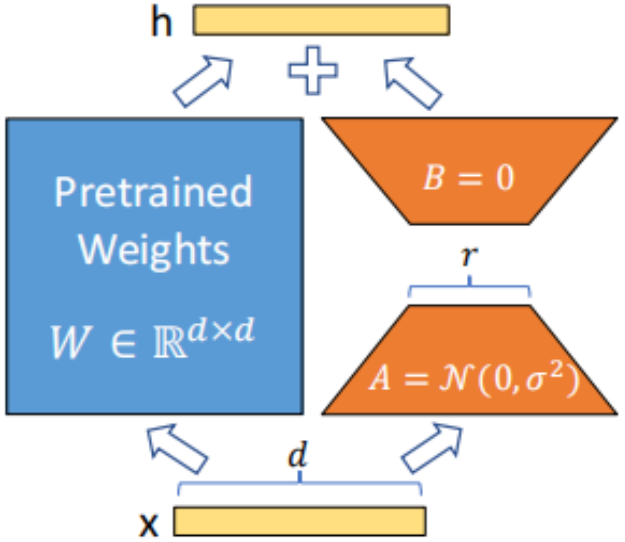
- **Adapter layers**:
  Add sequential modules → cause **extra inference latency**, especially in online/small-batch scenarios.
- **Prefix tuning**:
  Hard to optimize;
  Reduces effective sequence length;
  Performance unstable.

  Conclusion: Existing methods are inefficient or quality-limited in large-scale, latency-sensitive settings.

All in all, exsiting solution ain't good enough

3.LoRA — Proposed Solution

3.1 LOW-RANK-PARAMETRIZED UPDATE MATRICES



$$\mathbf{h = W_0x + \Delta Wx = W_0x + BAx} \qquad (*)$$

- $\mathbf{W_0}$ **:** The original weight matrix from the pretrained model.
  It is **frozen** during fine-tuning and does **not** get updated.
  Dimensions: $W_0 \in R^{d \times k}$
- **x:** The input vector.
- **ΔW=BA**: The LoRA-generated **update term** used to adapt the model to a new task.
- $\mathbf{B \in R^{d \times r}}$: low-rank matrix.
- $\mathbf{A \in R^{r \times k}}$: low-rank matrix.
- $\mathbf{r \ll min(d,k)}$

much smaller——> **low-rank decomposition——** > efficient and lightly

WashU

3.1 Comparision: LoRA vs Full-Fine-Tuning

Given an input vector $x \in R^k$, and a weight matrix $W \in R^{d \times k}$, the linear output is:

$$h = Wx$$

Parameter Update:

$$W \leftarrow W - \eta \cdot \nabla_W L(Wx, y)$$

Where:

- η is the learning rate
- $L(Wx,y)$
- is the loss function (e.g., cross-entropy)
- y is the target label

In Full-Fine-Tuning, the whole W need to be updated(W_0included)

3.1 Comparision: LoRA vs Full-Fine-Tuning

Method	Output Expression	Is W_0 Updated?
Full Fine-Tuning	h=Wx $h = Wx$	Yes – directly updated
LoRA	**h =W_0x+BAx** $h = W_0x + BAx$	No – kept frozen

In LoRA, only the small matrices A and B need to be trained, while the original weight matrix W_0 remains frozen.

3.2 Apply LoRA To Transformer

In principle, we can apply LoRA to any subset of weight matrices in a neural network to reduce the number of trainable parameters.

In the Transformer architecture, there are four weight matrices in the self-attention module (W_q, W_k, W_v, W_o) and two in the MLP module.

However, in this research, researchers limit their study on self-attention module for both simplicity and parameter-efficiency

3.3 Apply LoRA To Transformer

Which weight matrix in transformer should we apply LoRA to?
Validation accuracy on WikiSQL and MultiNLI after applying LoRA to different types of attention weights in GPT-3, given the same number of

	# 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**

Note that putting all the parameters in ΔW_q or ΔW_k results in significantly lower performance, while adapting both W_q and W_v yields the best result.

3.3 Apply LoRA To Transformer

What is the optimal rank r for LoRA?

Researchers Validation accuracy on WikiSQL and MultiNLI with different

	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

LoRA doesn't really sensitive to r because LoRA already performs competitively with a very small r.

3.3 Apply LoRA To Transformer

How does the adaption Matrix ΔW compare 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
$\|\|W_q\|\|_F = 61.95$	$\|\|\Delta W_q\|\|_F = 6.91$			$\|\|\Delta W_q\|\|_F = 3.57$		

First, ΔW has a stronger correlation with W compared to a random matrix, indicating that ΔW amplifies some features that are already in W.

Secondly, instead of repeating the top singular directions of W, ΔW only amplifies directions that are not emphasized in W.

WashU

4.Experimental Step & Key Result

Step1&2&3: Applyting different pre-training method on different scale model, starting from small and expanding to much larger, seeing the performance of LoRA and comparing to other 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**

Key Result:

- **LoRA perform really competitively on GLUEbenchmark in different scale model**
- **More parameter(higher rank), LoRA tend to perform well and finally go beyond FT**

WashU

4.Experimental Step & Key Result

Step4&5: answering if LoRA still prevails on NLG models, such as GPT-2 medium and large; and as a final test for LoRA, researchers scale up to GPT-3 with 175 billion parameters and see how LoRA matches.

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	$\mathbf{70.4}_{\pm.1}$	$\mathbf{8.89}_{\pm.02}$	$\mathbf{46.8}_{\pm.2}$	$\mathbf{72.0}_{\pm.2}$	$2.47_{\pm.02}$

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

Key Result:

- **LoRA performs better than prior approaches, including full fine-tuning.**
- **LoRA matches or exceeds the fine-tuning baseline on GPT-3**

WashU

5. Related Work

•**Transformer Language Models.** Transformer language models revolutionized NLP by combining large-scale pretraining with task-specific fine-tuning, with larger models like GPT-3 continuously pushing performance boundaries.

•**Prompt Engineering and Fine-Tuning.**While prompt engineering offers a lightweight way to adapt GPT-3 via careful input design, full fine-tuning—though more effective—is often infeasible for very large models due to computational constraints.

•**Parameter-Efficient Adaptation.** LoRA builds on the idea of adapter-based fine-tuning using a low-rank bottleneck, but differs by avoiding inference-time overhead. Unlike prompt tuning methods, it does not consume sequence length and has stronger scalability potential.

•**Low-Rank Structures in Deep Learning.** Low-rank structures are deeply rooted in both empirical practices and theoretical studies of machine learning. LoRA uniquely leverages this by applying low-rank updates to frozen pre-trained models for efficient downstream adaptation—something not explored in prior work.

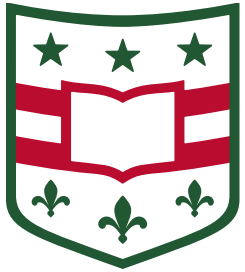
6. Conclusion and Future Work

- **Conclusion**

  Proposed LoRA: freeze pretrained weights, train low-rank matrices.
  Achieves near/full FT performance with 0.01% trainable params.
  No additional inference latency; easy task switching.
  Validated across RoBERTa, DeBERTa, GPT-2, GPT-3.

- **Future Work**

  Combine LoRA with other efficient tuning methods.
  Deeper understanding of why LoRA works.
  Systematic selection of which layers/matrices to adapt.
  Explore rank deficiency in pretrained weights.



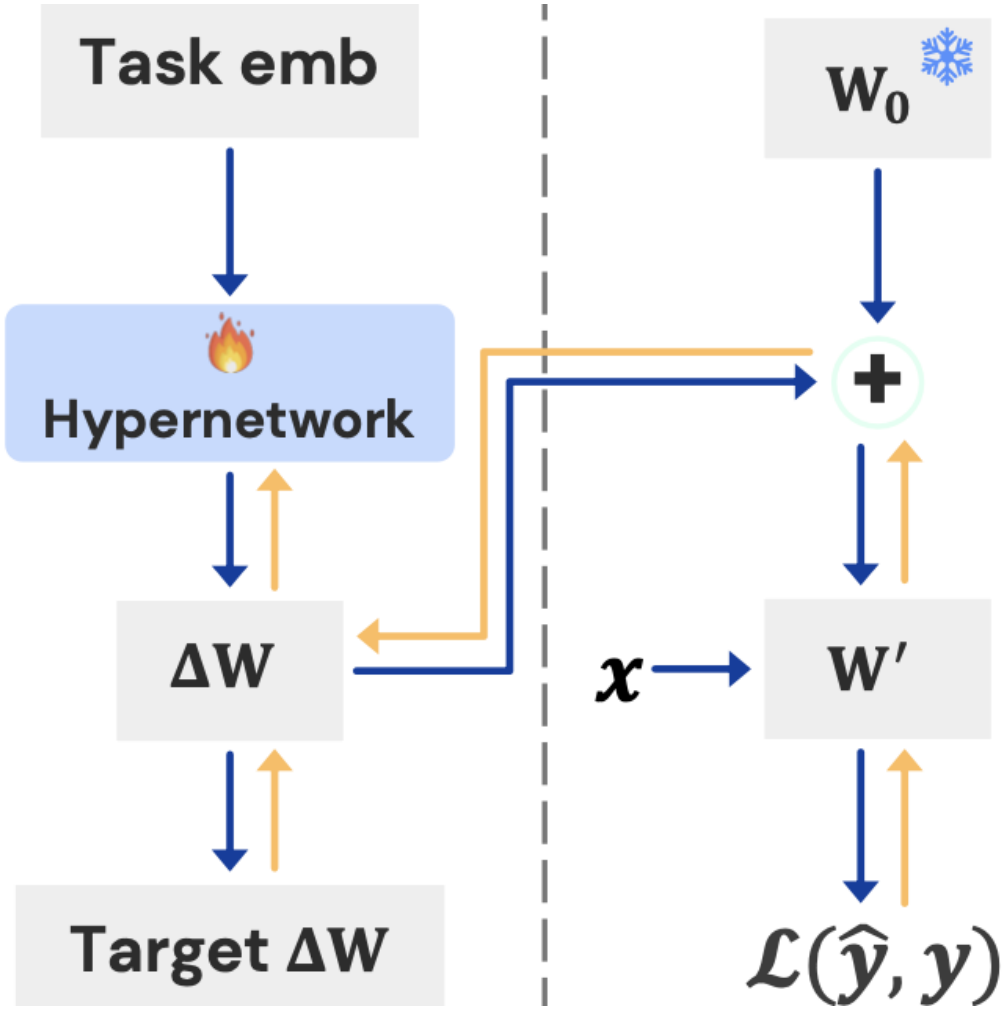
WashU

Text-to-LoRA: Instant Transformer Adaption

Rujikorn Charakorn [1] **Edoardo Cetin** [1] **Yujin Tang** [1] **Robert T. Lange** [1]

All group member

Introduction — Motivation



- Fine-tuning = expensive + slow
- Need separate LoRA per task

- Idea: Text-to-LoRA (T2L)
- Input: task description in natural language
- Hypernetwork instantly generates LoRA weights
- Add to frozen base model → fast task adaptation

WashU

Preliminaries

- Supervised Fine-Tuning (SFT)
- LoRA (Low-Rank Adaptation)
- Hypernetwork

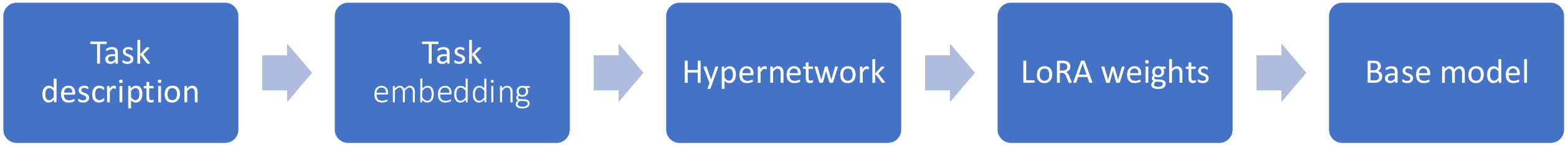
Preliminaries -LoRA

- Freeze base model weights
- Add trainable low-rank matrices (ΔW = BAx)
- Efficient fine-tuning with fewer parameters

$$h = W_0 x + \Delta W x = W_0 x + B^T A x$$

Preliminaries-Hypernetwork

- A neural network that **generates parameters** for another model
- Input: **task description embedding**
- Output: **LoRA weights** for the base model
- Enables **task-specific adaptation** in a single forward pass



Motivation and Setup

- LoRA adapters are effective but costly to fine-tune and store individually.
- The Goal: Use a hypernetwork thay, given a task description in natural language, generates LoRA weights directly.

LoRA WEIGHTS

$$\Delta W^i_{m,l} = h_\theta(\phi^i_{m,l}), \text{ with} \quad (3)$$

$$\phi^i_{m,l} = \texttt{concat}\left[f(z^i), E[m], E[l]\right], \quad (4)$$

$\Delta W^i_{m,L}$: The LoRA weights generated for task I, at module m, and layer l of the base model.

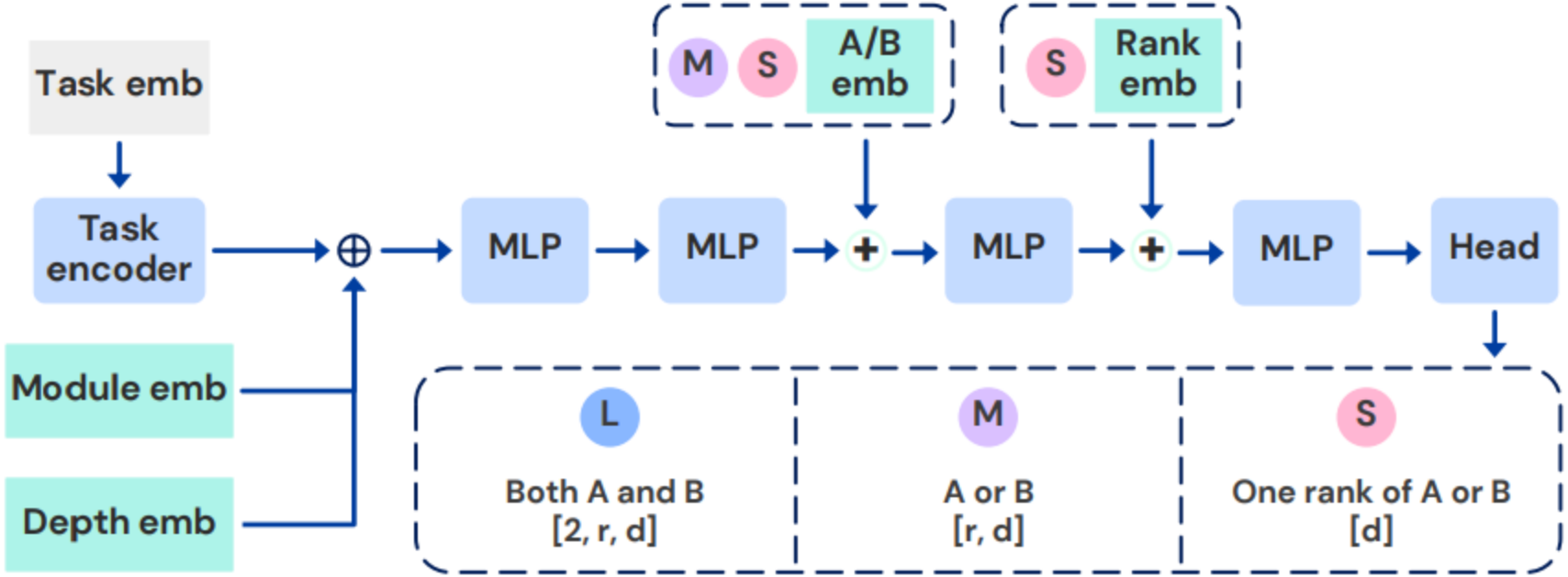
h_θ: The hypernetwork that outputs LoRA weights.

$\phi^i_{m,l}$: The input embedding given to the hypernetwork.

For each module and layer, the hypernetwork produces the corresponding LoRA weight matrix. Then concatenation of all three embeddings into a single vector $\phi^i_{m,l}$

Architectures (L,M,S)

- L (Large): Outputs both LoRA matrices A,B together. Highest parameter count, best performance.
- M (Medium): Shared output layer for A or B. Smaller than L, but still effective.
- S (Small): Outputs only one rank at a time. Most parameter-efficient, with strong inductive biases.



Training Objective for T2L

$$\theta = \underset{\theta}{\operatorname{argmin}}\ \mathbb{E}_{\mathcal{D}^i \sim \mathcal{D}, z^i \sim Z^i}\ \mathcal{L}_{\text{SFT}}(\mathcal{D}^i, \Psi, h_\theta(\phi^i)),$$

Input: task dataset D^{i} and task description z^{i}.
Output: hypernetwork-generated LoRA Δw^{i}
Loss: make the generated LoRA perform well on supervised training for that task.

We train the hypernetwork to output LoRA adapters such that the base model + adapter minimizes the supervised loss on task data.

Training Methods

Reconstruction Training

$$\mathcal{L}(\Omega, \theta) = \mathbb{E}_{\Delta W^i \sim \Omega} \, |\Delta W^i - h_\theta(\phi^i)|.$$

- Train T2L to **recreate existing LoRAs**.
- Pros: Simple, leverages existing LoRA libraries.
- Cons: Poor generalization to unseen tasks.

$$\theta = \underset{\theta}{\operatorname{argmin}} \; \mathbb{E}_{\mathcal{D}^i \sim \mathcal{D}, z^i \sim Z^i} \; \mathcal{L}_{\text{SFT}}(\mathcal{D}^i, \Psi, h_\theta(\phi^i)),$$

Supervised Fine-Tuning

- Train directly on task datasets with descriptions.
- Pros: Learns implicit clustering, better **zero-shot** generalization.
- Cons: Requires more training data.

Experiments – LoRA Compression

Setup: Mistral-7B-Instruct as base model; LoRAs rank=8.

Result: T2L can compress LoRAs and match (sometimes exceed) the performance of individually trained LoRAs.

Observation: Compression acts like regularization—sometimes improves generalization (e.g., PIQA, WG)

	ArcC (acc)	ArcE (acc)	BQ (acc)	GSM8K (acc)	HS (acc)	OQA (acc)	PIQA (acc)	WG (acc)	MBPP (pass@1)	Avg. (9 tasks)
Base model	65.4	77.8	71.6	40.9	49.7	54.2	72.8	45.0	43.1	55.8
One-Hot Task E.										
T2L (Recon) L	76.4	89.9	89.4	**53.8**	92.6	85.0	69.7	**51.2**	**52.6**	**73.4**
T2L (Recon) M	**76.7**	89.9	89.4	53.2	92.6	85.0	69.9	**51.4**	**52.9**	**73.4**
T2L (Recon) S	75.2	88.8	87.4	50.9	89.1	75.6	**83.9**	**58.1**	48.1	73.0
Task Description E.										
T2L (Recon) L	76.6	89.8	89.4	**53.9**	92.6	85.0	69.6	**51.2**	51.8	73.3
T2L (Recon) M	76.5	89.9	89.4	**53.9**	92.5	84.9	**70.4**	**51.6**	**52.8**	**73.5**
T2L (Recon) S	75.4	88.8	87.8	49.1	89.7	76.7	**84.2**	**56.9**	48.0	73.0
Task-specific LoRAs	76.6	89.9	89.4	53.5	92.6	85.0	69.9	51.1	52.1	73.3

WashU

Experiments - Zero-Shot LoRA Generation

Setup: T2L trained with 479 tasks from SNI dataset.

Result: Generates LoRAs for unseen tasks just from descriptions.

Performance: Beats multi-task LoRA baselines and approaches task-specific LoRA performance.

Example: T2L (L) achieved 73.9 avg. vs Multi-task LoRA 71.9

Takeaways

- T2L = Hypernetwork + Task Descriptions → LoRAs.
- Trade-offs: Larger architectures (L) = better performance; smaller ones (S) = efficiency.
- Three contributions
  - Compress many LoRAs into one model.
  - Generate new LoRAs from text descriptions (zero-shot).
  - Achieve competitive results with efficient training.

Ablations and Analyses

Does Increasing Training Compute Proportional to the Number of Training Tasks effect the performance of T2L?

Researchers explores the scalability of T2L by increasing the number of training tasks while proportionally increasing the compute budget, finding that generally, the performance improves while increasing task but it has limited capicity

	Number of tasks	Max SGD steps	ArcC (acc)	ArcE (acc)	BQ (acc)	GSM8K (acc)	HS (acc)	OQA (acc)	PIQA (acc)	WG (acc)	HE (pass@1)	MBPP (pass@1)	Avg.
T2L (SFT) L	479	1M	77.5	88.9	85.0	45.8	66.5	75.5	82.1	64.2	39.2	51.9	67.7 ▲
	256	640K	77.3	88.1	84.3	46.0	64.5	75.7	81.9	64.0	39.8	52.1	67.4 ▲
	128	320K	76.6	88.4	85.2	46.1	67.0	74.3	81.6	55.0	38.2	45.7	65.8 ▼
	64	160K	75.5	88.0	84.5	43.9	65.5	70.7	80.5	59.5	39.8	51.7	66.0
T2L (SFT) M	479	1M	77.2	89.0	84.3	45.2	65.1	76.1	81.8	64.0	41.3	50.5	67.5 ▲
	256	640K	75.9	89.3	85.0	47.0	65.3	73.7	81.6	63.2	39.8	48.6	66.9 ▲
	128	320K	74.9	88.3	85.5	44.9	64.8	72.8	80.7	61.6	42.9	43.5	66.0 ▲
	64	160K	73.6	87.7	84.5	43.2	64.6	70.5	79.9	56.0	40.7	51.4	65.2
T2L (SFT) S	479	1M	77.7	88.3	85.0	46.3	65.3	73.9	82.4	61.9	34.6	36.6	65.2 ▼
	256	640K	76.0	88.7	83.8	47.3	68.0	71.6	82.3	61.0	39.0	41.2	65.9 ▲
	128	320K	74.9	88.0	84.5	44.4	66.2	72.2	82.0	59.3	39.0	47.3	65.8 ▲
	64	160K	75.4	88.4	85.0	43.1	64.8	70.7	81.5	51.6	39.4	46.7	64.7

Ablations and Analyses

Does the choice of text embedding model affect the quality of LoRA generated by T2L?

Researchers compare 2 embedding model: gte-large-en-v1.5 and Mistral-7B-Instruct

Conclusion:

- Both models yield high-quality LoRA adapters with similar performance.
- T2L is **robust to the choice of task embedding model**, showing good generalization capabilities.

	gte			Mistral		
Avg. Benchmark performance	S	M	L	S	M	L
	65.8	66.0	65.8	64.7	66.2	66.0
Avg.		**65.9**			**65.6**	

Ablations and Analyses

Does the quality or alignment of task descriptions affect the LoRA performance generated by T2L?

Researchers use 4 types of task descriptions in training:

- **Train** – Original descriptions used in training
- **Eval** – New, unseen descriptions for the same task
- **Train (random)** – Descriptions from **other tasks**
- **Random strings** – Completely unrelated/random text

	Aligned		Unaligned	
	Train	Eval	Train (random)	Random strings
T2L **L**	73.3	73.6	49.1	68.2
T2L **M**	73.5	70.2	49.5	68.5
T2L **S**	73.0	72.9	55.7	53.9
Avg.	**73.3**	**72.2**	**51.4**	**63.5**

Conclusion:

T2L requires semantically aligned task descriptions to perform well.

Ablations and Analyses

two training strategies SFT and Reconstruction which one leads to better zero-shot performance?

	Recon			SFT		
Benchmark	S	M	L	S	M	L
performance	61.8	61.7	62.0	64.8	66.5	67.5
Avg.		**61.8**			**66.3**	

SFT perform better obviously

Why?
Because pre-trained LoRA adapters for similar tasks may lie in very different parameter spaces, making them harder to compress reliably.

Related Work

- **Hypernetworks**: Used in multi-task and continual learning, but typically rely on task IDs and lack zero-shot ability.
- **Zero-shot adaptation**: Prior works like Hyperdecoders and Gisting are less flexible.
- **Contribution**: First to achieve text-based LoRA generation and cross-task generalization.

Discussion & Limitations

- **Discussion**:
  - Task descriptions generated automatically by GPT-4o mini to ensure quality.
  - Approach is extendable to multimodal models (e.g., vision-language).
  - Potential: train T2L with small models, transfer to larger models.
- **Limitations**:
  - Only outputs LoRA as the adaptation space.
  - Compression could be improved further.
  - Zero-shot performance still below dedicated single-task LoRAs.

Conclusion

- T2L provides plug-and-play, low-cost adaptation using only natural language descriptions.
- Unifies LoRA compression and zero-shot task generation.
- Represents a step toward more automated and universal adaptation of large models.