Generative Artificial Intelligence

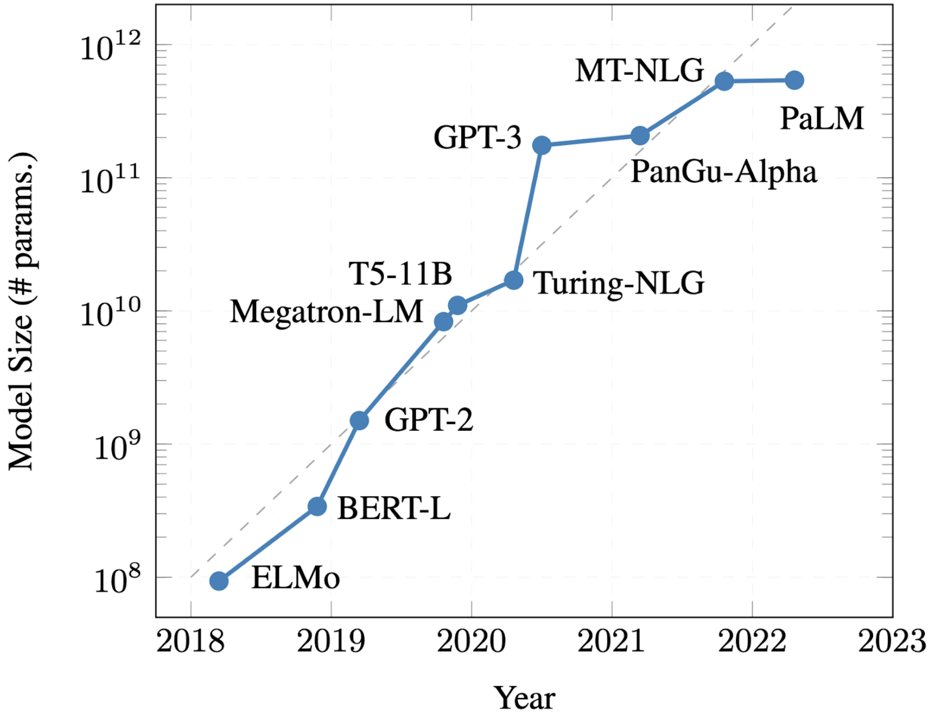
Parameter Efficient Fine-Tuning

Outline

1. PEFT Introduction
2. PEFT Theory
    - Intrinsic Dimensionality
3. PEFT Current Development & Method
    - Adapters
    - Prompt Tuning
    - Bitfit
    - LoRA
    - MAM Adapters
    - S4

PEFT
Introduction

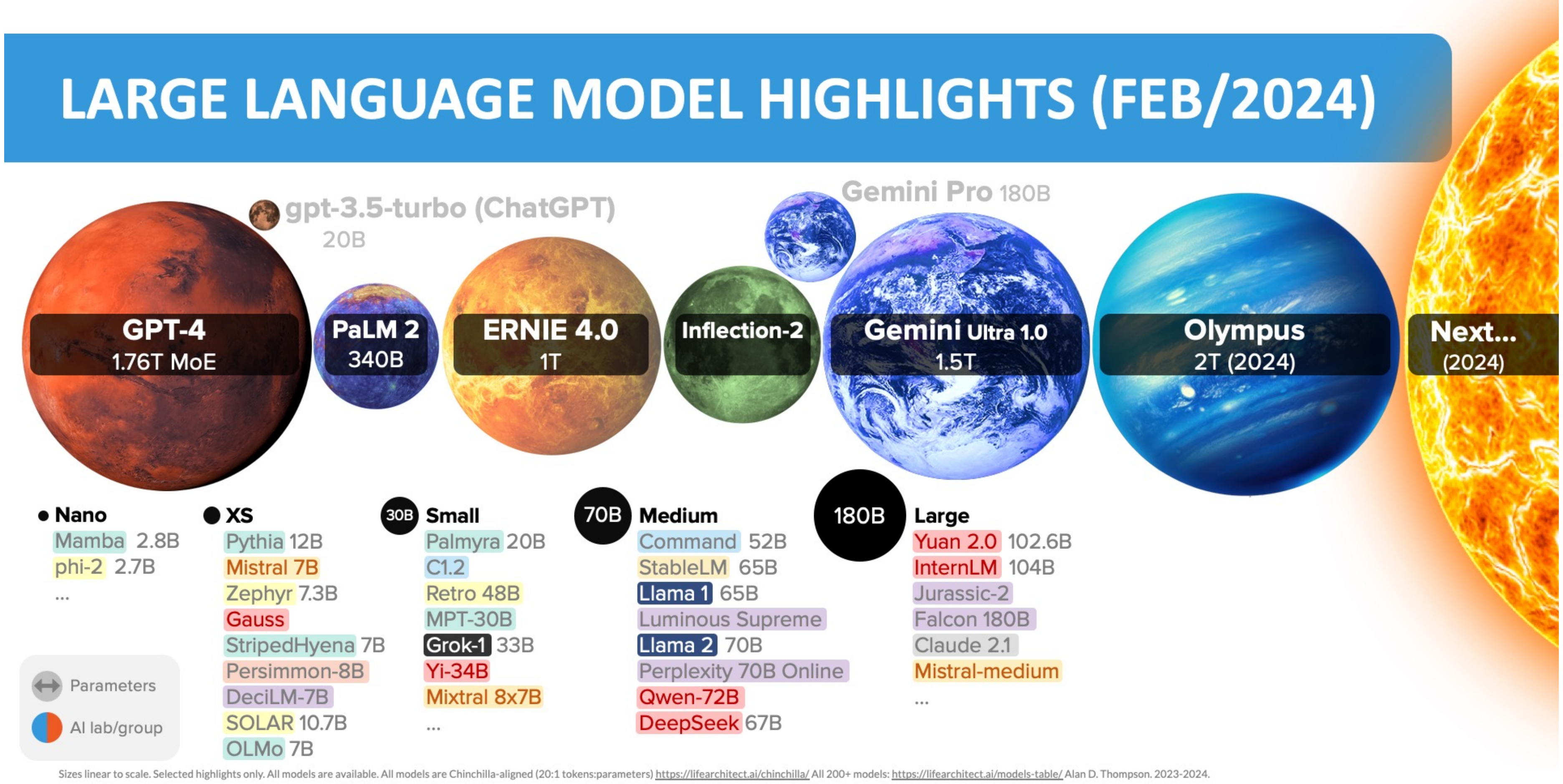
LLM Full Finetune Predicament



Model Name	Model Size	Developer
PaLM	540 Billion	Google
MT-NLG	530 Billion	Microsoft NVIDIA
PanGu-α	200 Billion	PengCheng
GPT-3	175 Billion	OpenAI
Turing-NLG	17.2 Billion	Microsoft

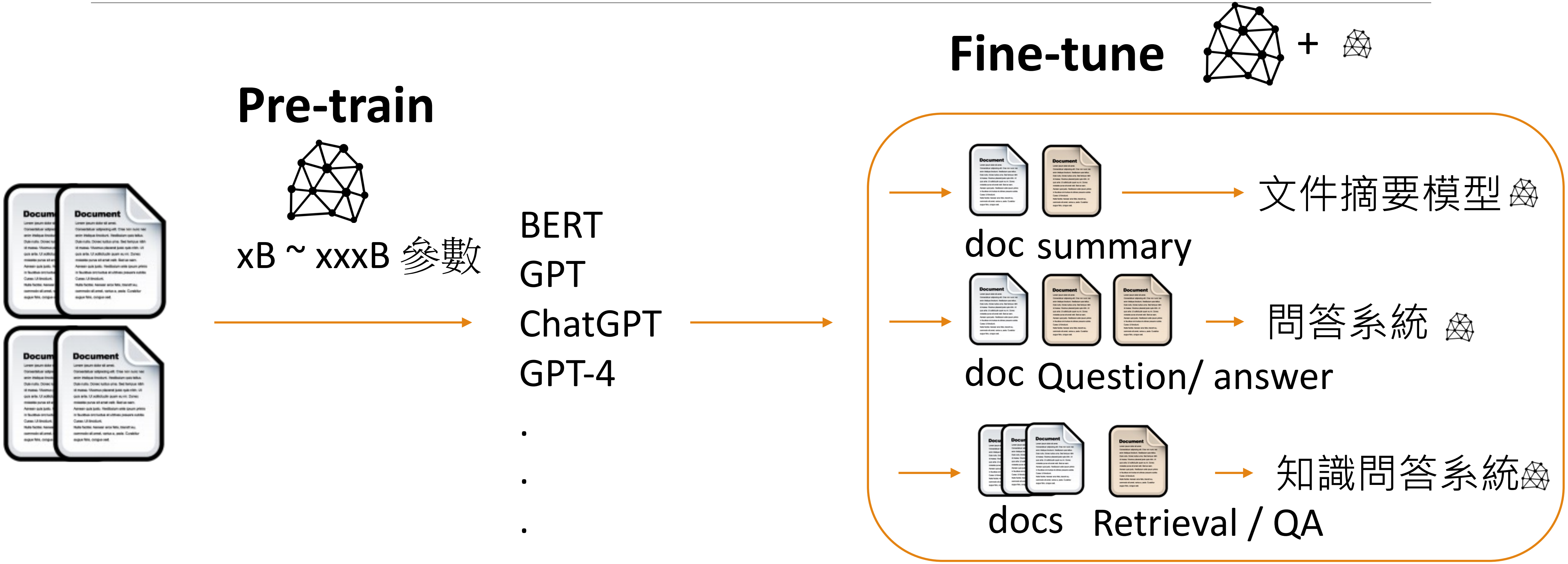
Treviso, Marcos, et al. "Efficient methods for natural language processing: A survey." *Transactions of the Association for Computational Linguistics* 11 (2023): 826-860.

LLM Full Finetune Predicament



Source: Inside language models (from GPT-4 to PaLM) – Dr Alan D. Thompson – Life Architect

Large Language Model (LLM) enabled NLP applications



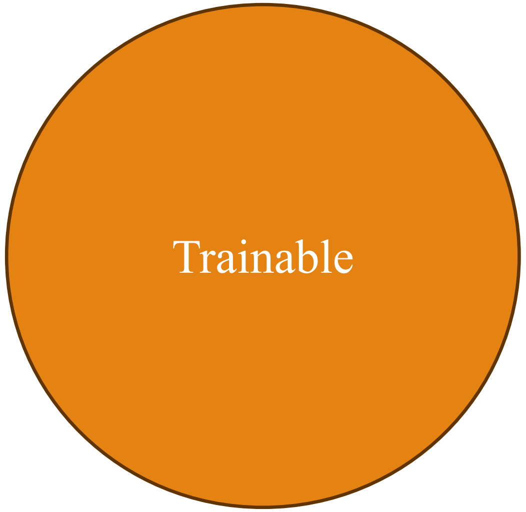
GPU Memory Estimated – Full Finetune

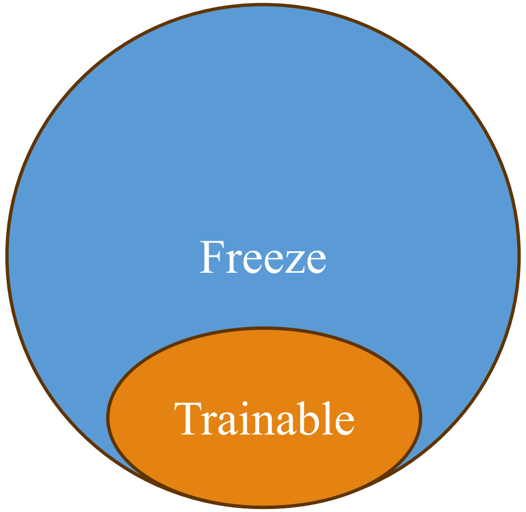
Llama 2-7B, 16-bit float, seq 4096		
CUDA		~1Gb
Model weights	size(float) * $N_{parameter}$	13.03Gb
Gradients	size(float) * $N_{trainable}$	13.03Gb
Hidden states	~size(float) L (20 seq + 3 seq^2)	3.16Gb (batch size = 1)
Optimizer states	2 * size(float) * $N_{trainable}$	26.06Gb

L : Number of layers in model (eq. 32 layers)
H : Number of attention heads (eq. 32 heads)

Estimate: 56.28Gb

From Fine-tuning to Parameter-efficient Fine-tuning





Full Fine-tuning
Update **all model parameters**

Parameter-efficient Fine-tuning
Update a **small subset** of model parameters

GPU Memory Estimated – Train Less Parameters

Training only 0.2M parameters

Llama 2-7B, 16-bit float, seq 4096		
CUDA		~1Gb
Model weights	size(float) * $N_{parameter}$	13.03Gb
Gradients	size(float) * $N_{trainable}$	**0.4Mb**
Hidden states	~size(float) L (20 H seq + 3 seq^2)	3.16Gb (batch size = 1)
Optimizer states	2 * size(float) * $N_{trainable}$	**0.8Mb**

L : Number of layers in model (eq. 32 layers)
H : Number of attention heads (eq. 32 heads)

Estimate: 17.19Gb

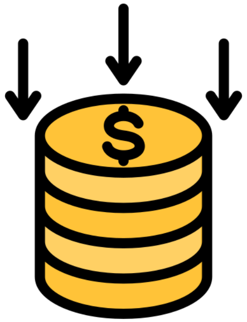
Haven't We Seen This Before ?

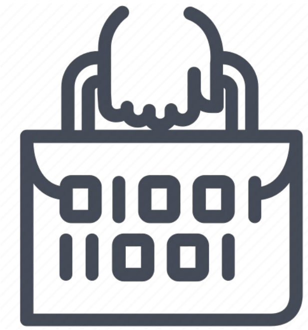
- Updating the last layer was common in computer vision
- In NLP, people experimented with static and non-static word embeddings
- ELMo did not fine-tune contextualized word embeddings

Matthew E. Peters, et al. "Deep Contextualized Word Representations." Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers). 2018.

Benefits of PEFT

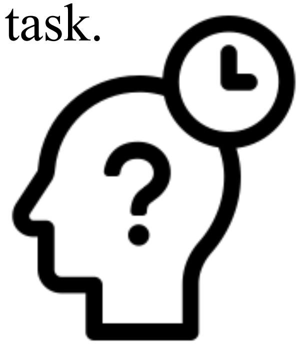
1. Decreased computational and storage costs
   - One significant benefit of parameter-efficient fine-tuning lies in its reduced computational and storage demands compared to the high costs associated with full fine-tuning.
2. Portability
   - It offers applicability across various domains and tasks. PEFT achieves this by **introducing a limited set of task-specific parameters while preserving the general-purpose parameters of the pre-trained model**, facilitating seamless transfer to new unlabeled datasets.





Benefits of PEFT

3. Overcoming catastrophic forgetting
   - **When extensively pre-trained models undergo full fine-tuning on a novel task, it often leads to the model forgetting previously acquired knowledge** from its pre-training phase. The insights gained from the prior task are overridden by the updates tailored for the new task.
   - Through PEFT, **only a small subset of parameters undergo modification during fine-tuning, leaving the majority of the model - which encapsulates general language knowledge from pre-training - unchanged**. This focused adjustment helps prevent the loss of knowledge from the original pre-training task.

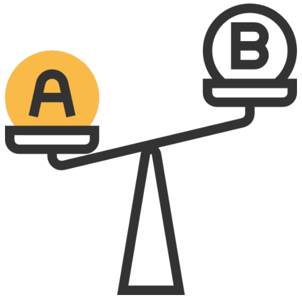


Benefits of PEFT

4. Better performance in low-data regimes
    - Large Language models of considerable size possess hundreds of millions or even tens of millions of trainable parameters, **thereby increasing the risk of overfitting when fine-tuned on tasks with limited labeled examples**.
    - Nevertheless, **PEFT selectively trains only a small subset of these parameters, leveraging knowledge from the robust pre-trained model**. This approach enables the model to generalize more effectively as it still heavily relies on the extensive pre-trained representation.

Benefits of PEFT

5. Performance comparable to full fine-tuning
    - **Extensive studies have demonstrated that parameter-efficient techniques can match or surpass the performance achieved by fully fine-tuning pre-trained models**, despite adjusting only a minute fraction of parameters.
    - For instance, **research indicates that incorporating a small adapter module during fine-tuning yields performance results within 1% of fully adapting BERT across various natural language understanding benchmarks**. This illustrates that PEFT isn't just a workaround with compromised accuracy, but rather a method capable of achieving similarly robust outcomes while leveraging its numerous advantages over full fine-tuning.



Trainable Parameters Comparison

Method	RTE (Acc)	Trainable parameters (M)	Ratio
Full Fine-tune Acc	83.75%	184	100%
AdaLoRA	88.09%	1.27	0.69%
LoRA	86.60%	0.8	0.43%
Random Rank LoRA† **(rank : 1 - 16)**	85.56%	0.62	0.34%
Random Rank LoRA† **(rank : 8 - 24)**	85.16%	1.18	0.64%
Random Rank LoRA† **(rank : 16 - 32)**	85.13%	1.77	0.96%
Random Rank LoRA† **(rank : 32 - 64)**	84.91%	3.54	1.92%

† : Average of Samples (10 samples)

* Model: DeBERTa-v3-base

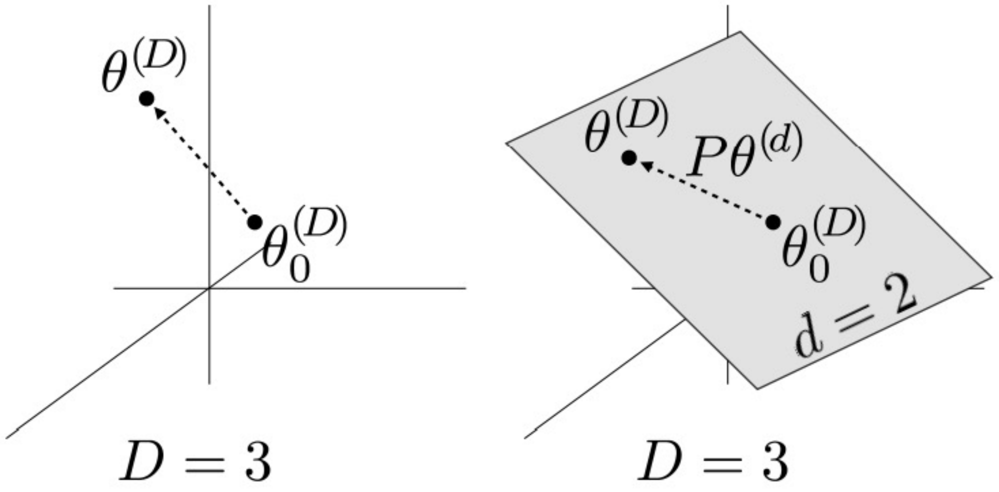
** Ratio means compare with Full Fine-tune

PEFT
Theory

Intrinsic Dimensionality

The definition of Intrinsic Dimensionality (d_{int})

- **Intrinsic dimensionality refers to the dimensionality of the solution set within the overall parameter space**
  - If a high-dimensional space (D = 1000), the effective dimensionality ($\mathbf{d_a}$ = 10) means we optimize by random subspace (d = 10) can achieve **"a %"** performance of original optimizing outcome.

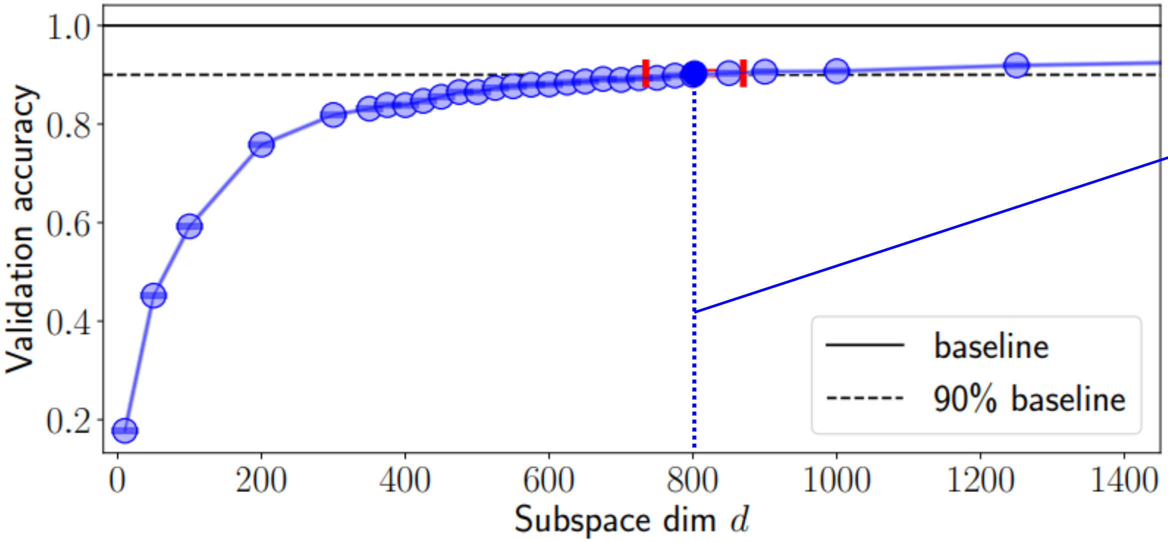


Li, Chunyuan, et al. "Measuring the Intrinsic Dimension of Objective Landscapes." *International Conference on Learning Representations*. 2018.

Intrinsic Dimensionality

Measuring the Intrinsic Dimension of Objective Landscapes

- Define $\mathbf{d_{100}}$ as the intrinsic dimension of the **"100%"** solution: solutions whose performance is statistically indistinguishable from baseline solutions



In this example, the d_{90} is approximately equal to 800.

Li, Chunyuan, et al. "Measuring the Intrinsic Dimension of Objective Landscapes." *International Conference on Learning Representations*. 2018.

Intrinsic Dimensionality

Many problems have smaller intrinsic dimensions than one might suspect

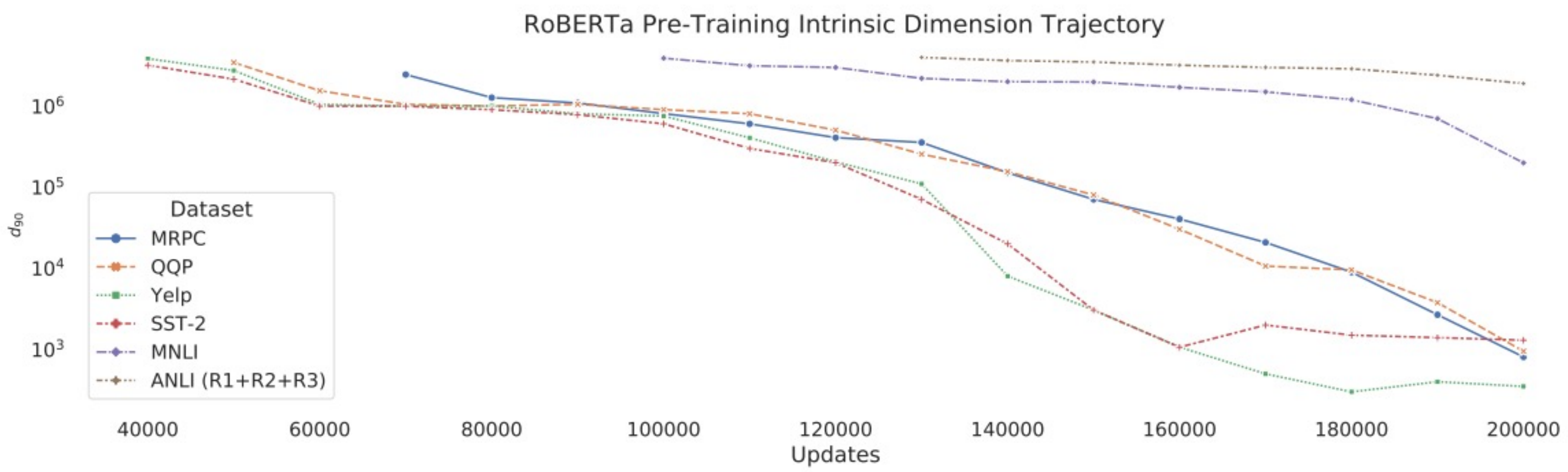
Dataset	**MNIST**		**CIFAR-10**		**Inverted Pendulum**	**Humanoid**	**Atrai Pong**
Network Type	**FC**	**LeNet**	**FC**	**LeNet**	**FC**	**FC**	**ConvNet**
Parameter Dim. D	199,210	44,426	656,810	62,006	562	166,673	1,005,974
Intrinsic Dim. d_{90}	750	290	9,000	2,900	4	700	6,000
d_{90} / D	**0.38%**	**0.65%**	**1.37%**	**4.68%**	**0.7%**	**0.42%**	**0.60%**

Li, Chunyuan, et al. "Measuring the Intrinsic Dimension of Objective Landscapes." *International Conference on Learning Representations*. 2018.

Intrinsic Dimensionality

Pre-training implicitly minimizes intrinsic dimension

- Compute d_{90} for six datasets: MRPC, QQP, Yelp Polarity, SST-2, MNLI, and ANLI
- See that **the intrinsic dimensionality of RoBERTa-base monotonically decreases as we continue pre-training**

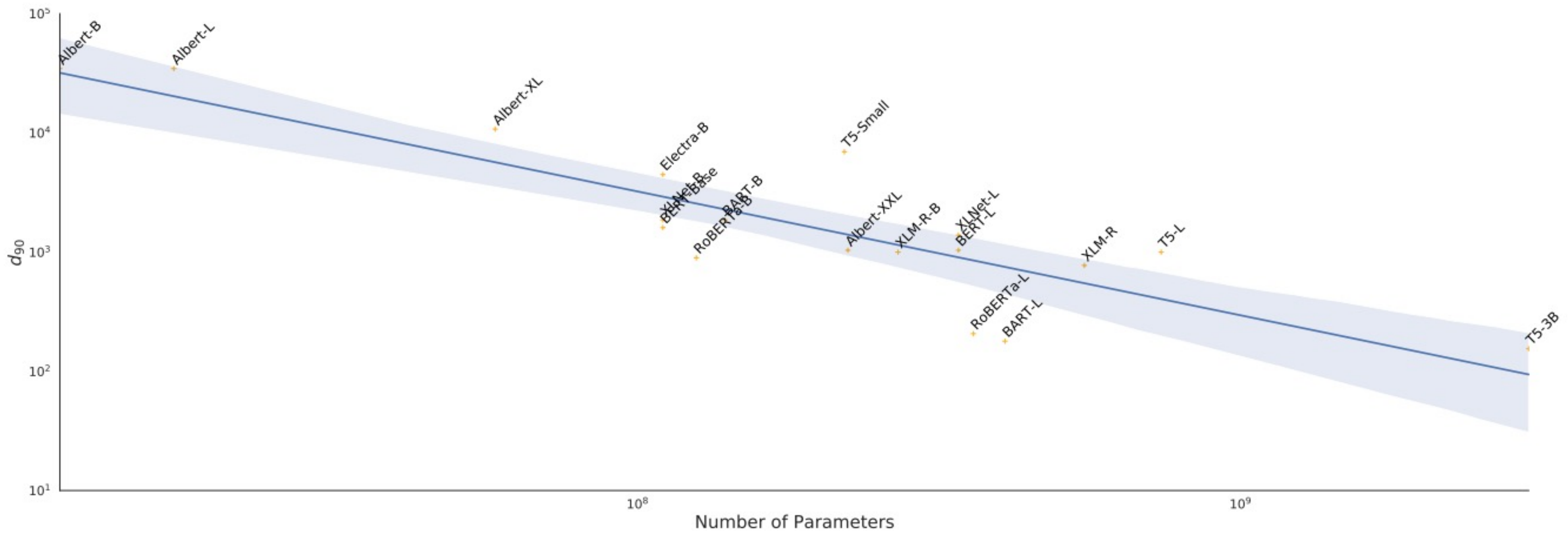


Aghajanyan, Armen, Sonal Gupta, and Luke Zettlemoyer. "Intrinsic Dimensionality Explains the Effectiveness of Language Model Fine-Tuning." *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. 2021.

Intrinsic Dimensionality

Larger models tend to have lower intrinsic dimension after a fixed number of pre-training updates

- Used the MRPC dataset and computed intrinsic dimension for every pre-trained model
- See a strong general trend that **as the number of parameters increases, the intrinsic dimension of fine-tuning on MRPC decreases**



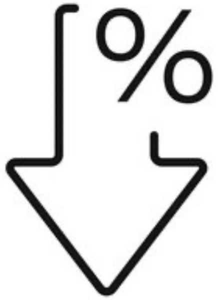
Aghajanyan, Armen, Sonal Gupta, and Luke Zettlemoyer. "Intrinsic Dimensionality Explains the Effectiveness of Language Model Fine-Tuning." *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. 2021.

Intrinsic Dimensionality

Observations

- Many problems have smaller intrinsic dimensions
- Intrinsic dimensionality decreases during pre-training
- Larger models have lower intrinsic dimensionality

- Parameters of model
- Pre-training updates

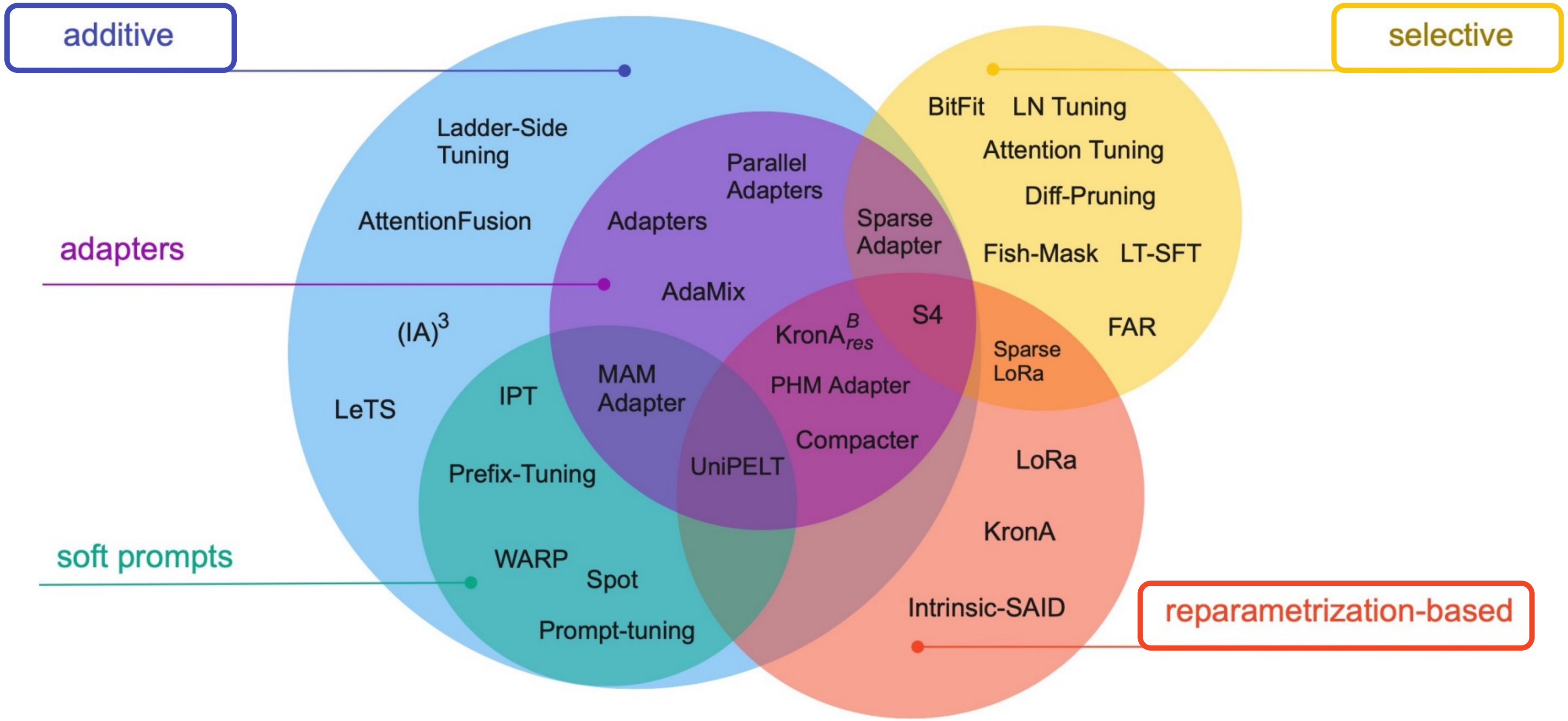


- Intrinsic dimension

Aghajanyan, Armen, Sonal Gupta, and Luke Zettlemoyer. "Intrinsic Dimensionality Explains the Effectiveness of Language Model Fine-Tuning." *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. 2021.

PEFT
Current Development & Method

PEFT Current Development

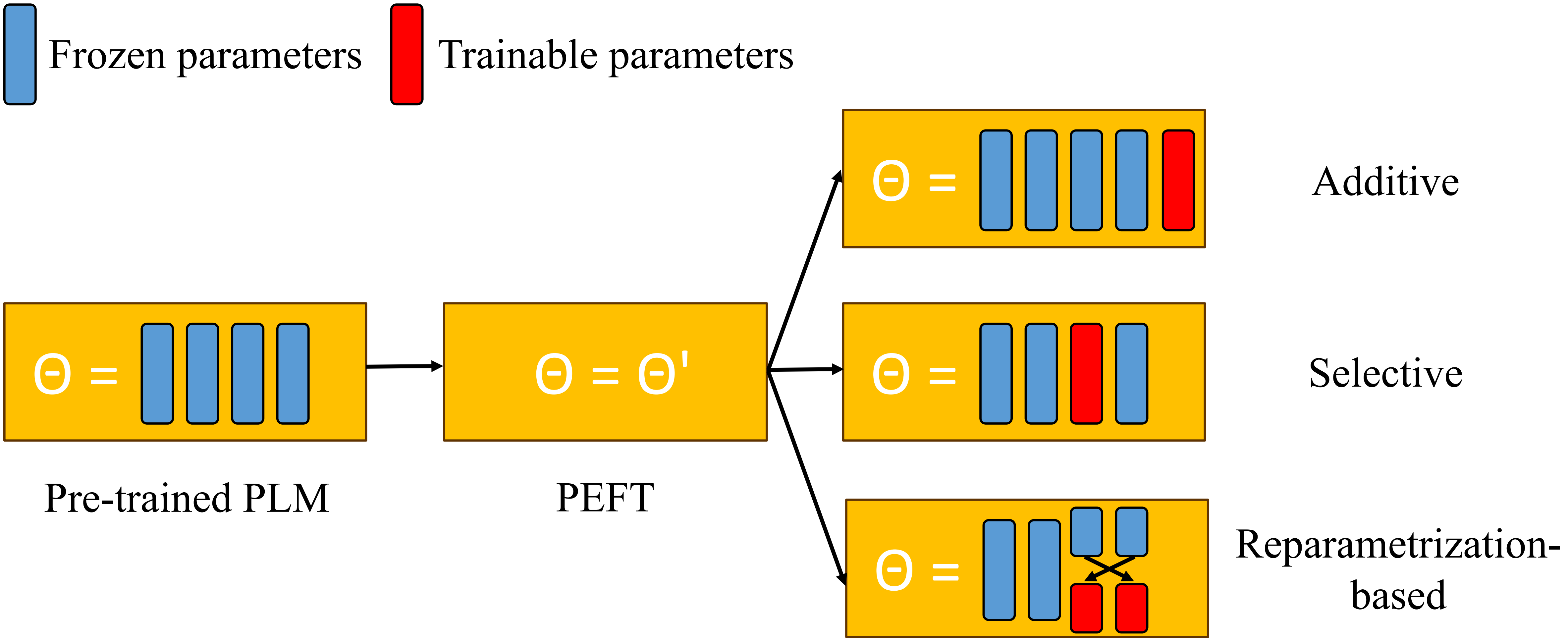


Lialin, Vladislav, Vijeta Deshpande, and Anna Rumshisky. "Scaling down to scale up: A guide to parameter-efficient fine-tuning." *arXiv preprint arXiv:2303.15647* (2023).

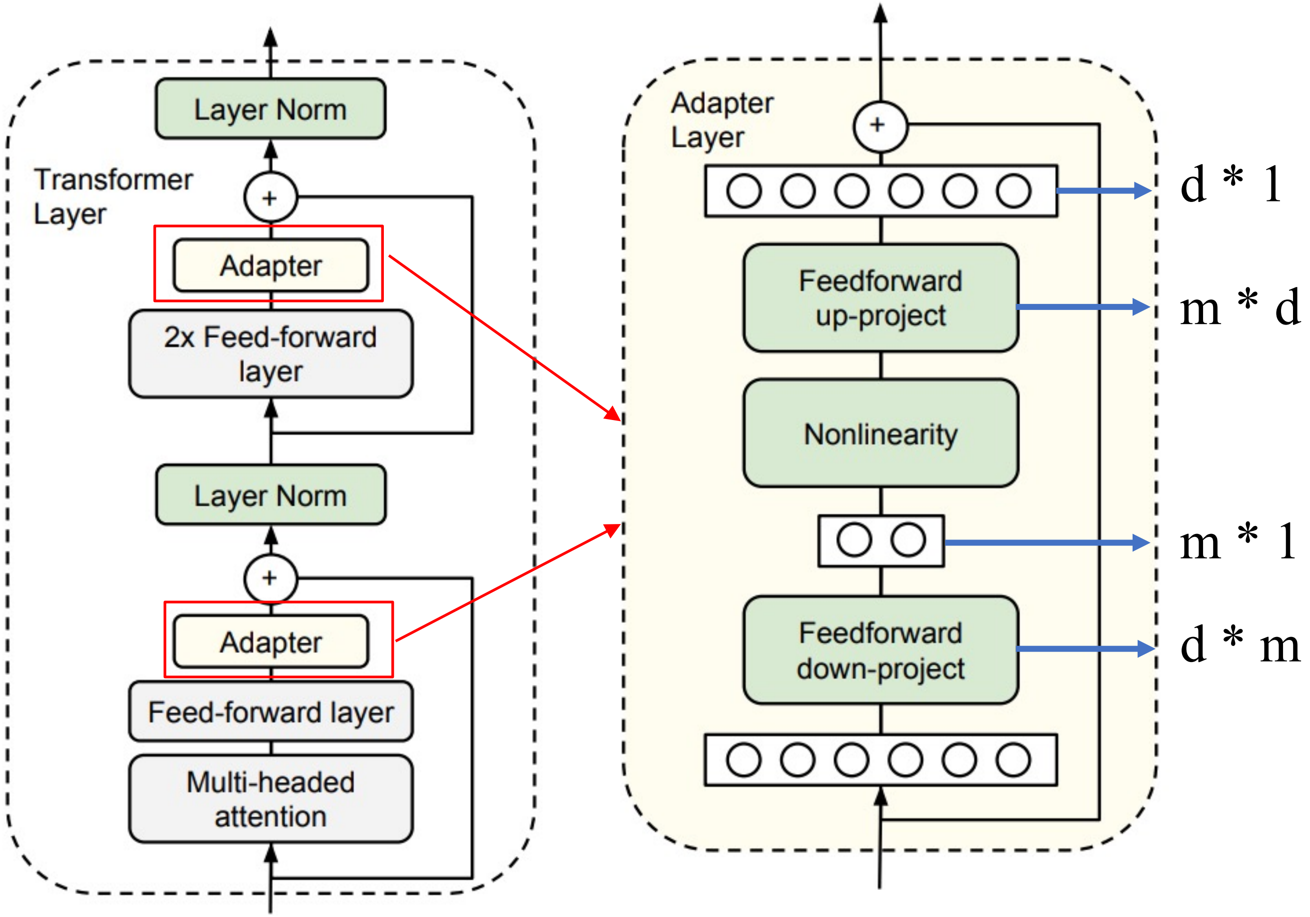
PEFT Current Development

- Additive Method:
  - Additive fine-tuning approaches involve introducing new extra trainable parameters for task-specific fine-tuning.
- Selective Method:
  - Selective fine-tuning methods aim to reduce the number of fine-tuned parameters by selecting a subset of pre-trained parameters that are critical to downstream tasks while discarding unimportant ones.
- Reparametrization-based Method:
  - Reparameterized fine-tuning methods utilize low-rank transformation to reduce the number of trainable parameters while allowing operating with high-dimensional matrices (e.g., pretrained weights).

PEFT Current Development



Additive PEFT: Adapters



- The adapters first project the original d-dimensional features into a smaller dimension, m, apply a nonlinearity, then project back to d dimensions.
- By setting m << d, we limit the number of parameters added per task

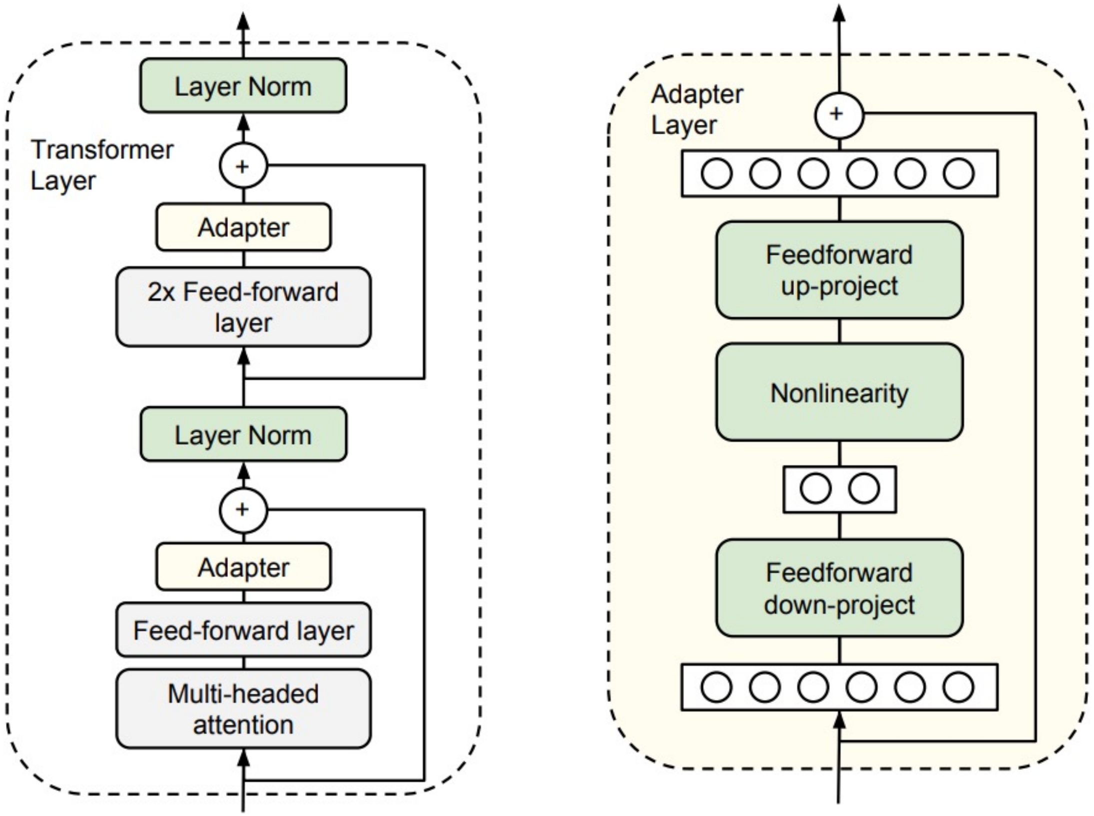
Houlsby, Neil, et al. "Parameter-efficient transfer learning for NLP." *International Conference on Machine Learning*. PMLR, 2019.

Additive PEFT: Adapters

Add fully-connected networks after attention and FFN layers

Pseudocode:

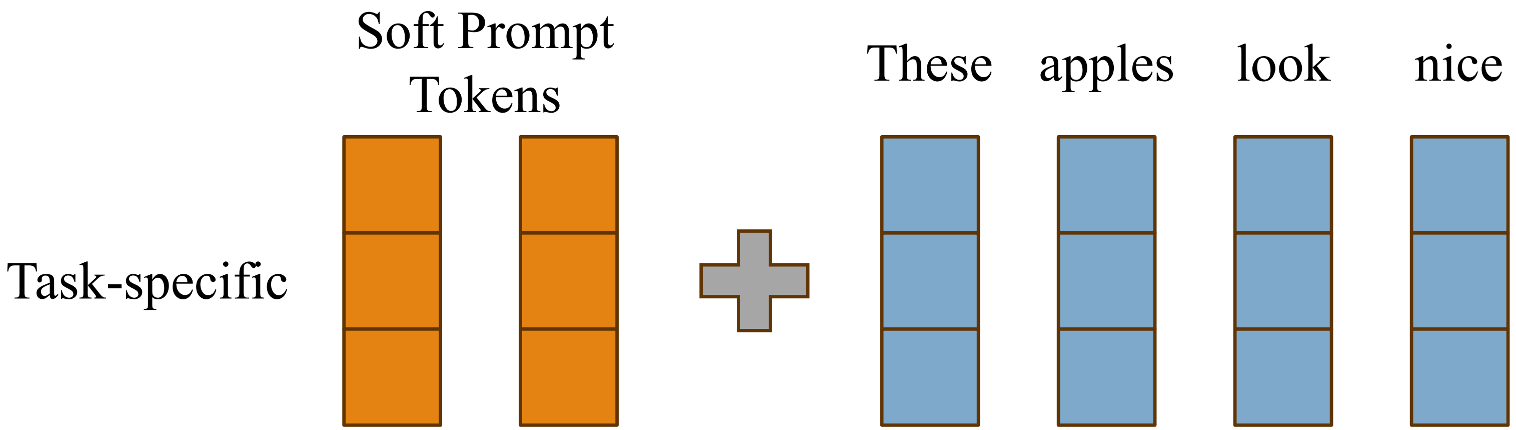
```
def transformer_block_with_adapter(x):
    residual = x
    x = SelfAttention(x)
    x = FFN(x)   # adapter
    x = LN(x + residual)
    residual = x
    x = FFN(x)   # transformer FFN
    x = FFN(x)   # adapter
    x = LN(x + residual)
    return x
```



Houlsby, Neil, et al. "Parameter-efficient transfer learning for NLP." *International Conference on Machine Learning*. PMLR, 2019.

Additive PEFT: Prompt Tuning

- Concatenates trainable parameters with the input embeddings
  - Learn a new sequence of task-specific embeddings
  - We call this prompt tuning because we only update prompt weights



Lester, Brian, Rami Al-Rfou, and Noah Constant. "The Power of Scale for Parameter-Efficient Prompt Tuning." *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. 2021.

Additive PEFT: Prompt Tuning

- Efficient for multi-task serving
    - Each task is a prompt, not a model
    - Prompts for various tasks can be applied to different inputs

	Soft Prompt Tokens		Input embedding vectors			
Sentiment	[.6,…,-4.3]	[.2,…,5.4]	These	apples	look	nice
Q&A	[-1.2,…,.8]	[1.3,…,-2.7]	When	should	I	leave
Translate	[-.5,…,-1.3]	[.9,…,-.5]	I	love	fresh	air

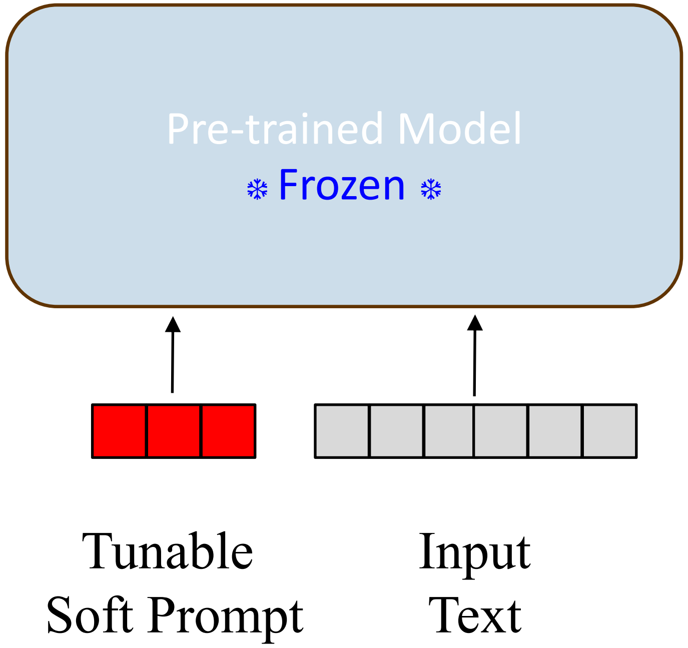
Lester, Brian, Rami Al-Rfou, and Noah Constant. "The Power of Scale for Parameter-Efficient Prompt Tuning." *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. 2021.

Additive PEFT: Prompt Tuning

Prepend the model input embeddings with a trainable tensor

Pseudocode:

```
# make it a parameter so that it can be trained
# Initialize soft_prompt tensor with random values
soft_prompt = torch.nn.parameter(
    torch.rand(num_tokens, embedding_dim)
)
# Concatenate soft_prompt with input x
def input_soft_prompt(x, soft_prompt):
    x = concatenate([soft_prompt, x],
    dim = seq_len)

    return x

# train soft_prompt tensor with gradient descent
train(model(input_soft_prompt(x, soft_prompt)))

# use model with soft_prompt
model(input_soft_prompt(x, soft_prompt))
```



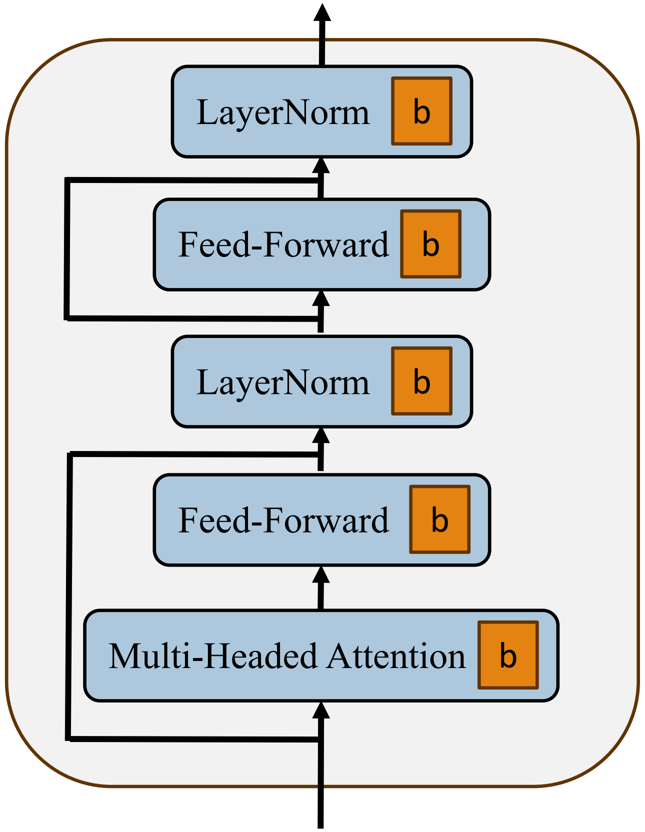
Lester, Brian, Rami Al-Rfou, and Noah Constant. "The Power of Scale for Parameter-Efficient Prompt Tuning." *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. 2021.

Selective PEFT: BitFit

Fine-tune only model biases

Pseudocode:

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



Zaken, Elad Ben, Yoav Goldberg, and Shauli Ravfogel. "BitFit: Simple Parameter-efficient Fine-tuning for Transformer-based Masked Language-models." *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. 2022.

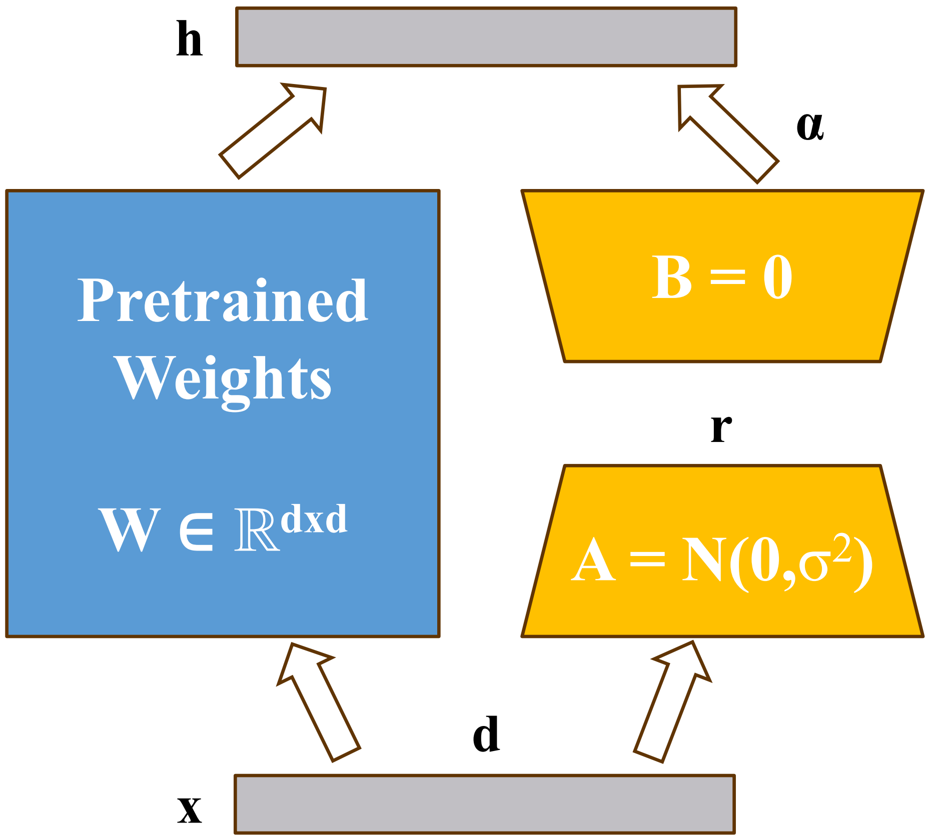
Reparametrization-based PEFT: LoRA

$$h = W_0 x + \alpha \Delta W x$$
$$= W_0 x + \alpha B A x$$

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

Matrix rank Hidden dimension Input dimension

$\alpha \in \mathbb{R}^{+}$ Constant



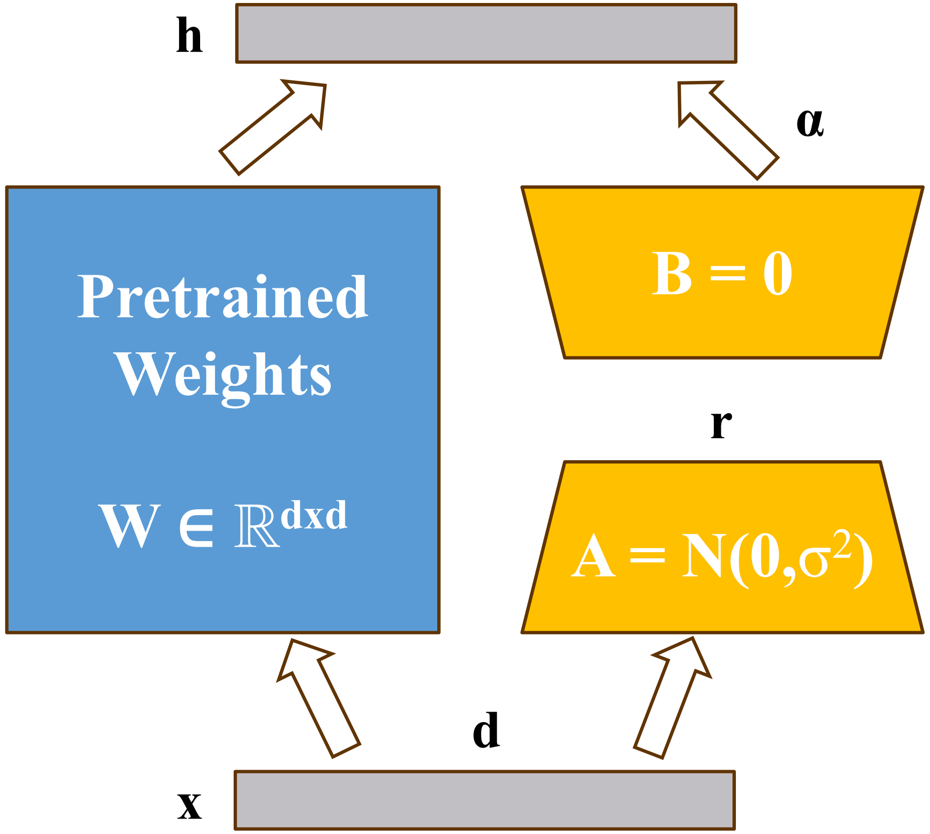
Hu, Edward J., et al. "LoRA: Low-Rank Adaptation of Large Language Models." *International Conference on Learning Representations*. 2021.

Reparametrization-based PEFT: LoRA

Decompose a weight matrix into lower-rank matrices

Pseudocode:

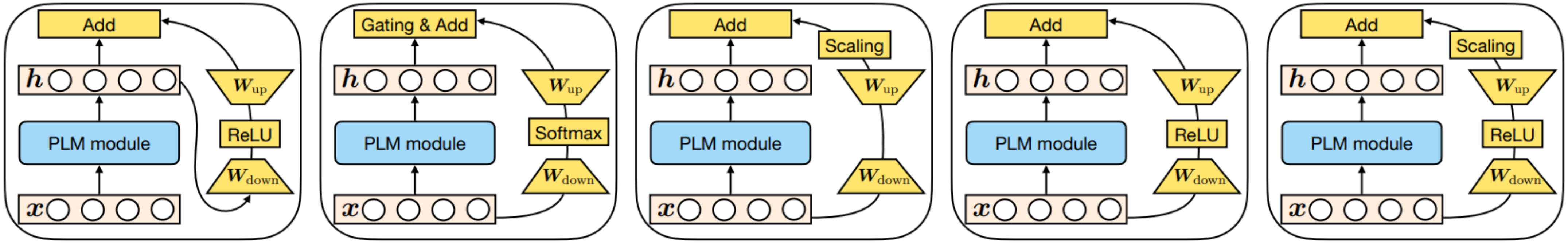
```
input_dim = 768  # the hidden size of the pre-trained model
output_dim = 768  # the output size of the layer
rank = 8  # The rank 'r' for the low-rank adaptation

W = ... # from pretrained network with shape input_dim x
output_dim

W_A = nn.Parameter(torch.empty(input_dim, rank)) # LoRA weight A
W_B = nn.Parameter(torch.empty(rank, output_dim)) # LoRA weight B

# Initialization of LoRA weights
nn.init.kaiming_uniform_(W_A, a=math.sqrt(5))
nn.init.zeros_(W_B)

def regular_forward_matmul(x, W):
    h = x @ W
return h

def lora_forward_matmul(x, W, W_A, W_B):
    h = x @ W  # regular matrix multiplication
    h += x @ (W_A @ W_B) * alpha # use scaled LoRA weights
return h
```



Hu, Edward J., et al. "LoRA: Low-Rank Adaptation of Large Language Models." *International Conference on Learning Representations*. 2021.

Hybrid PEFT: MAM Adapters

- Break down the design of state-of-the-art parameter-efficient transfer learning methods and present a unified framework that establishes connections between them.



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

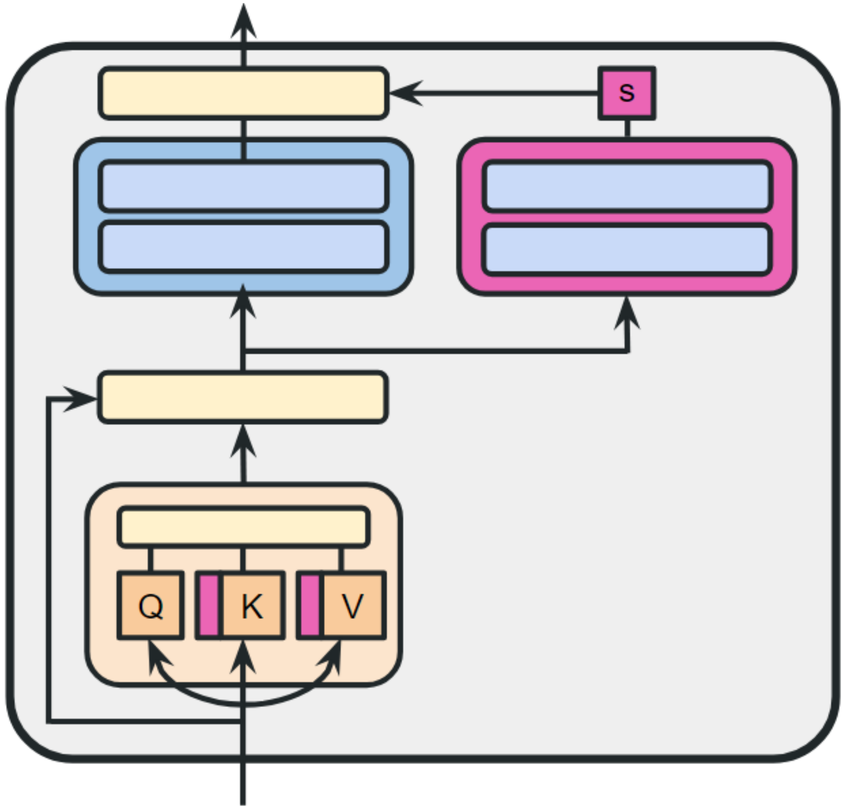
He, Junxian, et al. "Towards a Unified View of Parameter-Efficient Transfer Learning." *International Conference on Learning Representations*. 2021.

Hybrid PEFT: MAM Adapters

Scaled parallel adapter for FFN layer + soft prompt

Pseudocode:

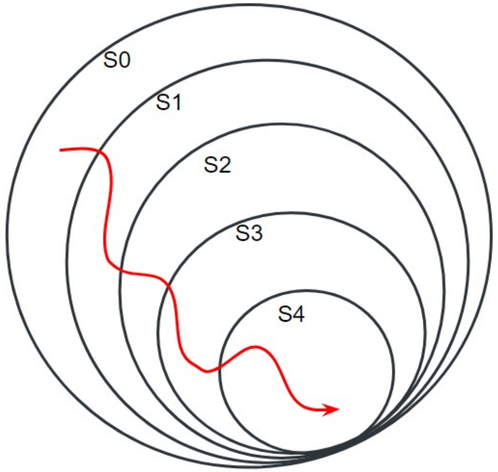
```
def transformer_block_mam(x):
    x =  concat([x,soft_prompt],
                dim=seq)
    residual = x
    x = SelfAttention(x)
    x = LN(x + residual)
    x_a = FFN(x) # parallel adapter
    x_a = scale * x_a
    x = LN(x + x_adapter)
    return x
```

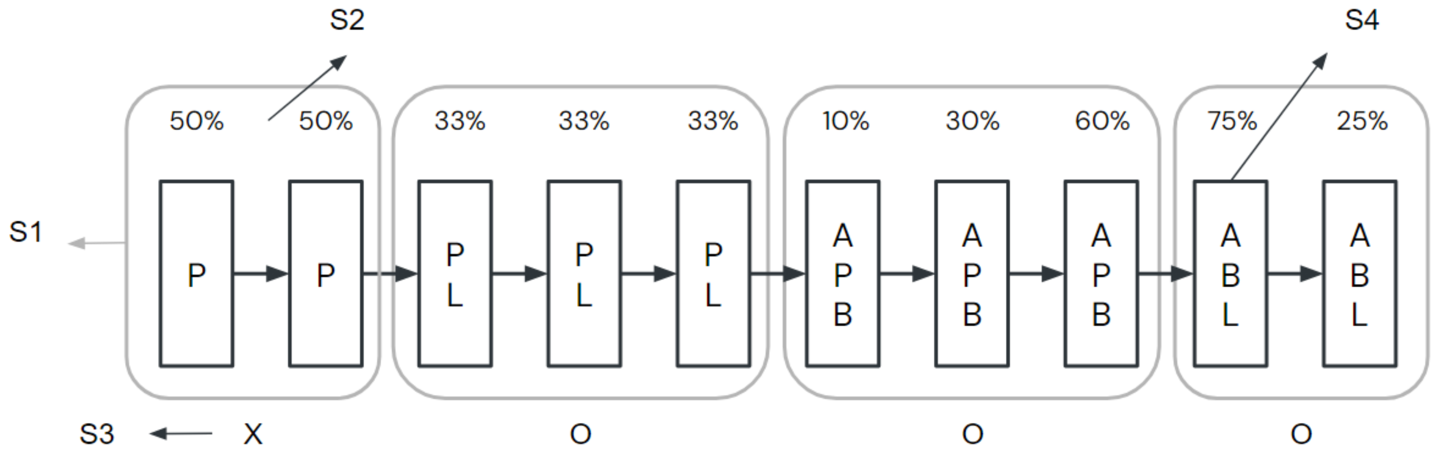


He, Junxian, et al. "Towards a Unified View of Parameter-Efficient Transfer Learning." *International Conference on Learning Representations*. 2021.

Hybrid PEFT: S4

- Designing Network Design Spaces
  - S0_The Initial Design Space: a random strategy
  - S1_Layer Grouping: Increasing, Uniform, Decreasing, Spindle, Bottleneck
  - S2_Trainable Parameter Allocation: Increasing, Uniform, Decreasing
  - S3_Tunable Groups: Tune or not
  - S4_Strategy Assignment: {Adapter (A), Prefix (P), BitFit (B), and LoRA (L)}





Chen, Jiaao, et al. "Parameter-Efficient Fine-Tuning Design Spaces." *The Eleventh International Conference on Learning Representations*. 2022.

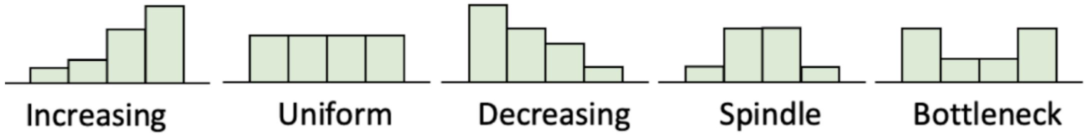
Hybrid PEFT: S4

Automatically found combination of Adapters, Prefix-Tuning, BitFit, and LoRA

Search process:

At 0.1% additional parameters

1. Find optimal grouping pattern: spindle
2. Find optimal parameter allocation pattern: uniform
3. Find which groups need tuning: all
4. find optimal combinations of PEFT techniques sequentially for G1, G2, G3, G4



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

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

Chen, Jiaao, et al. "Parameter-Efficient Fine-Tuning Design Spaces." *The Eleventh International Conference on Learning Representations*. 2022.

PEFT Method: Summary

Method	Type	Storge	Memory	Infernce overhead	Trainable parameters	Changed parameters
Adapters	A	yes	yes	Extra FFN	0.1% - 6%	0.1% - 6%
Prompt Tuning	A	yes	yes	Extra input	0.1%	0.1%
Bitfit	S	yes	yes	Extra input	0.5%	0.5%
LoRA	R	yes	yes	No overhead	0.01% - 0.5%	0.5% - 30%
MAM Adapters	A	yes	yes	Extra FFN & input	0.5%	0.5%
S4	ASR	yes	yes	Extra FFN & input	0.5%	> 0.5%

Lialin, Vladislav, Vijeta Deshpande, and Anna Rumshisky. "Scaling down to scale up: A guide to parameter-efficient fine-tuning." *arXiv preprint arXiv:2303.15647* (2023).

What Matters in PEFT ?

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

Appendix

Formula for Hidden States Estimate

During training:

$(3\ h\ seq + L(4\ h\ seq + 3\ h\ seq^2 + 8\ h\ seq + 2\ h\ seq + 4\ h\ seq) + vocab\ seq)\ bs =$

Emb,pos, Pre-logit h	K,V,Q,O	Score, Probs, Dropout	FFN hidden, activation	FFN out, dropout	residual, LN	logits

$= 3\ h\ seq\ bs + 18L\ h\ seq\ bs + 3L\ heads\ seq2 + vocab\ seq\ \ bs$

L : Number of layers in model (eq. 32 layers)
H : Number of attention heads (eq. 32 heads)
bs : batch size