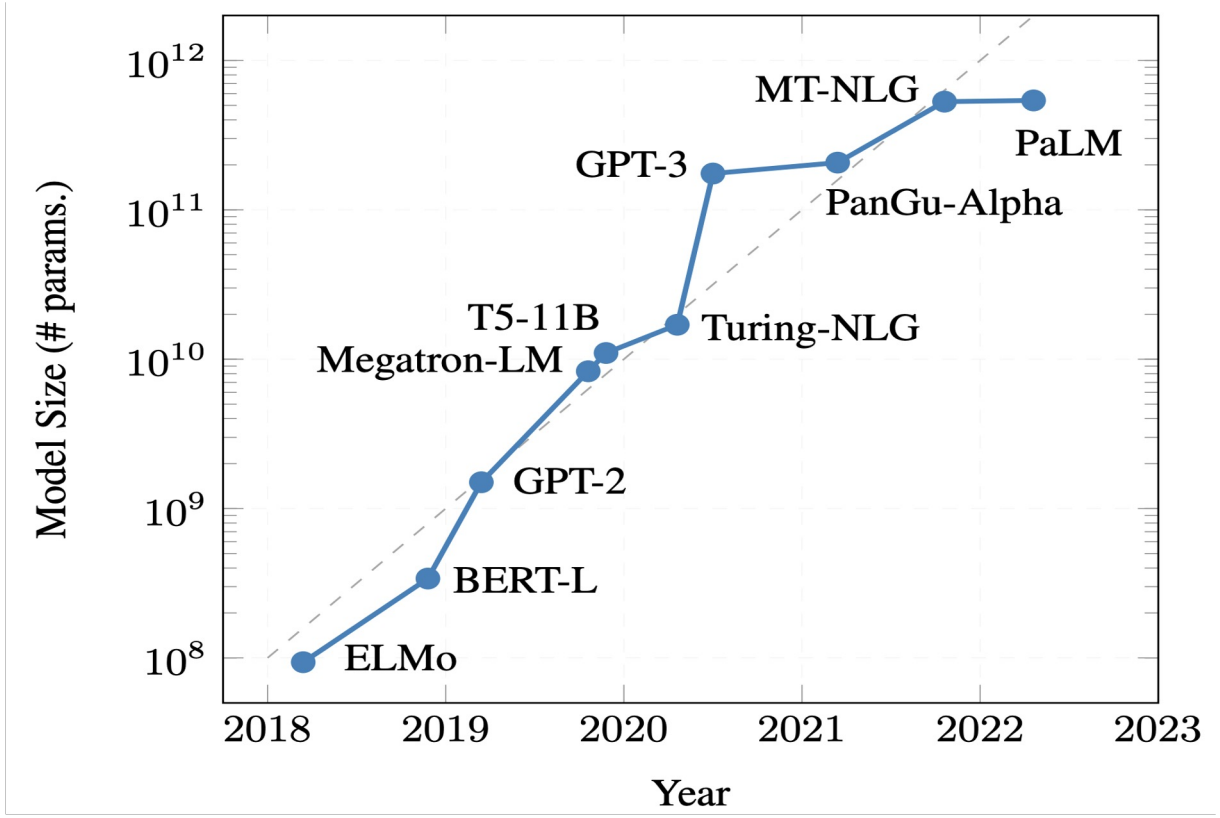
PEFT

Harpreet Singh

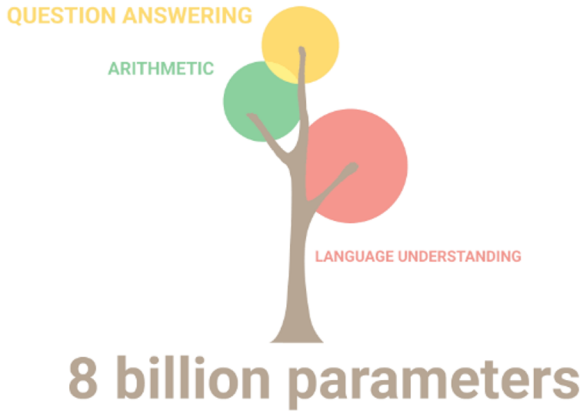
Fall 2023

State-of-the-art NLP Models Are Getting Ever Larger



Evolution of the size of large pre-trained models [Treviso et al., 2022]

Large Models Develop Increasing NLU Capabilities



- Slides: https://tinyurl.com/modular-fine-tuning-tutorial

Credit: Google AI Blog

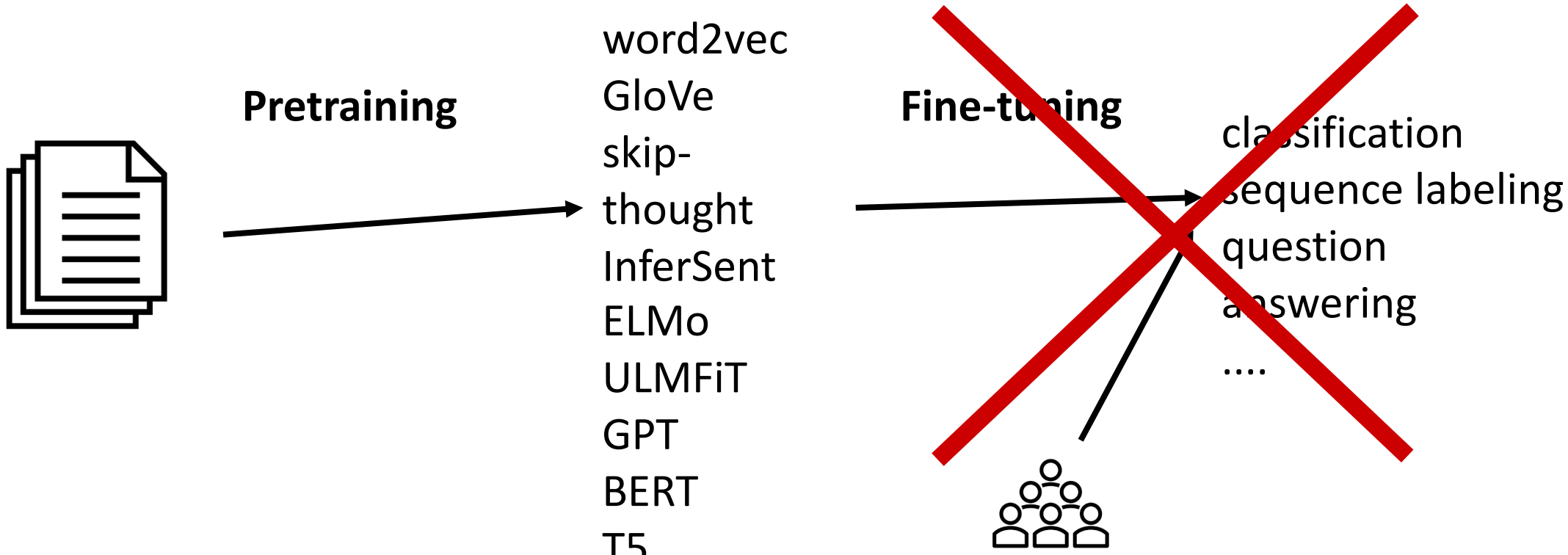
GPU Memory breakdown

		OPT-1.3B, 16-bit float, seq 512
cuDNN and CUDA		~1Gb
Model weights	size(float) * N	2.6Gb
Gradients	size(float) * $N_{trainable}$	2.6Gb
Hidden states	*~size(float) L (20 h seq + 3 seq^2)*	1Gb per example
Optimizer states	2 * size(float) * $N_{trainable}$	5.2Gb
(maybe) fp32 copy of the gradients	4 * $N_{trainable}$	10.2Gb

Slide Source: https://docs.google.com/presentation/d/1wY3pcS-rnxKGQTSF9cCTft4xd9l5B-4diJ6xb2EfBOI/edit#slide=id.p

Transfer Learning in the Era of Large Models

- With increasing model size, fine-tuning becomes increasingly expensive
- The standard transfer learning formula breaks down

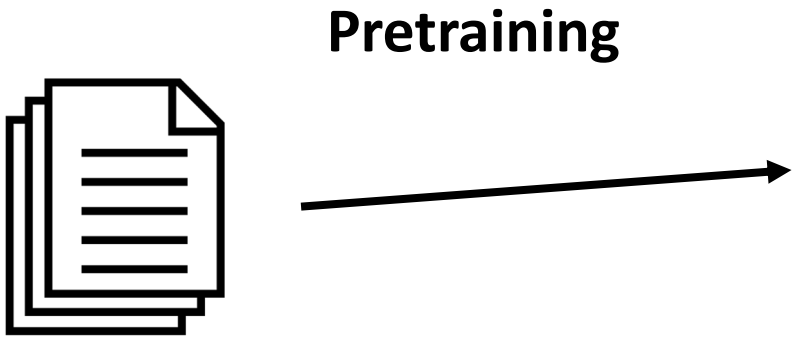


- Slides: https://tinyurl.com/modular-fine-tuning-tutorial

Credit: NAACL 2019 Transfer learning tutorial

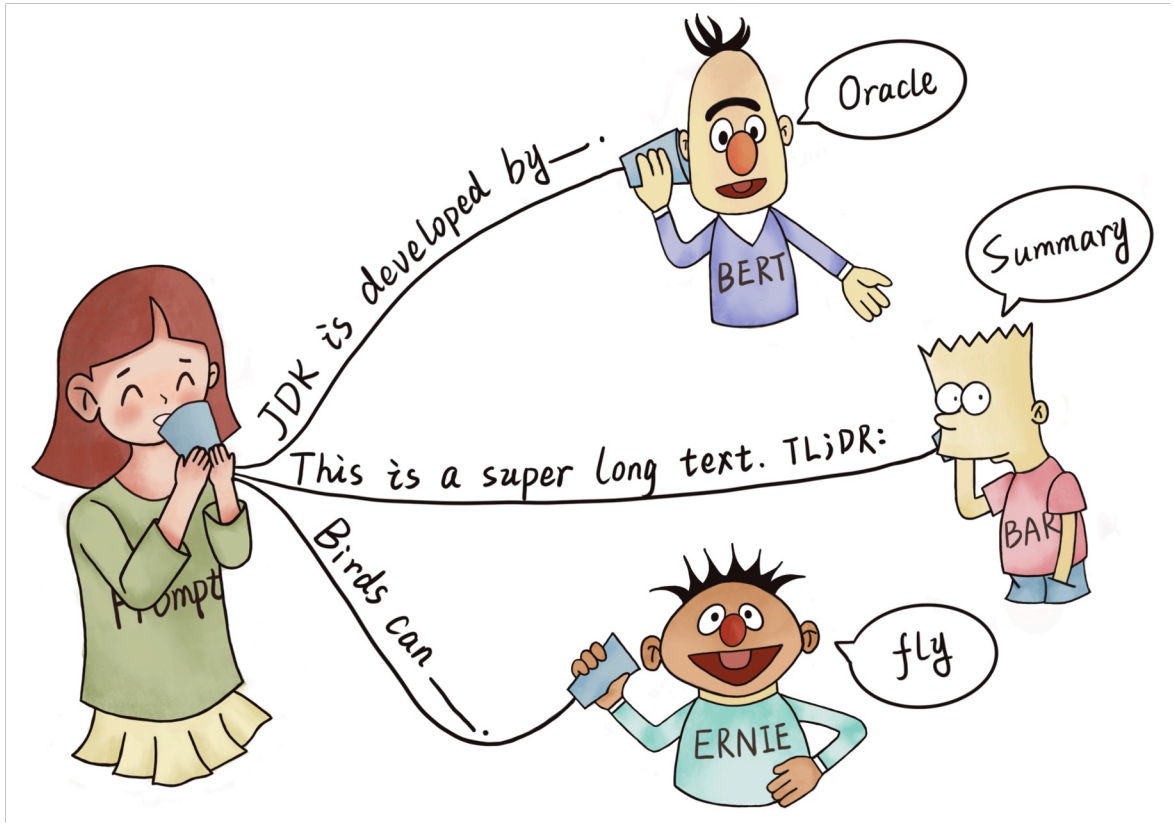
Transfer Learning in the Era of Large Models

- In-context learning has mostly replaced fine-tuning for large models



Pretraining

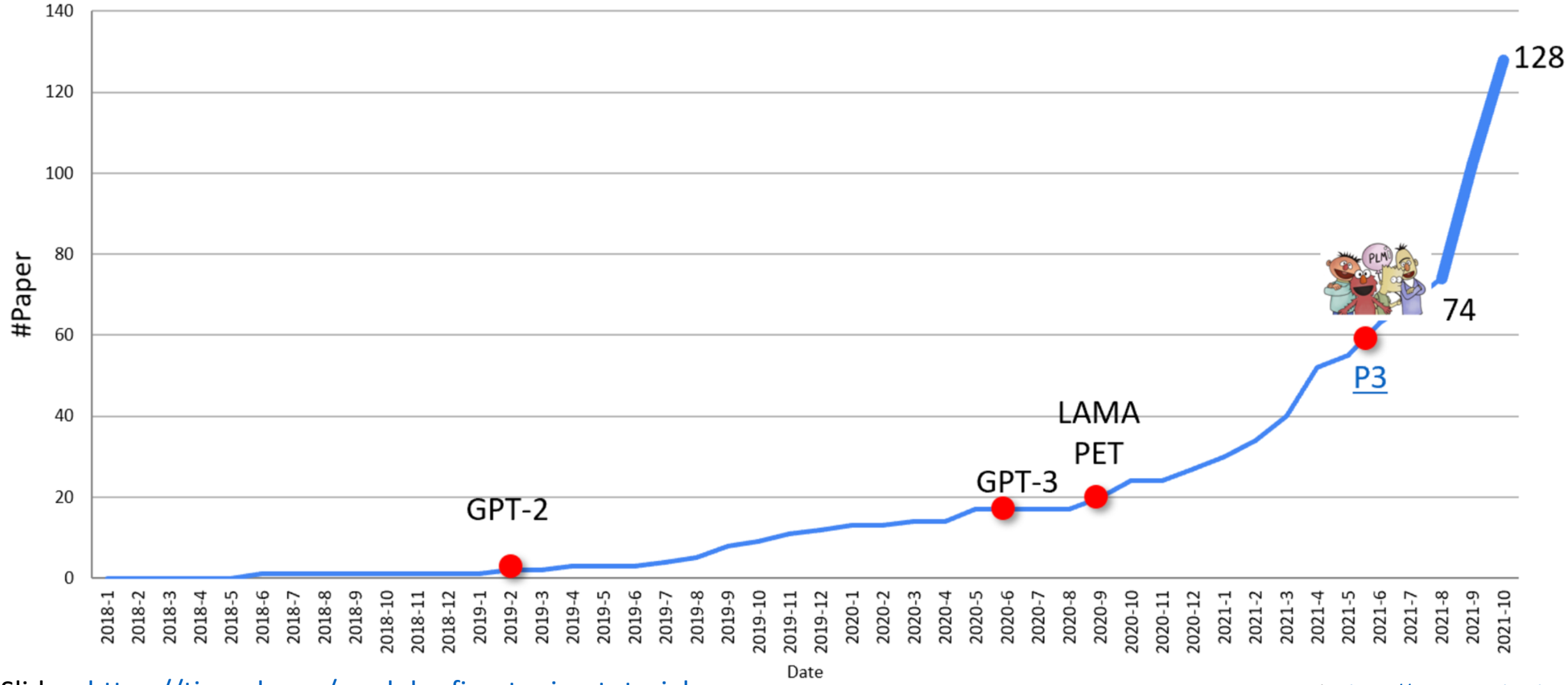
BERT
BART
ERNIE
GPT-3
PaLM



Slides: https://tinyurl.com/modular-fine-tuning-tutorial

Credit: Liu et al. (2021)

Prompt-based Learning has Taken NLP by Storm



Slides: https://tinyurl.com/modular-fine-tuning-tutorial

Credit: http://pretrain.nlpedia.ai/

Downsides of Prompt-based Learning

1. **Inefficiency:** The prompt needs to be processed *every time* the model makes a prediction.

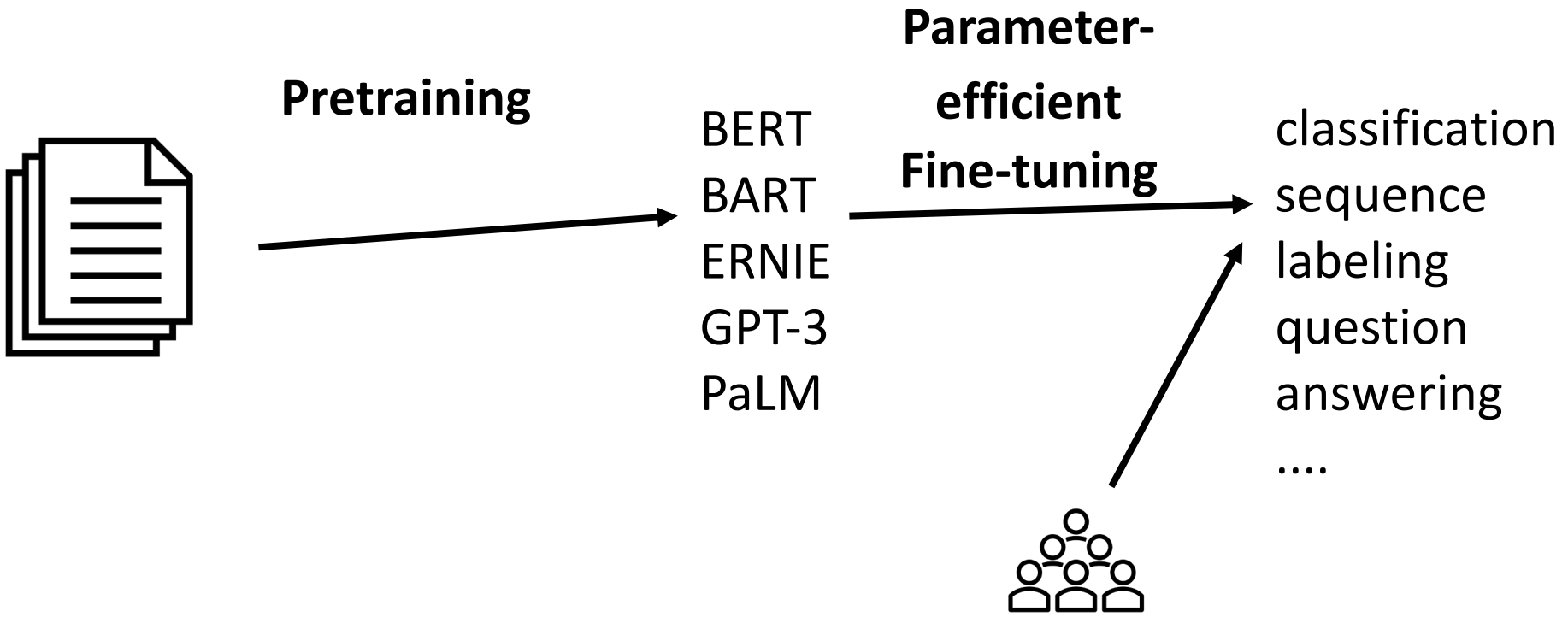
2. **Poor performance**: Prompting generally performs worse than fine-tuning [Brown et al., 2020].

3. **Sensitivity** to the wording of the prompt [Webson & Pavlick, 2022], order of examples [Zhao et al., 2021; Lu et al., 2022], etc.

4. **Lack of clarity** regarding what the model learns from the prompt. Even random labels work [Min et al., 2022]!

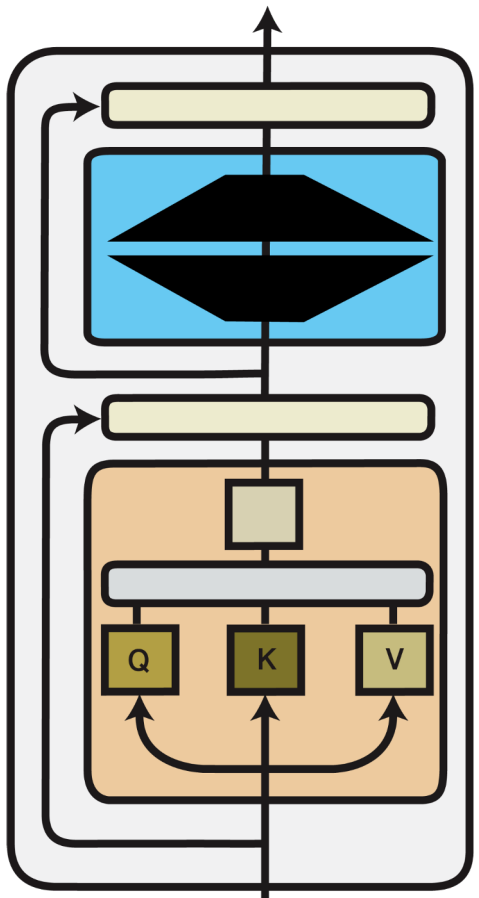
Slides: https://tinyurl.com/modular-fine-tuning-tutorial

From Fine-tuning to Parameter-efficient Fine-tuning



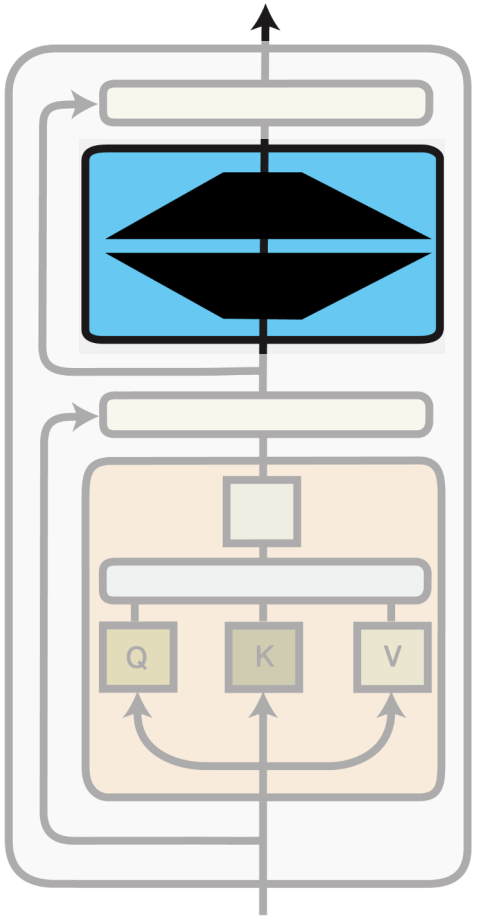
Slides: https://tinyurl.com/modular-fine-tuning-tutorial

From Fine-tuning to Parameter-efficient Fine-tuning



Full Fine-tuning

Update **all model parameters**

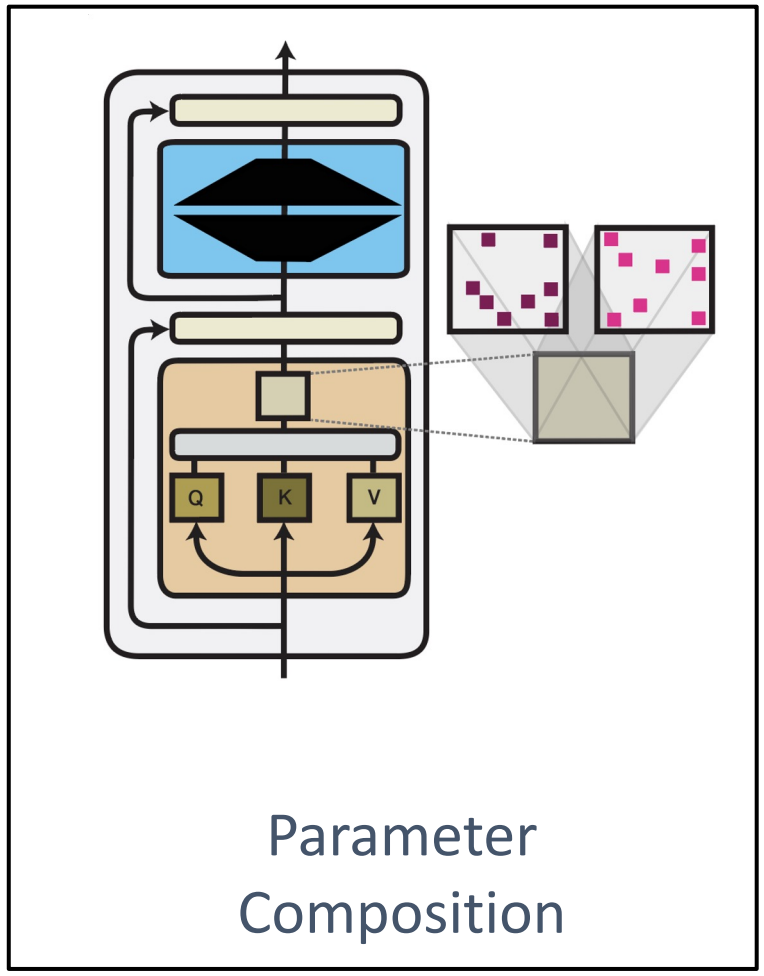


Parameter-efficient Fine-tuning

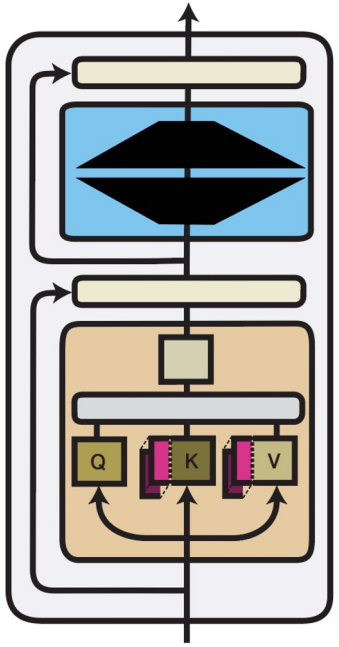
Update a **small subset** of model parameters

Slides: https://tinyurl.com/modular-fine-tuning-tutorial

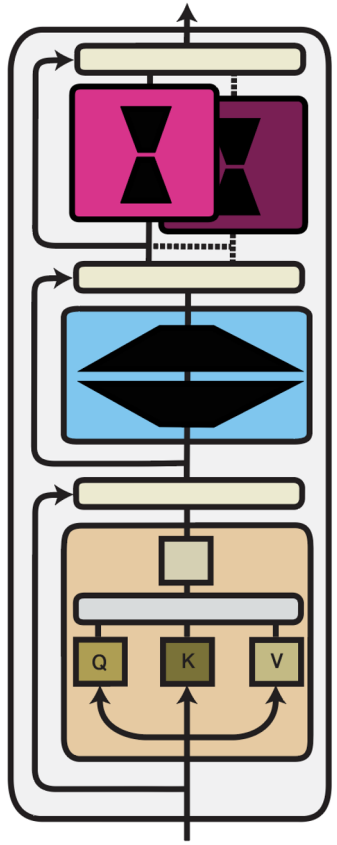
Three Computation Functions



Parameter Composition



Input Composition



Function Composition

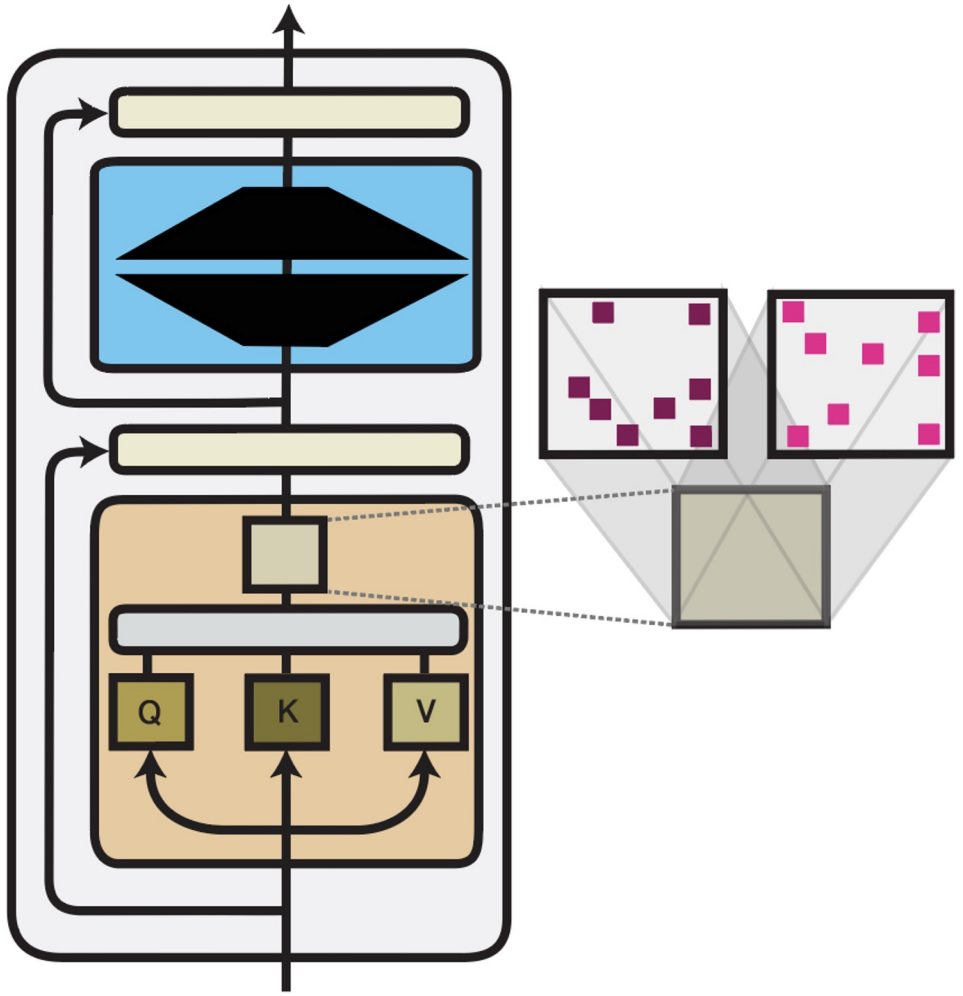
Slides: https://tinyurl.com/modular-fine-tuning-tutorial

Parameter Composition

1. Sparse Subnetworks

2. Structured Composition

3. Low-rank Composition

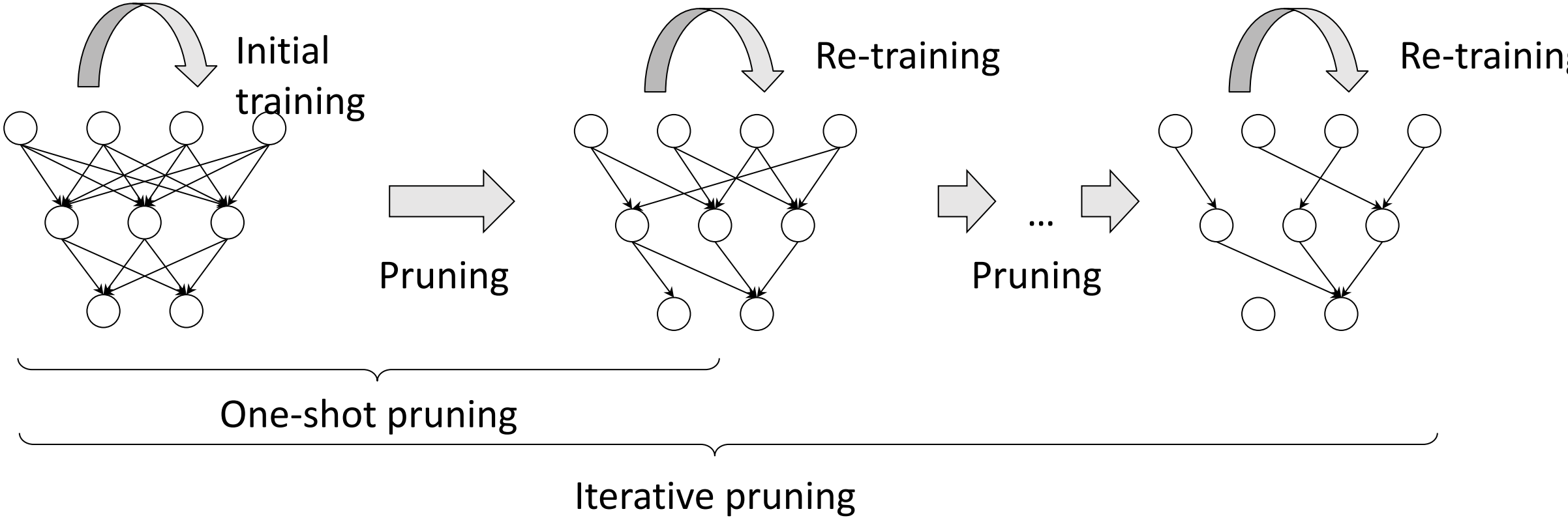


Slides: https://tinyurl.com/modular-fine-tuning-tutorial

Sparse Subnetworks

- A common inductive bias on the module parameters is **sparsity**
- Most common sparsity method: **pruning**
- Pruning can be seen as applying a binary mask that selectively keeps or removes each connection in a model and produces a subnetwork.
- Most common pruning criterion: **weight magnitude** [Han et al., 2017]

Pruning



- During pruning, a fraction of the lowest-magnitude weights are removed
- The non-pruned weights are re-trained
- Pruning for multiple iterations is more common [Frankle & Carbin, 2019]

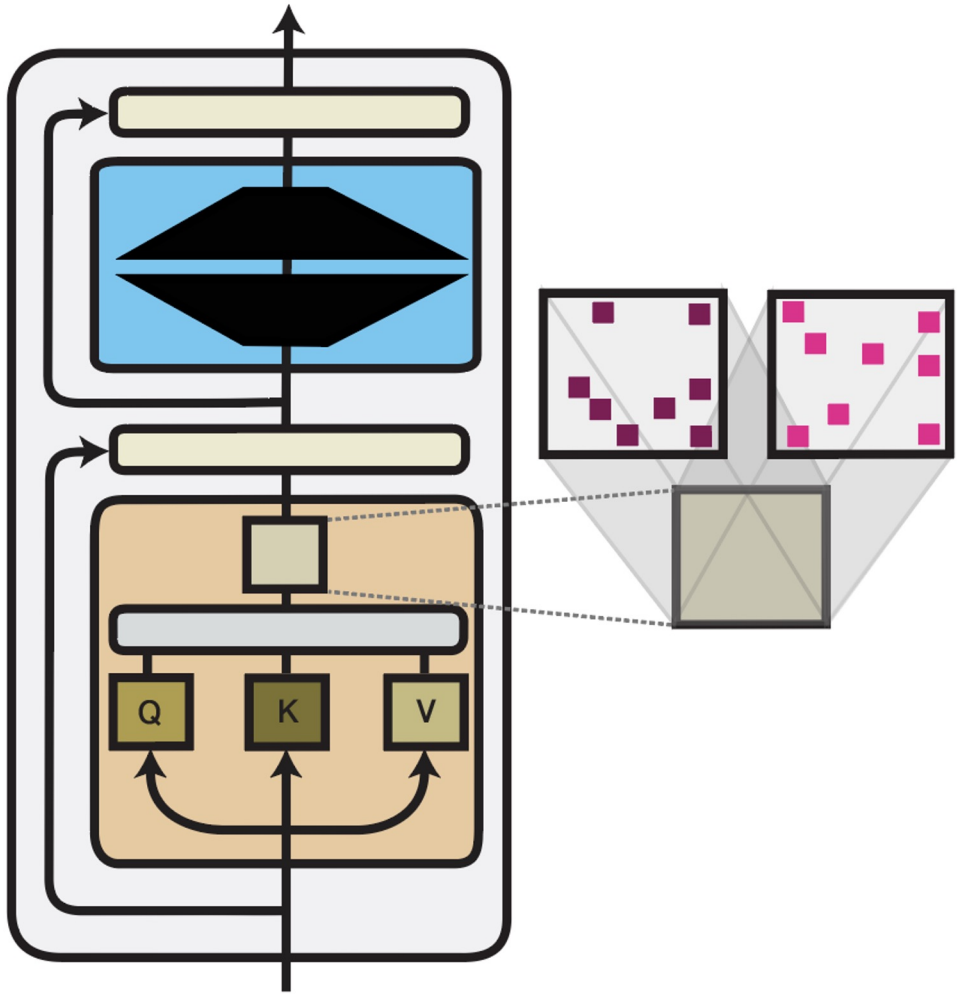
Slides: https://tinyurl.com/modular-fine-tuning-tutorial

Parameter Composition

1. Sparse Subnetworks

2. Structured Composition

3. Low-rank Composition



Slides: https://tinyurl.com/modular-fine-tuning-tutorial

Structured Composition

- We can additionally impose a structure on the weights that we select
- Specifically, we can only modify the weights that are associated with a pre-defined group $\mathcal{G}$: $f_i' = f_{\theta_i + \phi_i} \; \forall f_i' \in \mathcal{G}$

Slides: https://tinyurl.com/modular-fine-tuning-tutorial

Group-based Fine-tuning

- Most common setting: each group $\mathcal{G}$ corresponds to a layer; only update the parameters associated with certain layers

  - Groups can also relate to more fine-grained components

Slides: https://tinyurl.com/modular-fine-tuning-tutorial

Parameter Composition

One-sentence idea:
Fine-tune only model biases

Pseudocode:

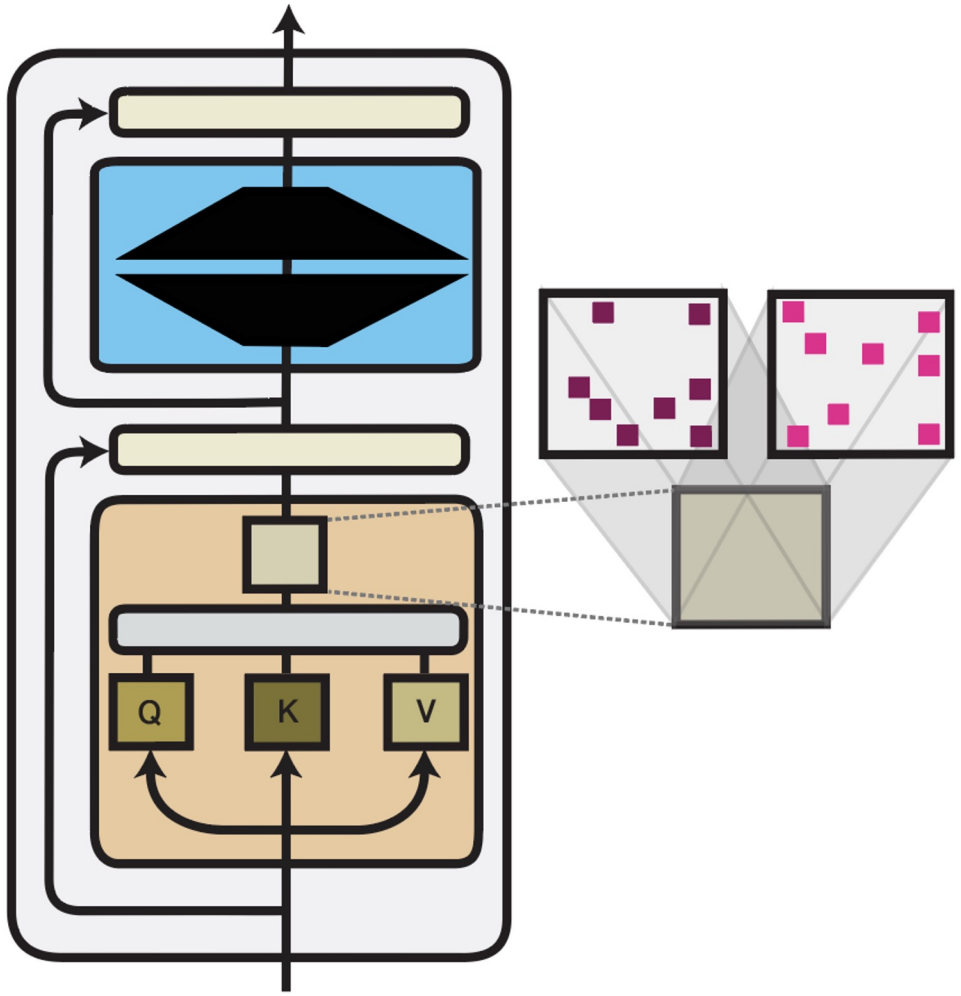
```
params = (p for n, p
          in model.named_parameters()
          if "bias" in n)
optimizer = Optimizer(params)
```

Slide Source: https://docs.google.com/presentation/d/1wY3pcS-rnxKGQTSF9cCTft4xd9l5B-4diJ6xb2EfBOI/edit#slide=id.p

https://arxiv.org/abs/2106.10199

Parameter Composition

1. Sparse Subnetworks
2. Structured Composition
3. Low-rank Composition



Slides: https://tinyurl.com/modular-fine-tuning-tutorial

Low-rank Composition

- Another useful inductive bias: module parameters ϕ should lie in a low-dimensional space

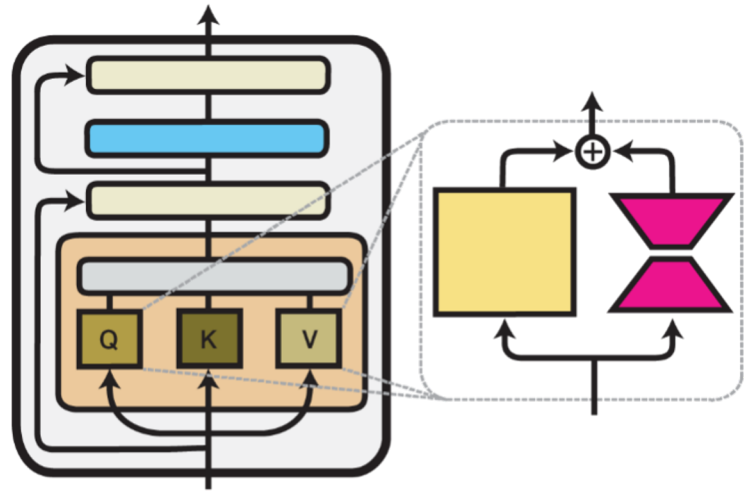
- Li et al. [2018] show that models can be optimized in a low-dimensional, randomly oriented subspace rather than the full parameter space

Low-rank Adaptation (LoRA)

One-sentence idea:
Parameter update for a weight matrix in LoRA is decomposed into a product of two low-rank matrices

Pseudocode:

```
def lora_linear(x):
    h = x @ W # regular linear
    h += x @ W_A @ W_B # low-rank update
    return scale * h
```

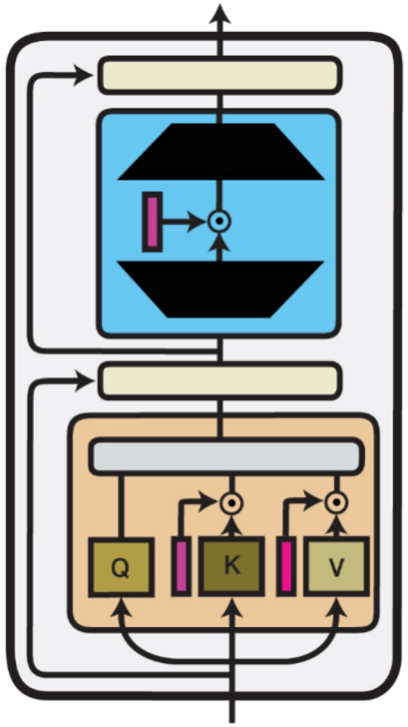


Slide Source: https://docs.google.com/presentation/d/1wY3pcS-rnxKGQTSF9cCTft4xd9l5B-4diJ6xb2EfBOI/edit#slide=id.p

https://arxiv.org/abs/2106.09685
Image source: adapterhub.ml

(IA)3 Infused Adapter by Inhibiting and Amplifying Inner Activations

One-sentence idea:
Rescale key, value, and hidden FFN activations



Pseudocode:

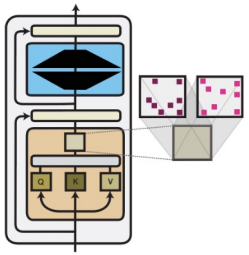
```
def transformer_block_with_ia3(x):
    residual = x
    x = ia3_self_attention(x)
    x = LN(x + residual)
    residual = x
    x = x @ W_1           # FFN in
    x = l_ff * gelu(x)    # (IA)3 scaling
    x = x @ W_2           # FFN out
    x = LN(x + residual)
    return x

def ia3_self_attention(x):
    k, q, v = x @ W_k, x @ W_q, x @ W_v
    k = l_k * k
    v = l_v * v
    return softmax(q @ k.T) @ V
```

Slide Source: https://docs.google.com/presentation/d/1wY3pcS-rnxKGQTSF9cCTft4xd9l5B-4diJ6xb2EfBOI/edit#slide=id.p

https://arxiv.org/abs/2205.05638

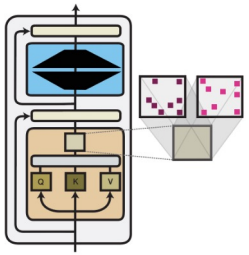
Image source: adapterhub.ml

Computation Functions Comparison

		Parameter efficiency	Training efficiency	Inference efficiency	Performance	Composition-ality
Parameter composition		Methods such as diff pruning require < 0.5% of parameters	Pruning requires re-training iterations	Does not increase the model size	E.g., LoRA achieves strong performance	Subnetworks can be composed

Slides: https://tinyurl.com/modular-fine-tuning-tutorial

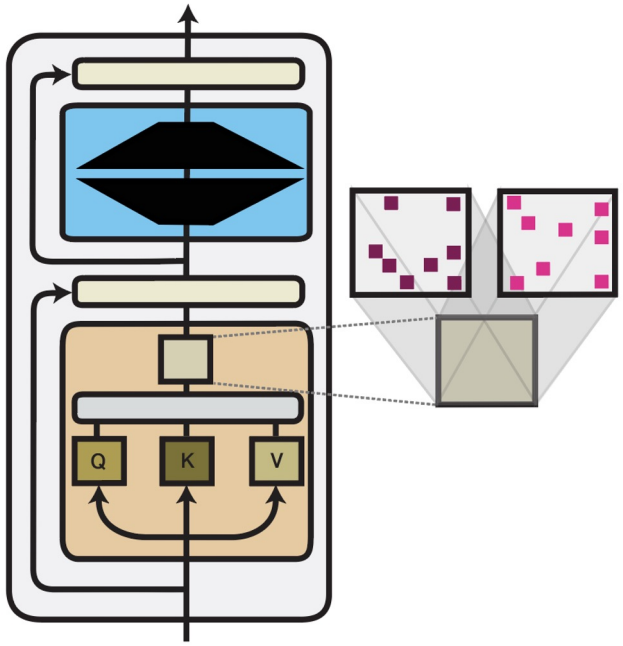
Computation Functions Comparison

		Parameter efficiency	Training efficiency	Inference efficiency	Performance	Composition-ality
Parameter composition		+	-	++	+	+

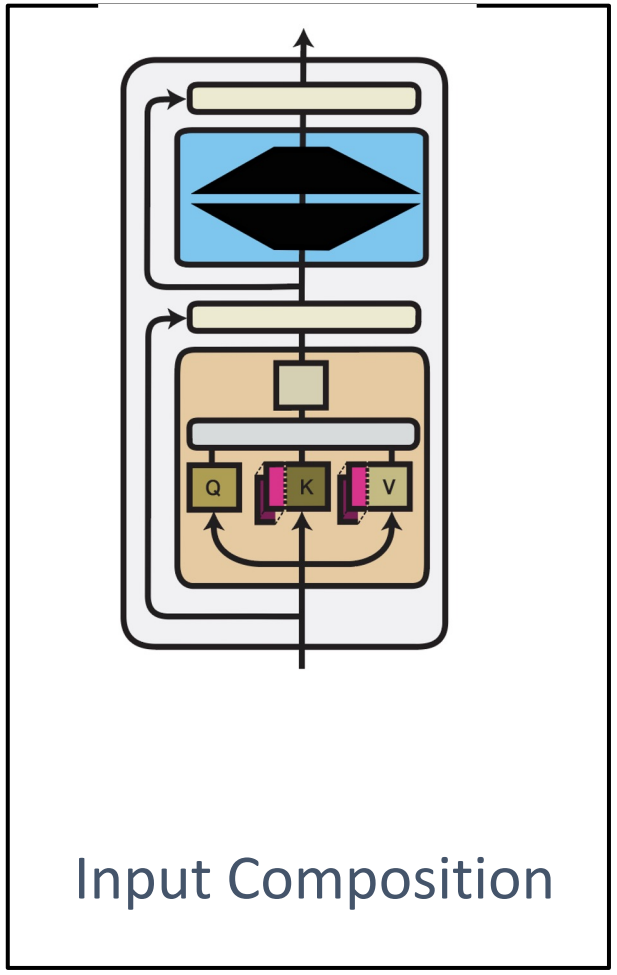
This is mainly meant as high-level guidelines. Individual methods may have different trade-offs and mitigate certain weaknesses.

Slides: https://tinyurl.com/modular-fine-tuning-tutorial

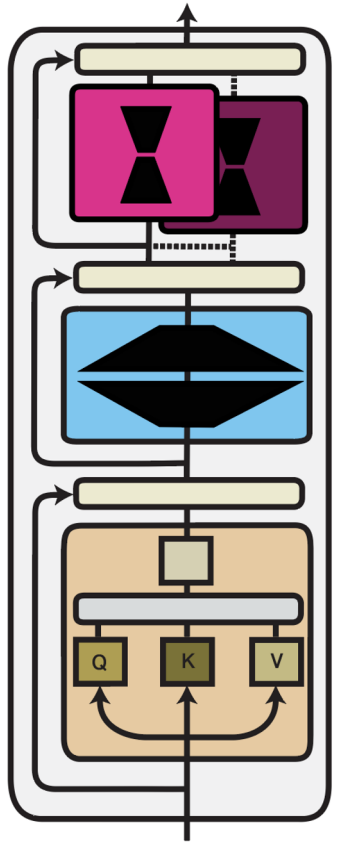
Three Computation Functions



Parameter Composition



Input Composition



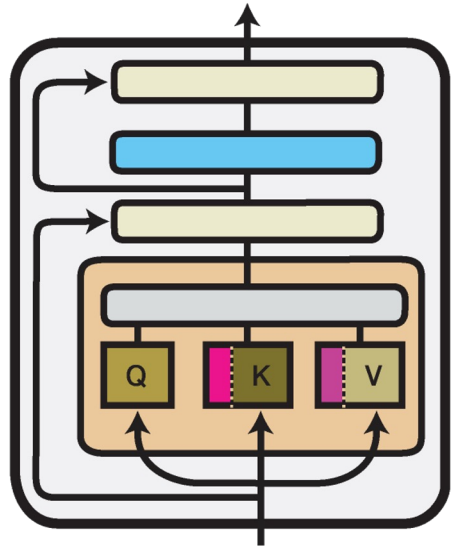
Function Composition

Slides: https://tinyurl.com/modular-fine-tuning-tutorial

Input Composition

One-sentence idea:
Fine-tune part of your attention input – soft prompt

```
def prompt_tuning_attention(input_ids):
    q = x @ W_q
    k = cat([s_k, x]) @ W_k   # prepend a
    v = cat([s_v, x]) @ W_v   # soft prompt
    return softmax(q @ k.T) @ V
```



https://arxiv.org/abs/2104.08691

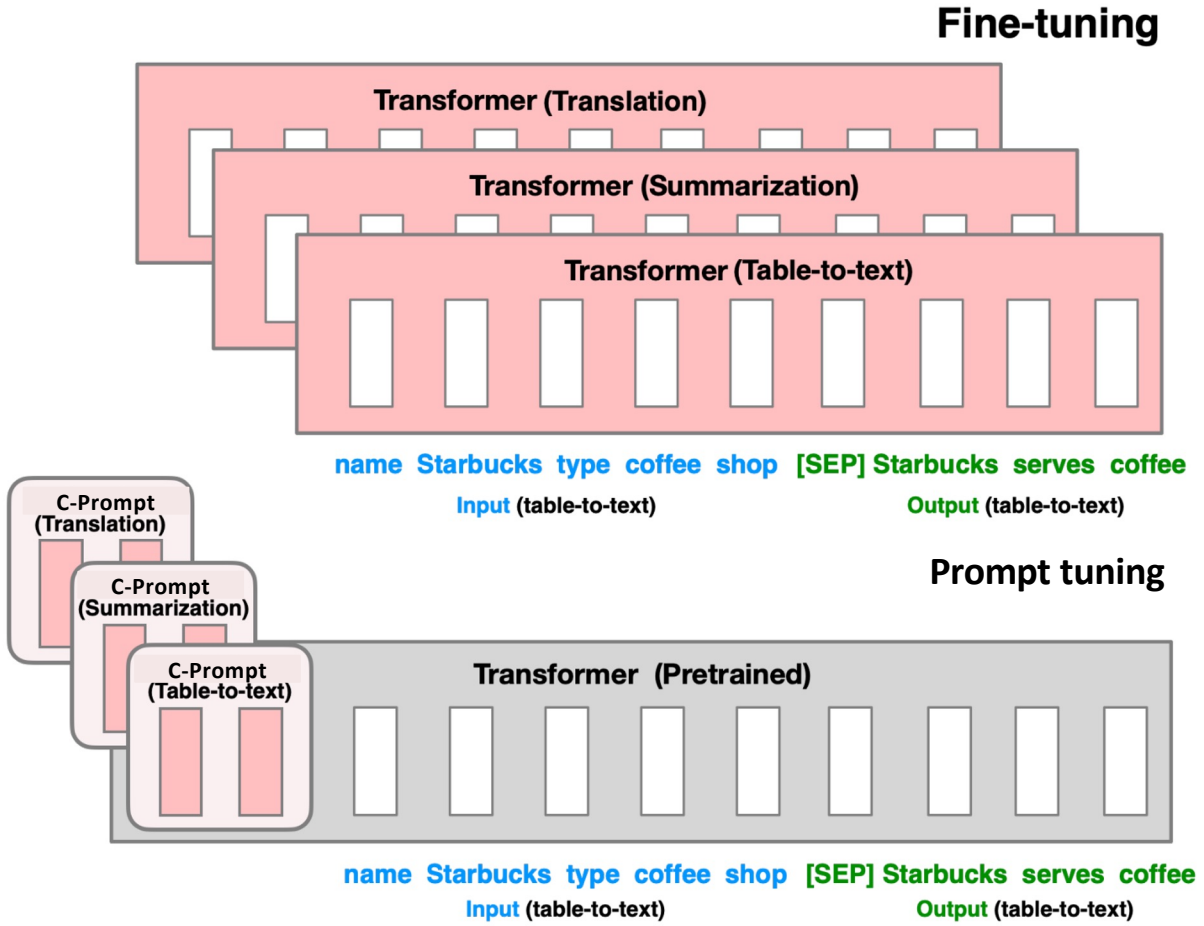
Image source: adapterhub.ml

Slide Source: https://docs.google.com/presentation/d/1wY3pcS-rnxKGQTSF9cCTft4xd9l5B-4diJ6xb2EfBOI/edit#slide=id.p

Prompt Tuning

- Instead, we can directly learn a continuous prompt ϕ that is prepended to the input [Liu et al., 2021; Hambardzumyan et al., 2021; Lester et al., 2021]

- ϕ is typically a matrix consisting of a sequence of continuous prompt embeddings



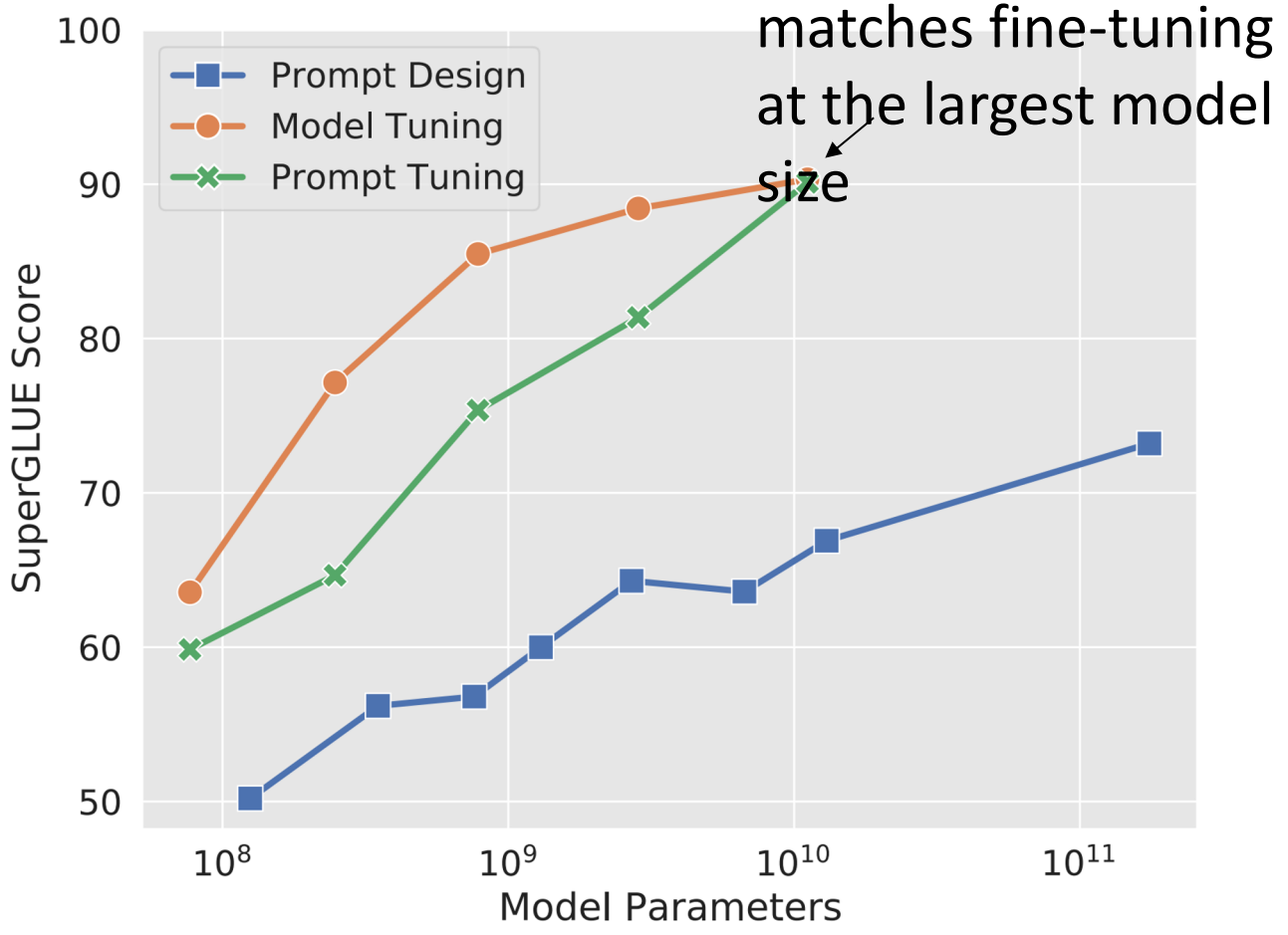
Fine-tuning vs Prompt tuning (adapted from [Li & Liang, 2021])

Slides: https://tinyurl.com/modular-fine-tuning-tutorial

Prompt Tuning Only Works Well at Scale

- Only using trainable parameters at the input layer limits capacity for adaptation

→ Prompt tuning performs poorly at smaller model sizes and on harder tasks [Mahabadi et al., 2021; Liu et al., 2022]

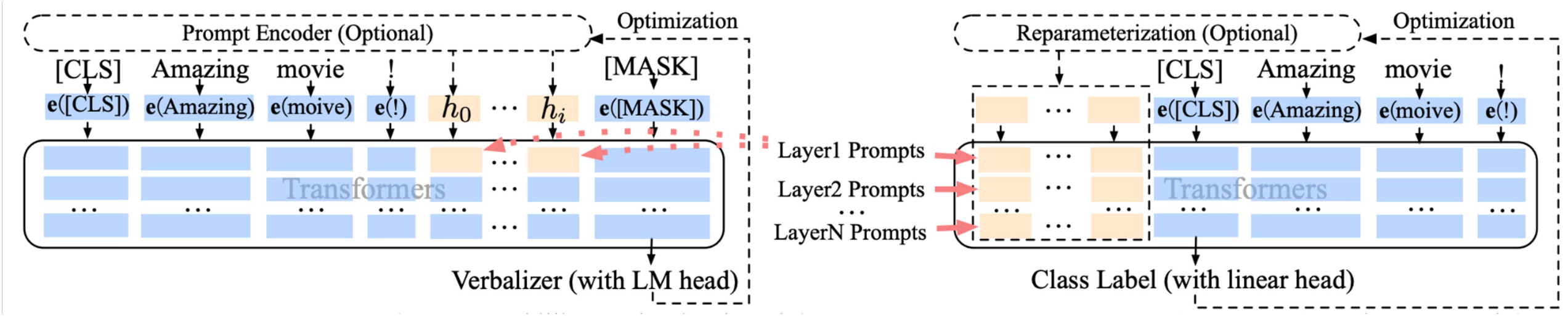


Prompt tuning vs standard fine-tuning and prompt design across T5 models of different sizes

Slides: https://tinyurl.com/modular-fine-tuning-tutorial

Multi-Layer Prompt Tuning (Prefix Tuning)

- Instead of learning ϕ_i parameters only at the input layer, we can learn them at *every layer* of the model [Li & Liang, 2021; Liu et al., 2022]
- Continuous prompts in later layers are more important

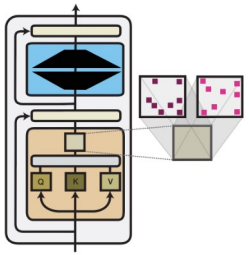
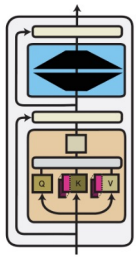


Prompt tuning

Multi-layer prompt tuning

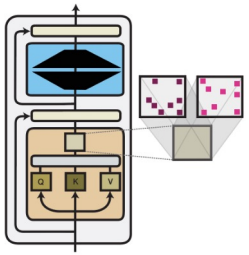
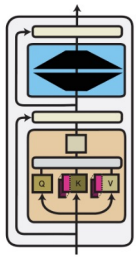
Slides: https://tinyurl.com/modular-fine-tuning-tutorial

Computation Functions Comparison

		Parameter efficiency	Training efficiency	Inference efficiency	Performance	Composition-ality
Parameter composition		+	-	++	+	+
Input composition		Only add a small number of parameters	Extend the model's context window		Requires large models to perform well	Continuous prompts have been composed

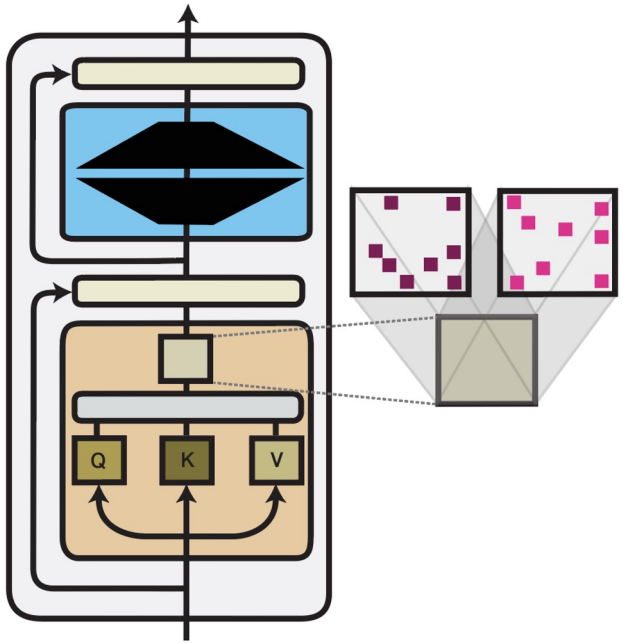
Slides: https://tinyurl.com/modular-fine-tuning-tutorial

Computation Functions Comparison

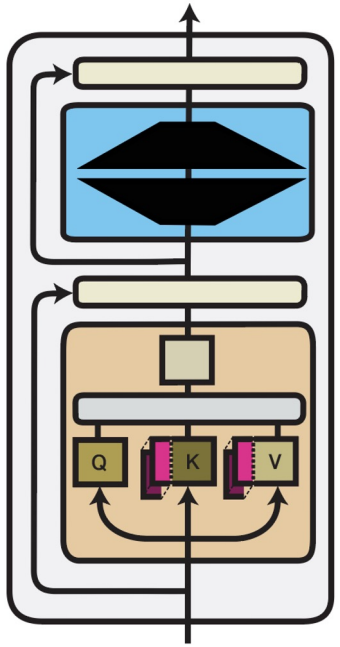
		Parameter efficiency	Training efficiency	Inference efficiency	Performance	Composition-ality
Parameter composition		+	-	++	+	+
Input composition		++	--	--	-	+

Slides: https://tinyurl.com/modular-fine-tuning-tutorial

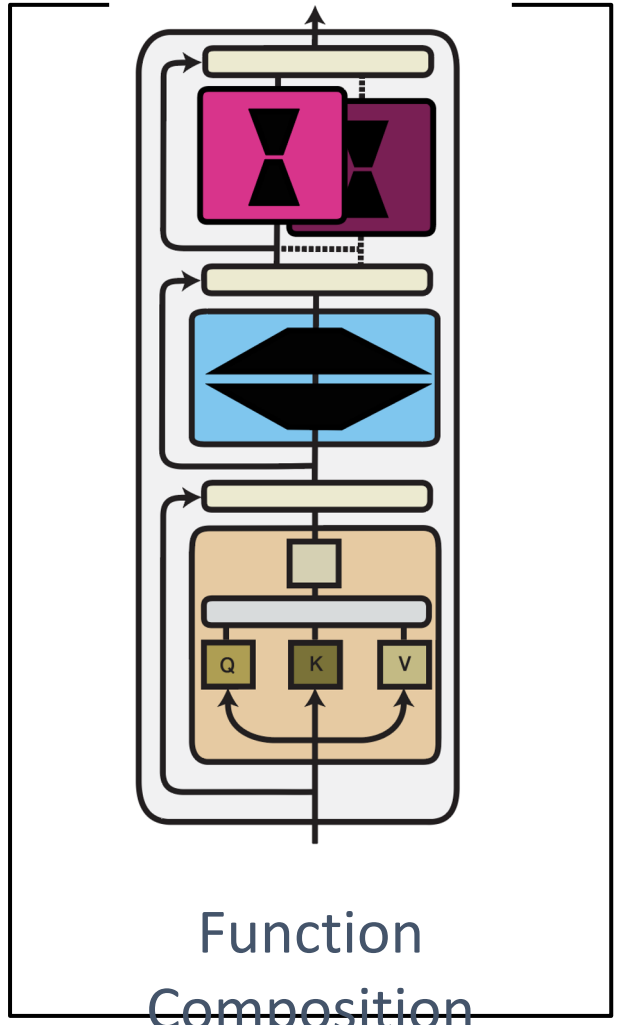
Three Computation Functions



Parameter Composition



Input Composition



Function Composition

Slides: https://tinyurl.com/modular-fine-tuning-tutorial

Function Composition

- Function composition augments a model's functions with new task-specific functions:

$$f_i'(\boldsymbol{x}) = f_{\theta_i}(\boldsymbol{x}) \odot f_{\phi_i}(\boldsymbol{x})$$

- Most commonly used in multi-task learning where modules of different tasks are composed.
- Relevant surveys: [Ruder, 2017; Crawshaw, 2020]

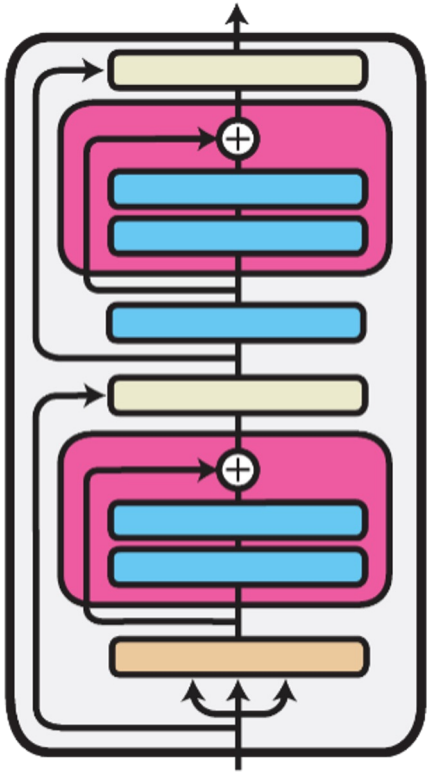
Slides: https://tinyurl.com/modular-fine-tuning-tutorial

Adapters

One-sentence idea:
Add fully-connected networks after attention and FFN layers

Pseudocode:

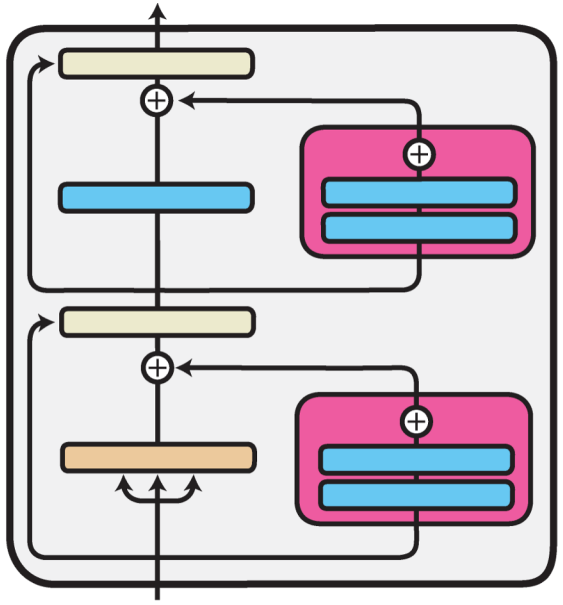
```
def transformer_block_with_adapter(x):
    residual = x
    x = SelfAttention(x)
    x = FFN(x)   # adapter
    x = LN(x + residual)
    residual = x
    x = FFN(x)   # transformer FFN
    x = FFN(x)   # adapter
    x = LN(x + residual)
    return x
```



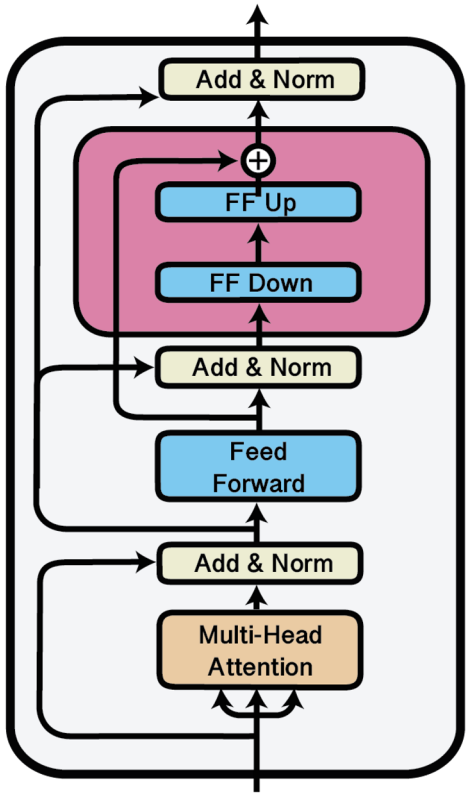
https://arxiv.org/abs/1902.00751

Image source: adapterhub.ml

Sequential and Parallel Adapters



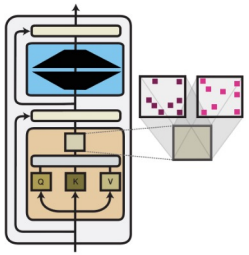
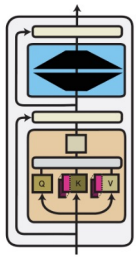
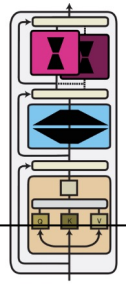
Two parallel adapters
[Stickland & Murray, 2019]



A sequential adapter
[Houlsby et al., 2019]

Slides: https://tinyurl.com/modular-fine-tuning-tutorial

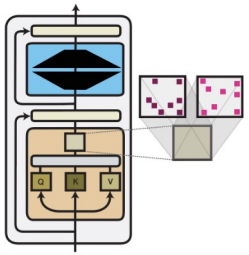
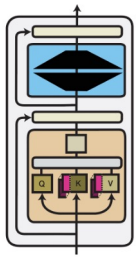
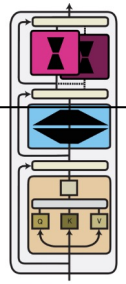
Computation Functions Comparison

		Parameter efficiency	Training efficiency	Inference efficiency	Performance	Composition-ality
Parameter composition		+	-	++	+	+
Input composition		++	--	--	-	+
Function Composition		Adapters depend on the hidden size	Does not require gradients of frozen params	New functions increase # of operations	Match or outperform standard fine-tuning	Adapters can be composed

Slides: https://tinyurl.com/modular-fine-tuning-tutorial

Slides: https://tinyurl.com/modular-fine-tuning-tutorial

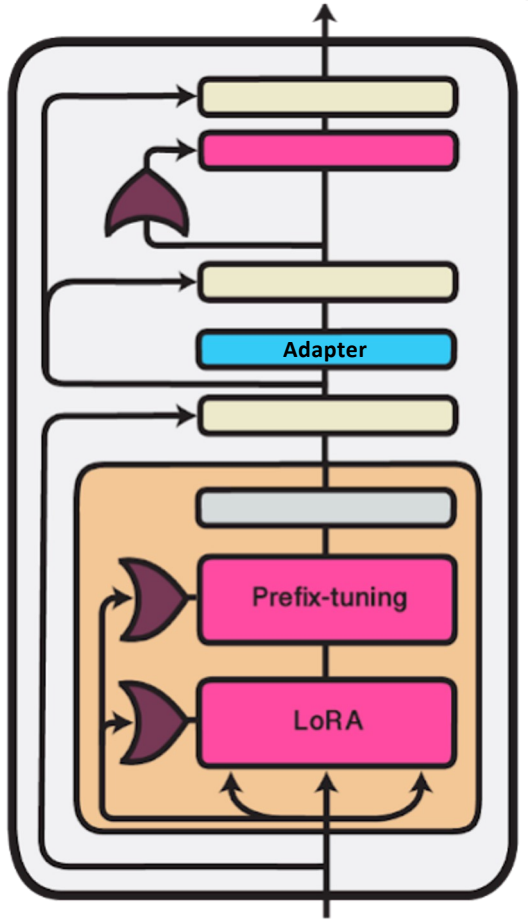
Computation Functions Comparison

		Parameter efficiency	Training efficiency	Inference efficiency	Performance	Composition-ality
Parameter composition		+	-	++	+	+
Input composition		++	--	--	-	+
Function Composition		-	+	-	++	+

Combining Computation Functions

- Different computation functions are complimentary
- We can combine different approaches in the same model

- UniPELT [Mao et al., 2022] combines adapters, prefix tuning, and LoRA in the same architecture with a learned scalar gate for each



Slides: https://tinyurl.com/modular-fine-tuning-tutorial

What matters in PEFT?

- Number of parameters…
  which ones?
  - Trainable — memory
  - Changed — storage
  - Rank of the delta — how much the network subspace is affected

  Current papers frequently do not distinguish between these three

Slide Source: https://docs.google.com/presentation/d/1wY3pcS-rnxKGQTSF9cCTft4xd9l5B-4diJ6xb2EfBOI/edit#slide=id.p

Method	% Trainable parameters	% Changed parameters	Evaluated on <1B	<20B	>20B
Adapters (Houlsby et al., 2019)	0.1 - 6	0.1 - 6	yes	yes	yes
AdaMix (Wang et al., 2022)	0.1 - 0.2	0.1 - 0.2	yes	no	no
SparseAdapter (He et al., 2022b)	2.0	2.0	yes	no	no
BitFit (Ben-Zaken et al., 2021)	0.05 - 0.1	0.05 - 0.1	yes	yes	yes
DiffPruning (Guo et al., 2020)	200	**0.5**	yes	no	no
Fish-Mask (Sung et al., 2021)	0.01 - 0.5	0.01 - 0.5	yes	yes	no
Prompt Tuning (Lester et al., 2021)	0.1	0.1	yes	yes	yes
Prefix-Tuning (Li and Liang, 2021)	0.1 - 4.0	0.1 - 4.0	yes	yes	yes
IPT (Qin et al., 2021)	56.0	56.0	yes	no	no
MAM Adapter (He et al., 2022a)	0.5	0.5	yes	no	no
Parallel Adapter (He et al., 2022a)	0.5	0.5	yes	no	no
Intrinsinc SAID (Aghajanyan et al., 2020)	0.001 - 0.1	**∼0.1 or 100**	yes	yes	no
LoRa (Hu et al., 2022)	0.01 - 0.5	**∼0.5 or ∼30**	yes	yes	yes
UniPELT (Mao et al., 2021)	1.0	1.0	yes	no	no
Compacter (Karimi Mahabadi et al., 2021)	0.05-0.07	**∼0.07 or ∼0.1**	yes	yes	no
PHM Adapter (Karimi Mahabadi et al., 2021)	0.2	**∼0.2 or ∼1.0**	yes	no	no
KronA (Edalati et al., 2022)	0.07	**∼0.07 or ∼30.0**	yes	no	no
KronA$^{B}_{res}$ (Edalati et al., 2022)	0.07	**∼0.07 or ∼1.0**	yes	no	no
(IA)^3 (Liu et al., 2022)	0.02	0.02	no	yes	no
Ladder Side-Tuning(Sung et al., 2022)	7.5	7.5	yes	yes	no
FAR (Vucetic et al., 2022)	6.6-26.4	6.6-26.4	yes	no	no
S4-model (Chen et al., 2023)	0.5	**more than 0.5**	yes	yes	no

Slide Source: https://docs.google.com/presentation/d/1wY3pcS-rnxKGQTSF9cCTft4xd9l5B-4diJ6xb2EfBOI/edit#slide=id.p

Parameter efficient fine-tuning in practice

- github.com/adapter-hub/adapter-transformers
- github.com/huggingface/peft
- github.com/thunlp/OpenDelta