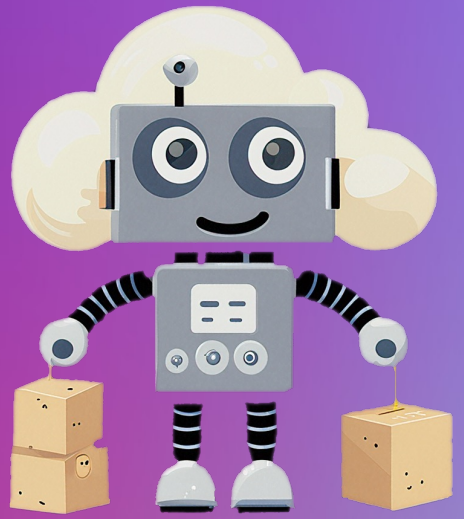
SES109

Parameter Efficient Finetuning (PEFT) with LoRA

Mariano Kamp

Principal Solutions Architect
Amazon Web Services



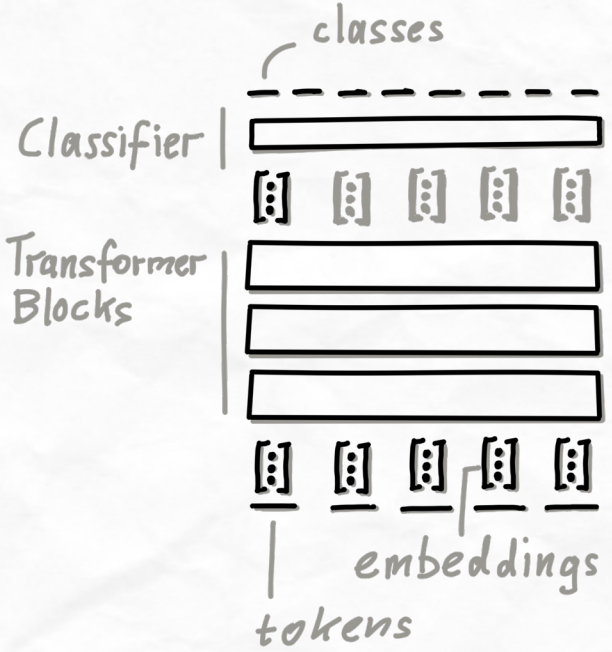
Model	BoolQ	CB	CoPA	MultiRC	Record	RTE	WiC	WSC
Few-shot	89.1	89.3	95	86.3/-	92.9/-	81.2	64.6	89.5
Finetuned	92.2	100/100	100	90.1/69.2	94.0/94.6	95.7	78.8	100

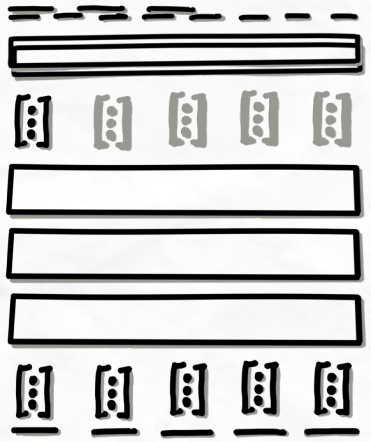
Table 8: Results on SuperGLUE dev set comparing PaLM-540B few-shot and finetuned.

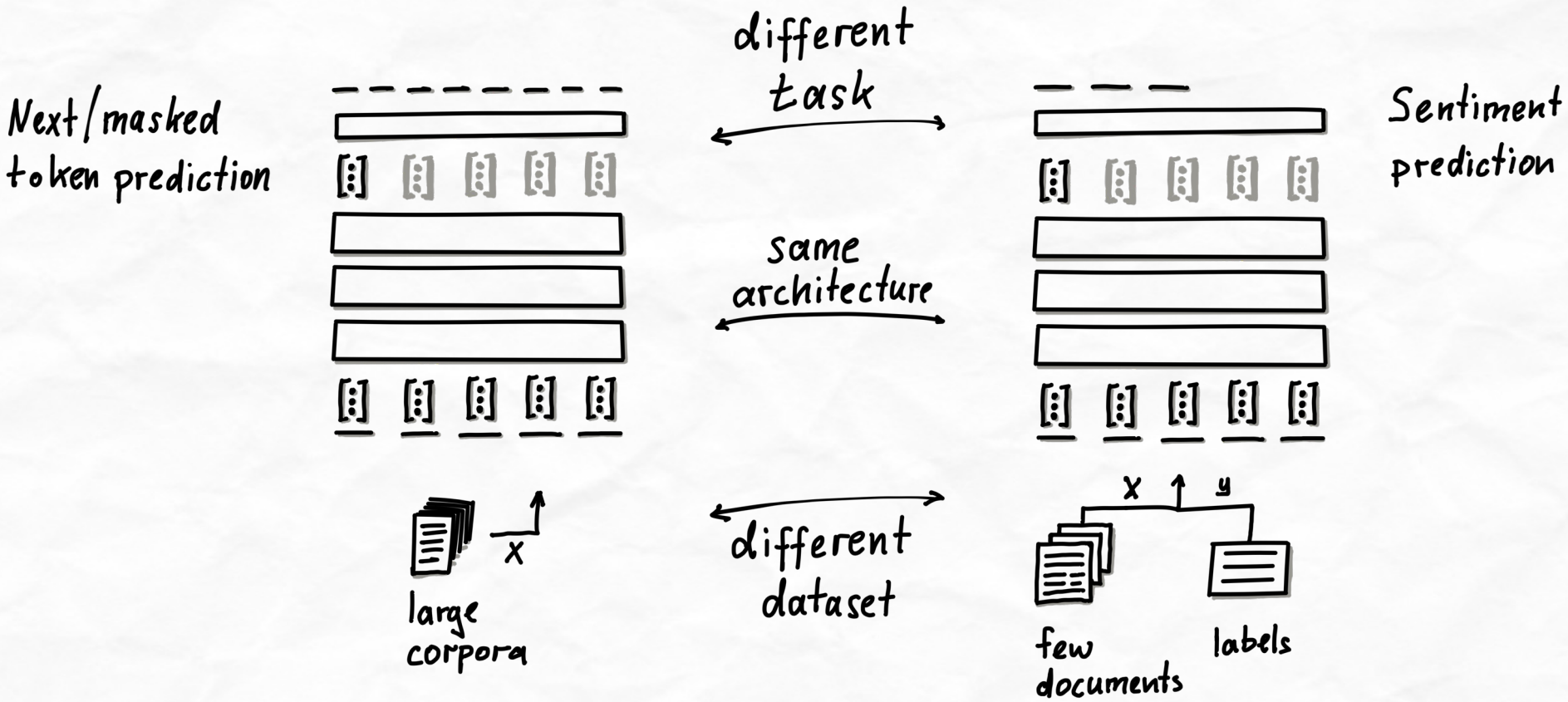
PaLM: Scaling Language Modeling with Pathways
Chowdhery, Narang, Devlin et al.

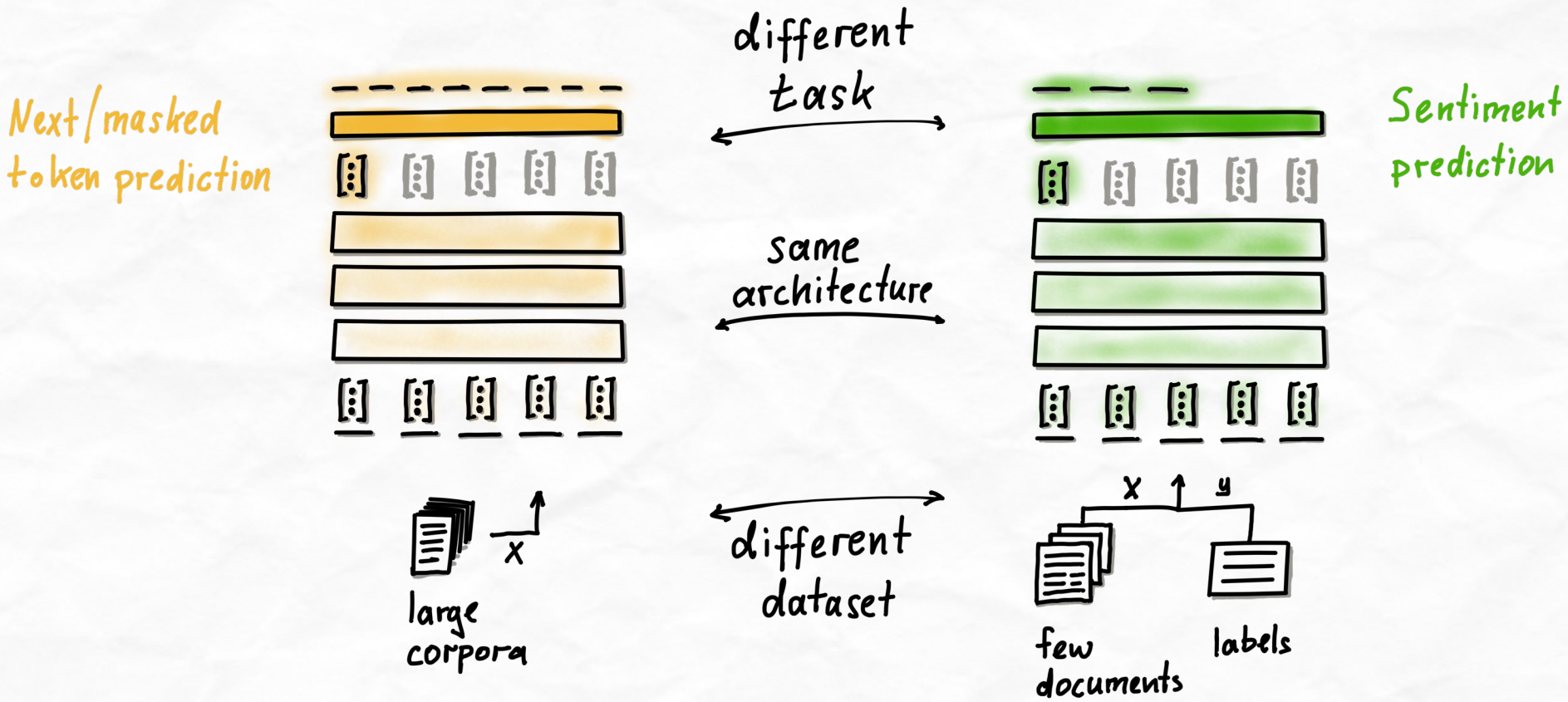
PRE-TRAINING

FINE-TUNING

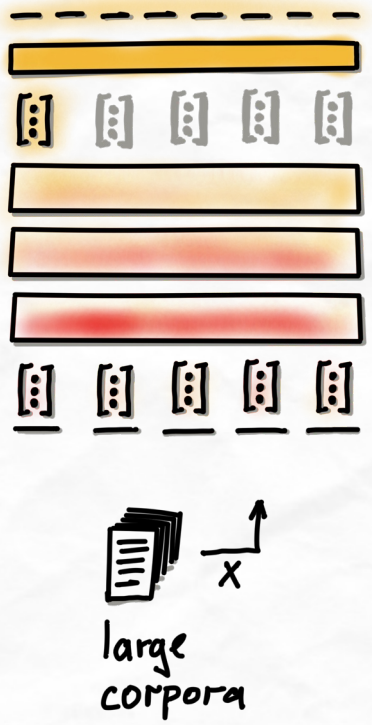




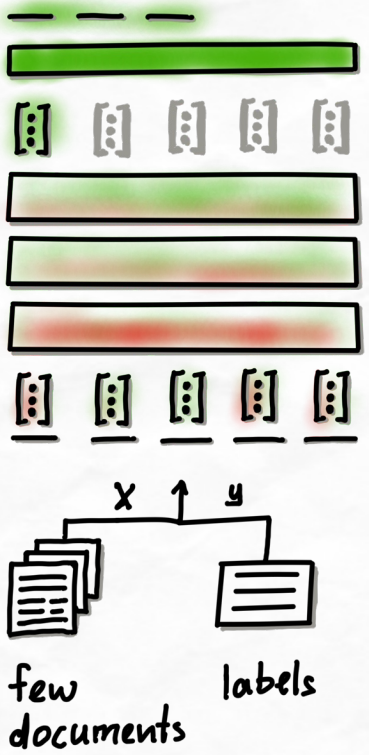




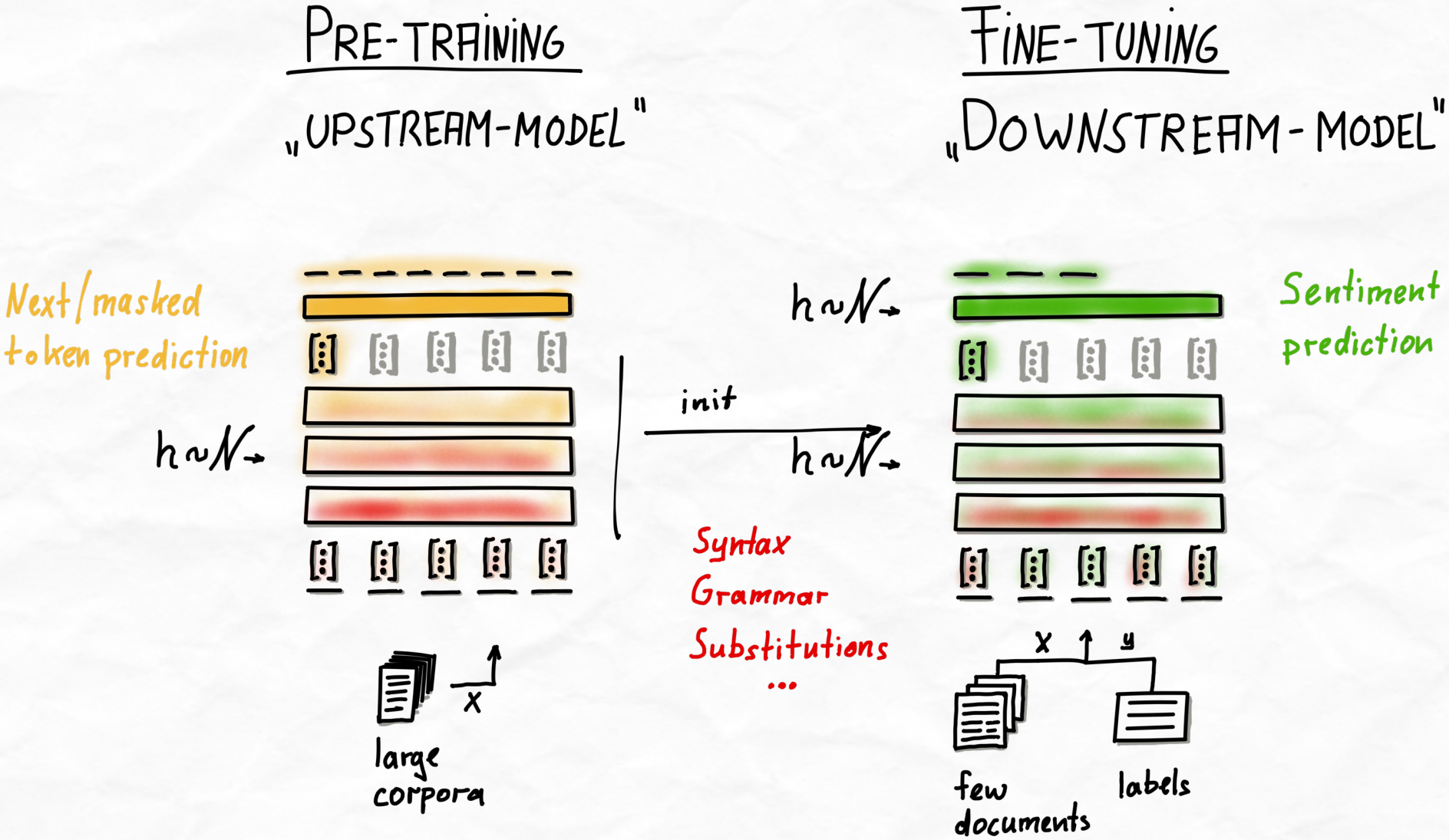
Next/masked token prediction



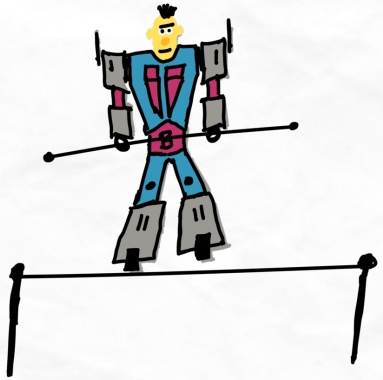
Syntax
Grammar
Substitutions
...



Sentiment prediction

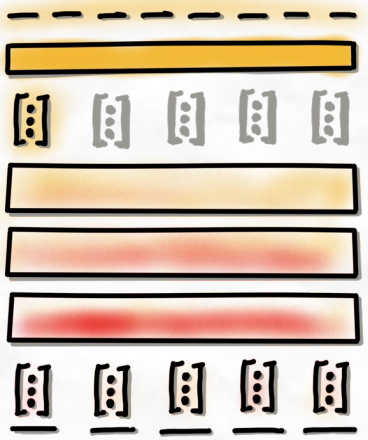


Fine-tuning



Next/masked token prediction

$h \sim N \rightarrow$

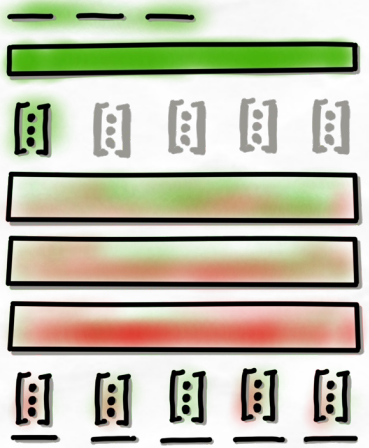


large corpora

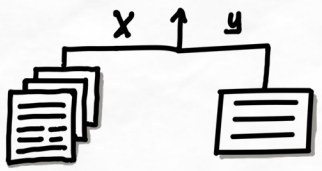
init

Syntax
Grammar
Substitutions
...

$h \sim N \rightarrow$

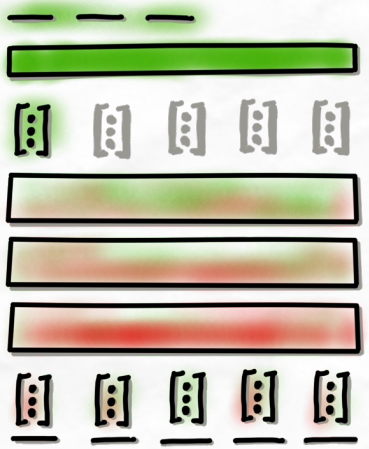


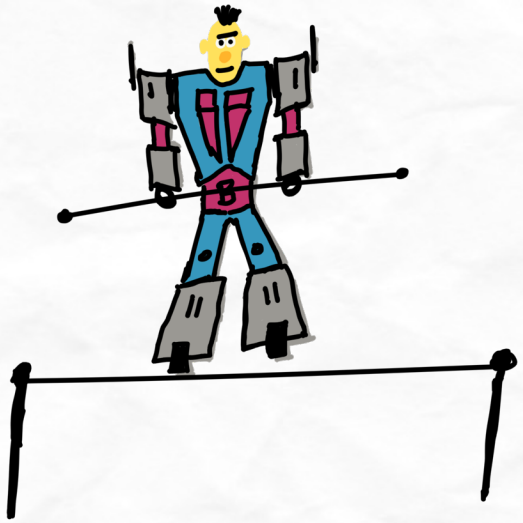
Sentiment prediction



few documents

labels

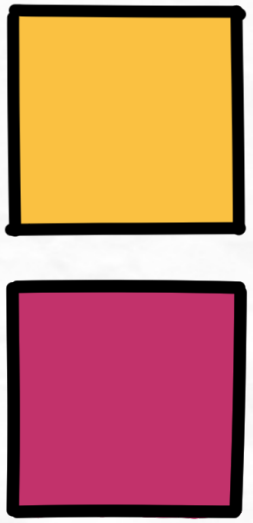




Tightrope Walk

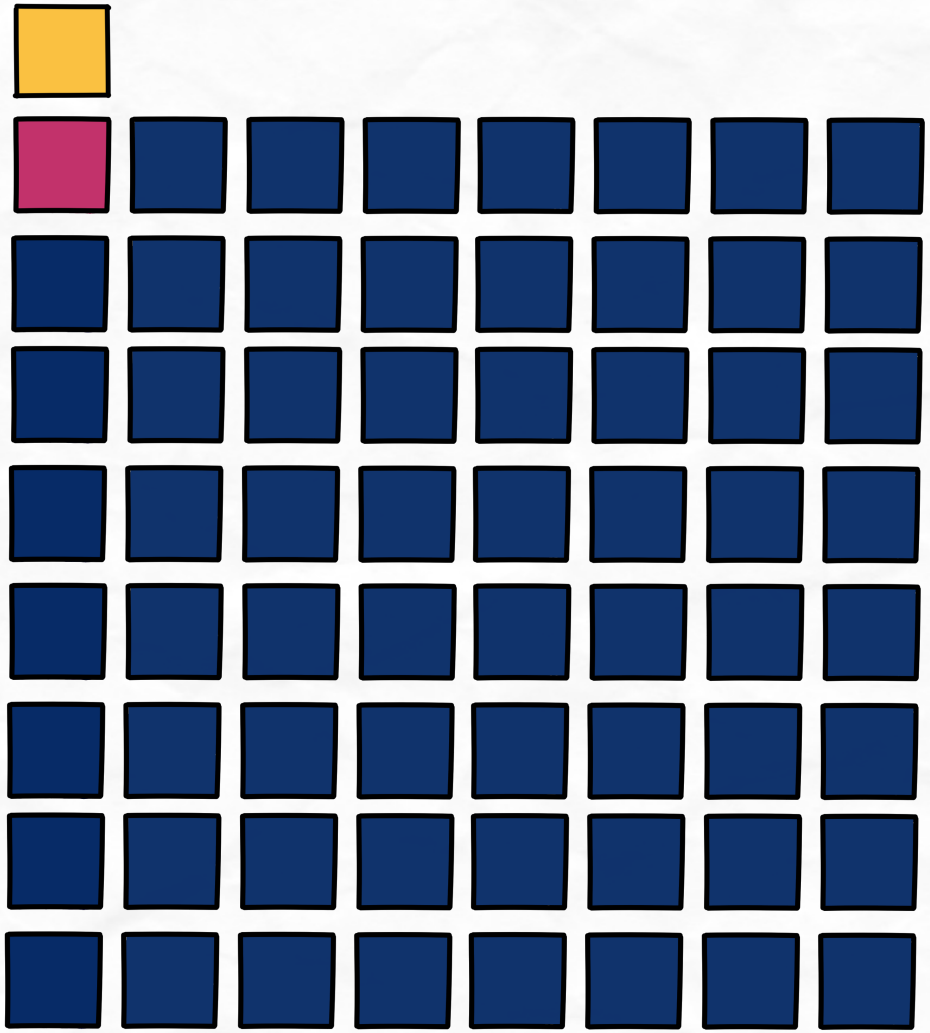
- Fewer epochs
- Cautious LR
- Per-Layer-LR
- Frozen layers

Size



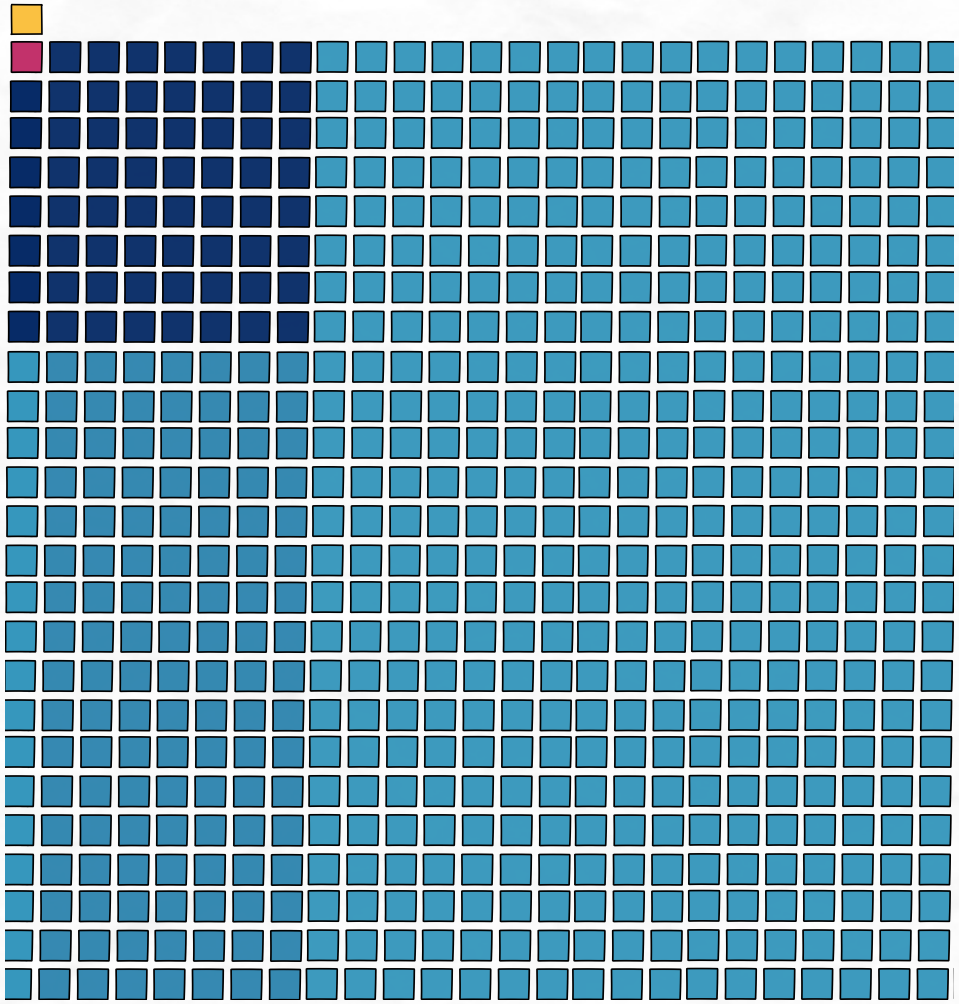
Labeled Data

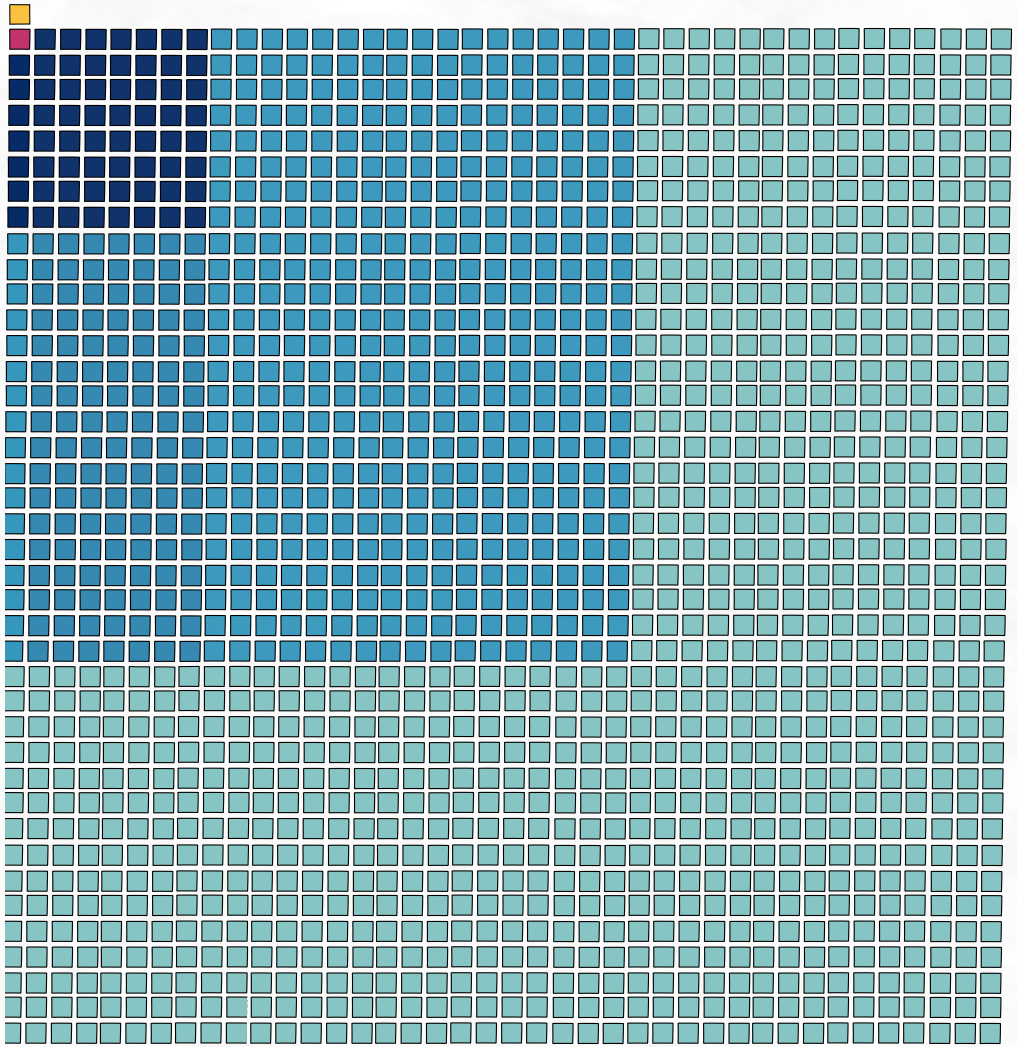
BERT Base 110M

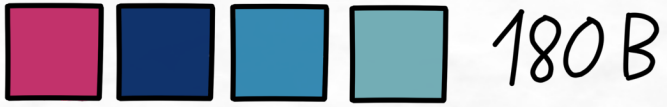


LABELED DATA

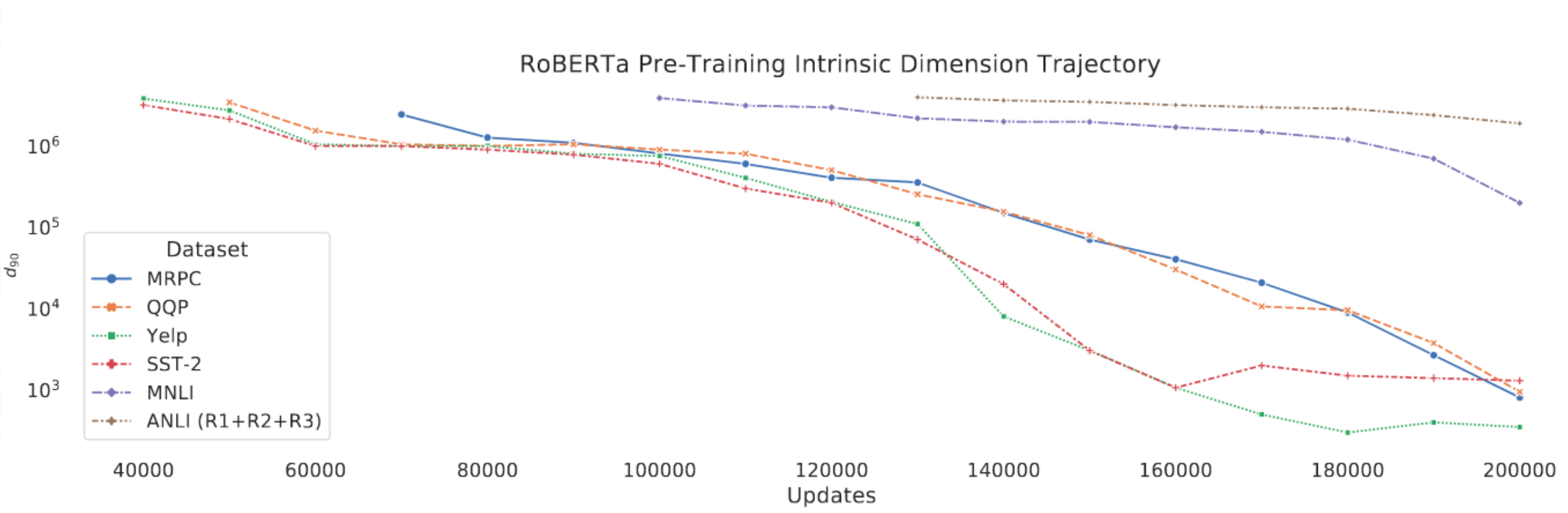
7B



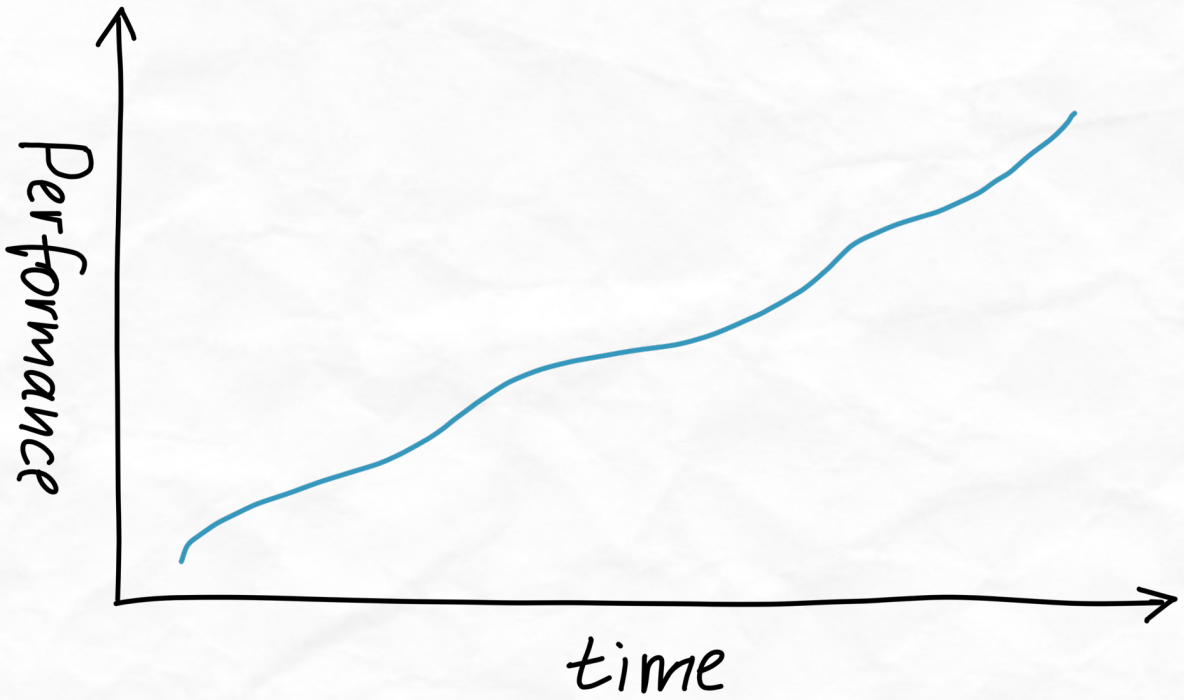


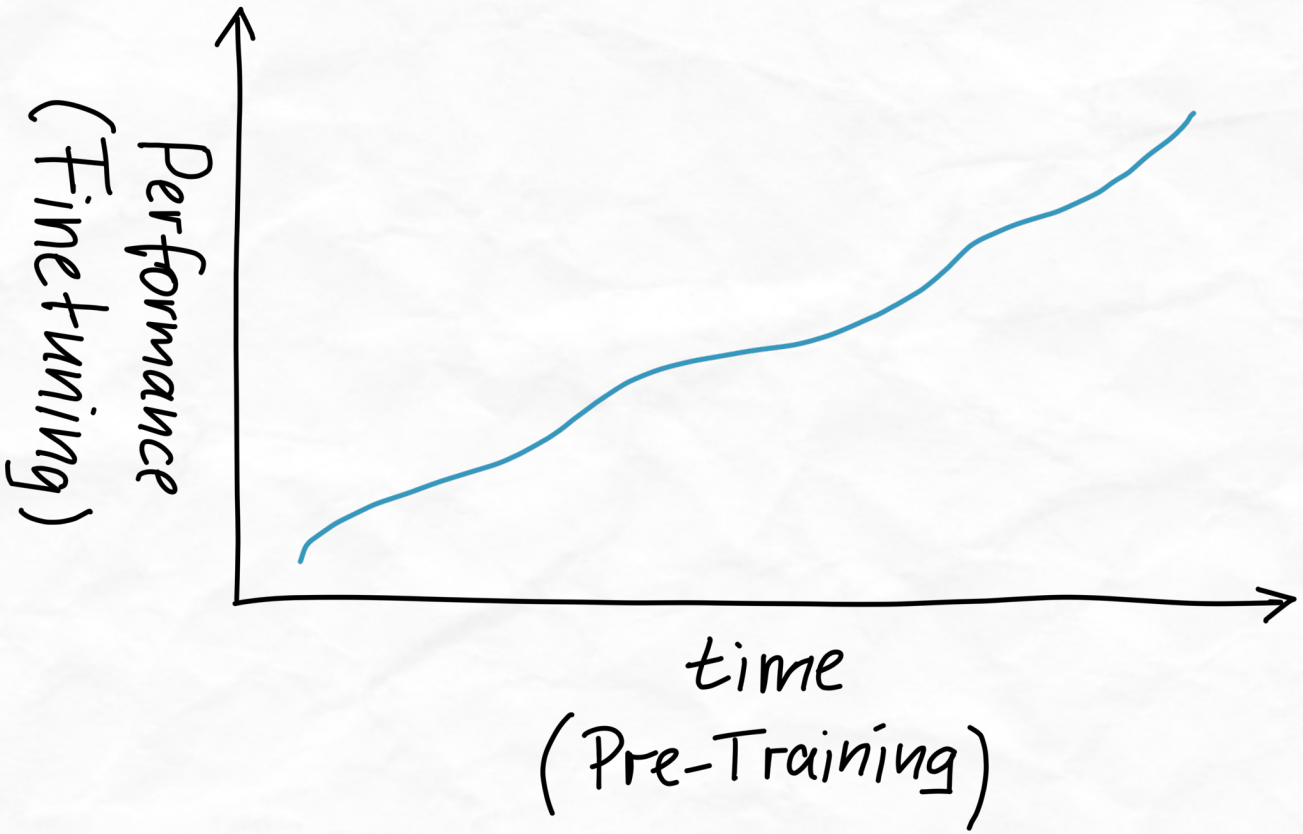


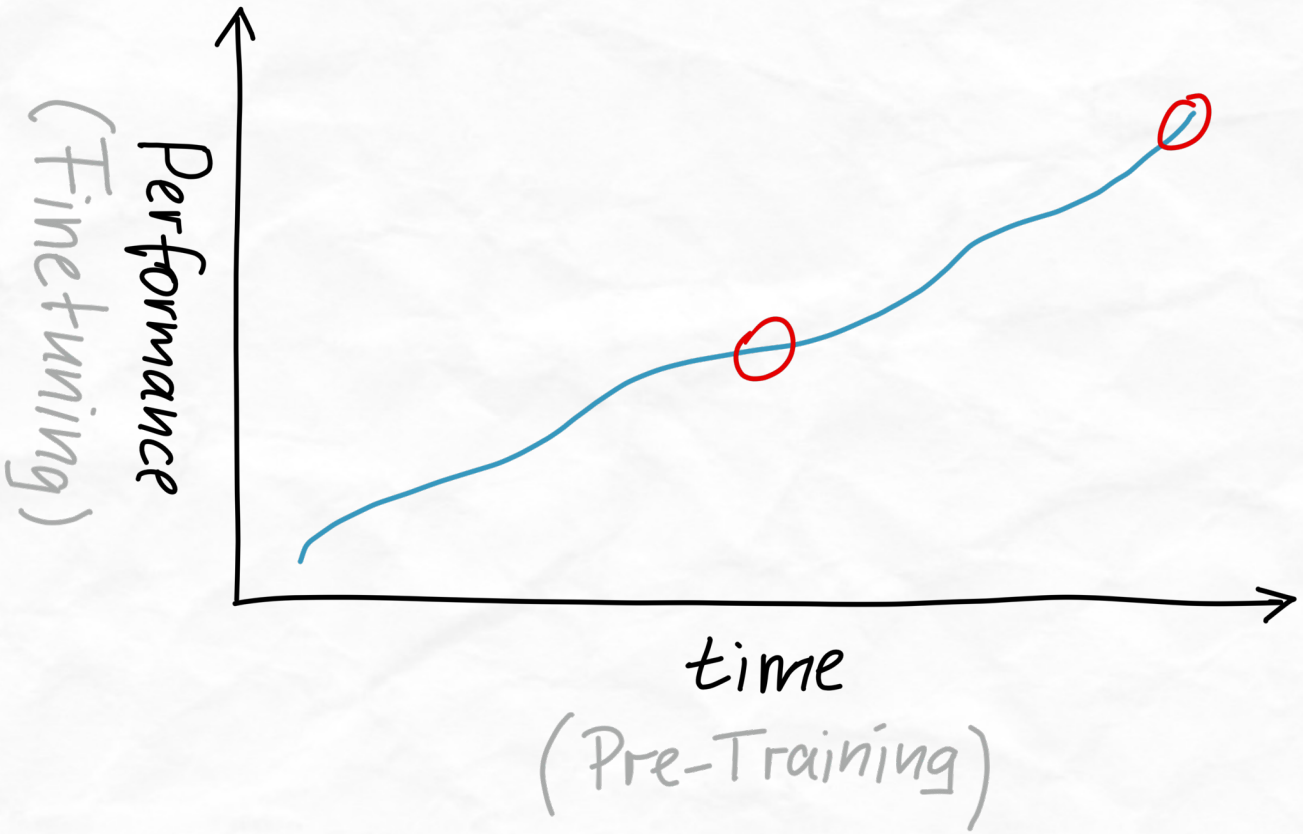
Intrinsic Dimensionality

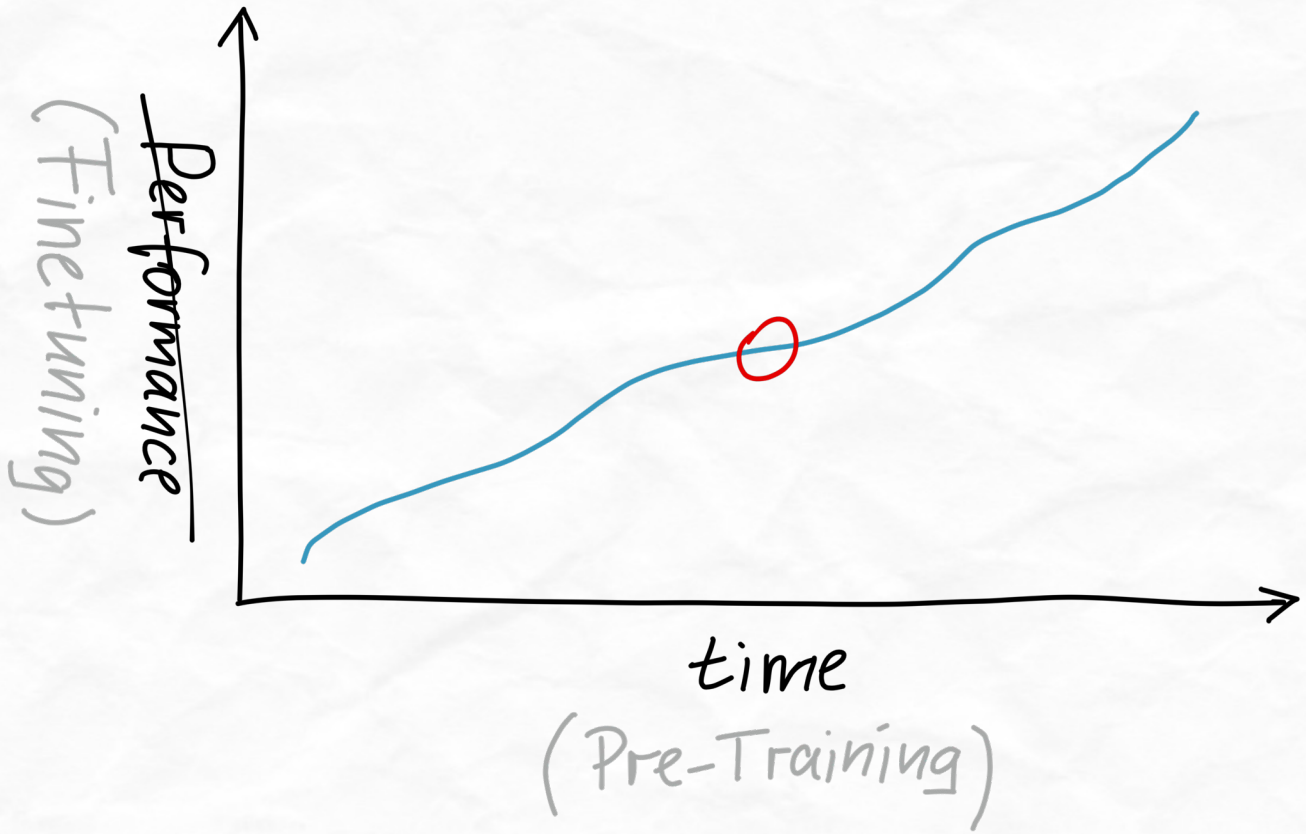


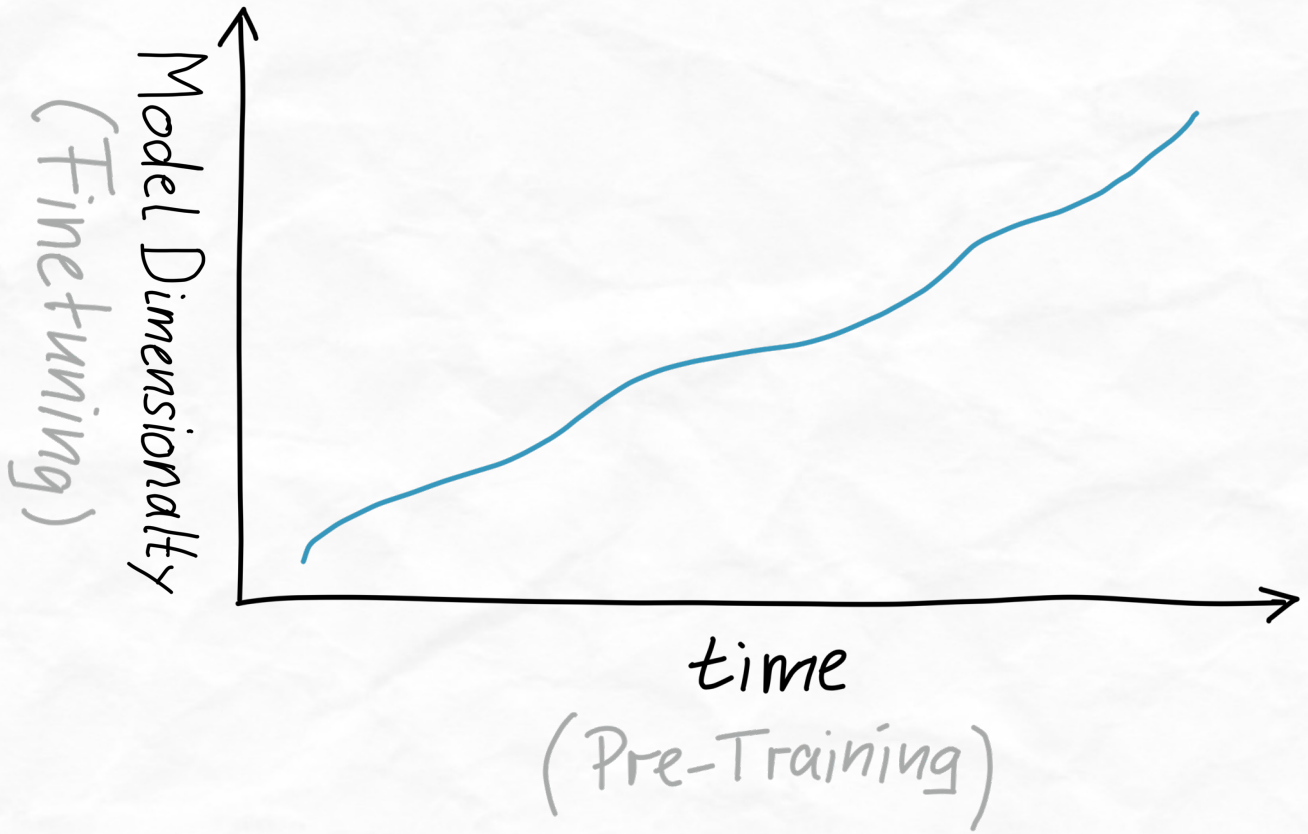
INTRINSIC DIMENSIONALITY EXPLAINS THE EFFECTIVENESS OF LANGUAGE MODEL FINE-TUNING
Aghajanyan, Zettlemoyer, Gupta, 2020

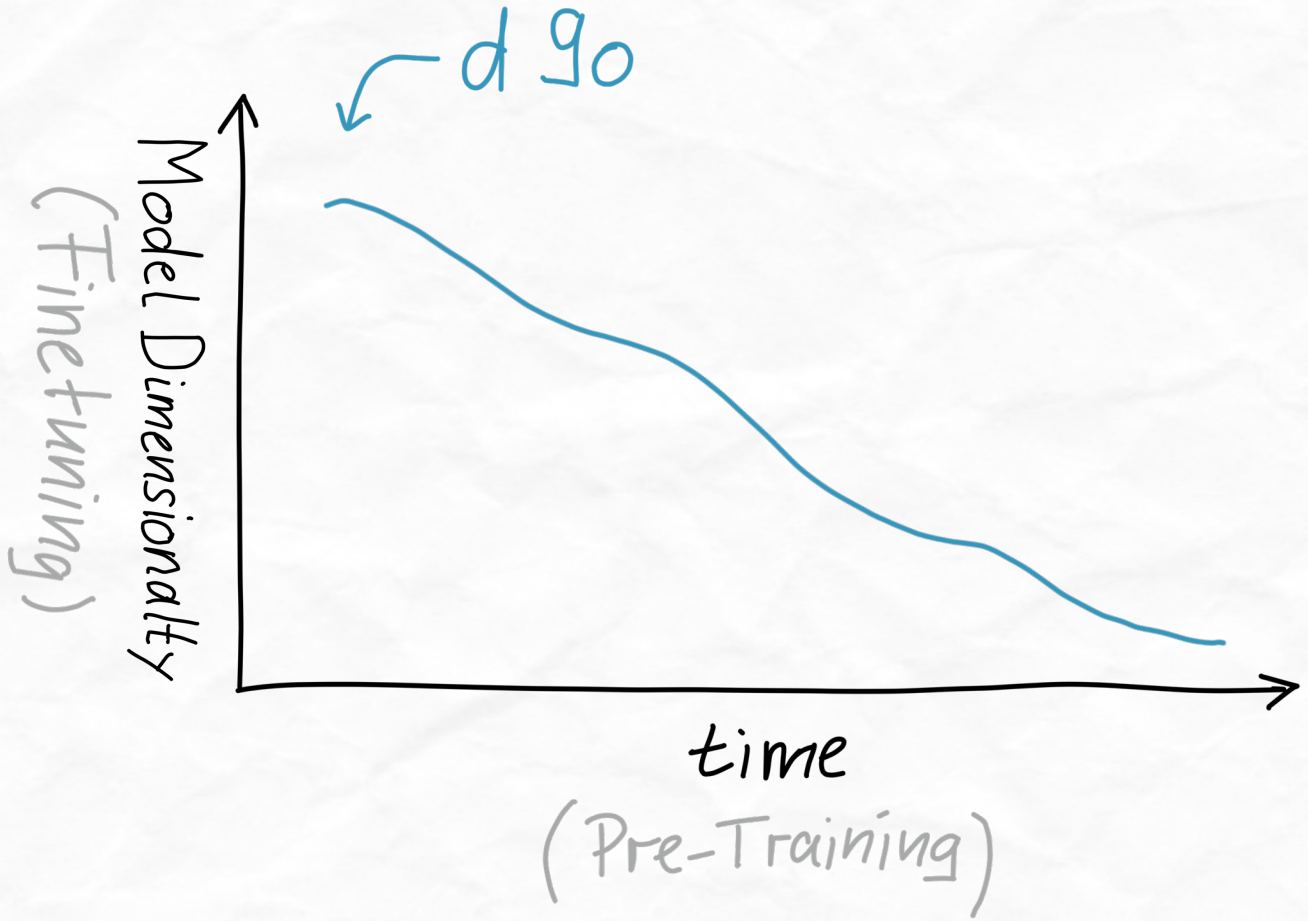


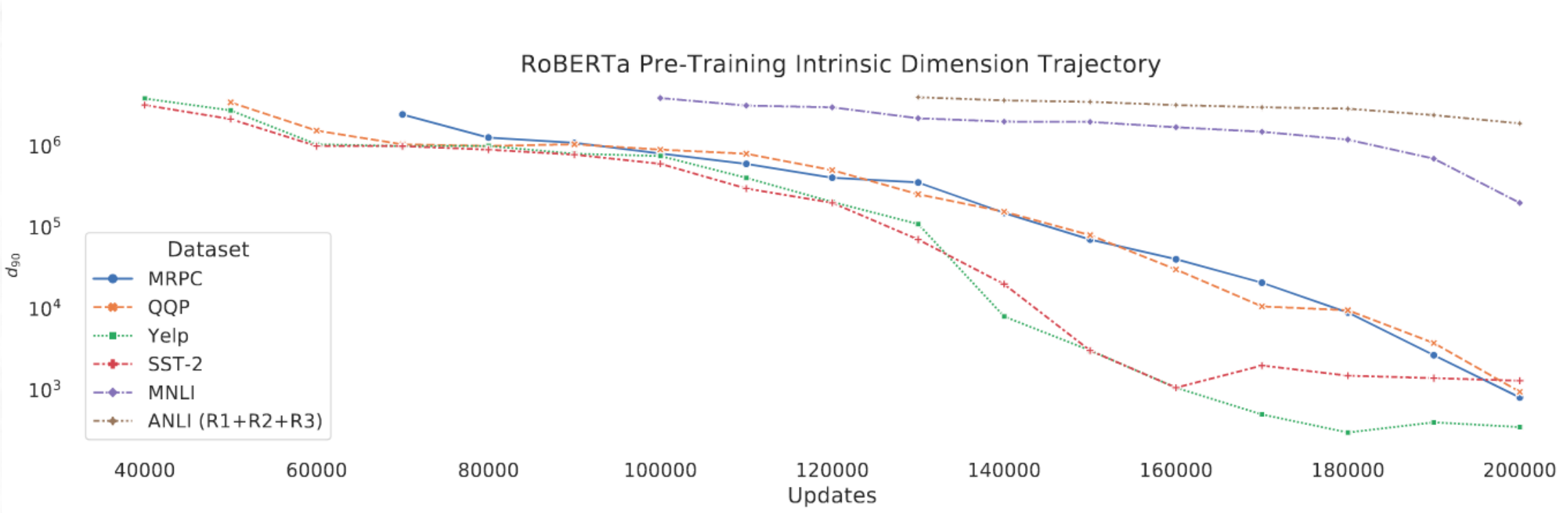




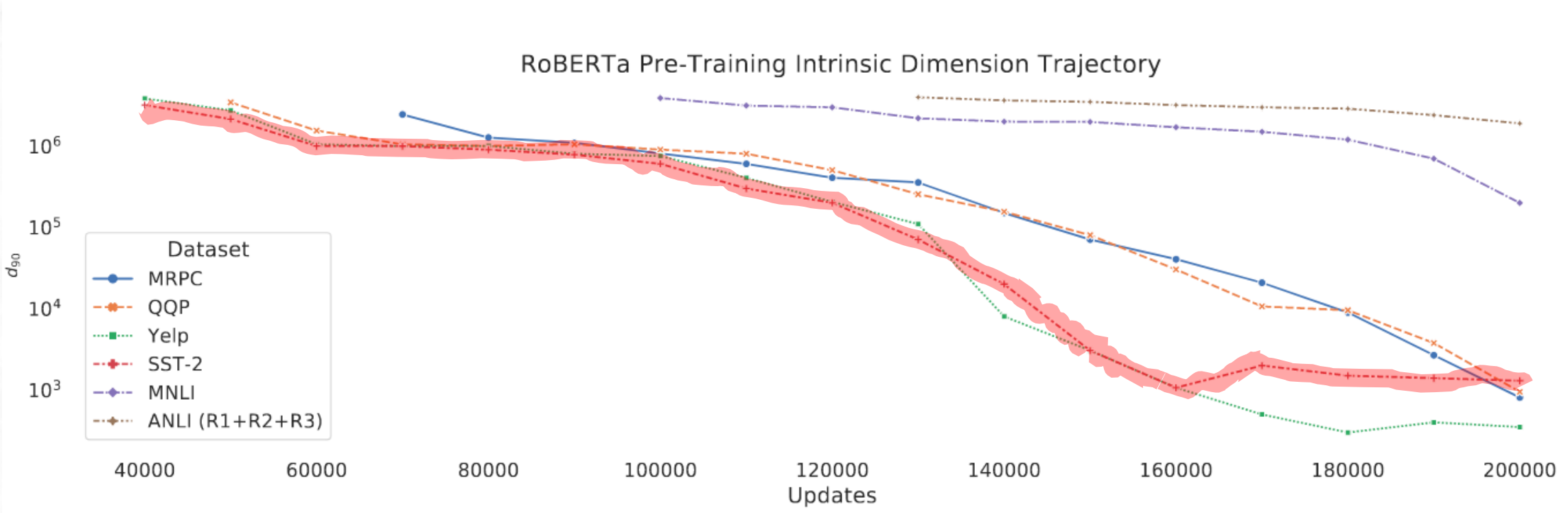




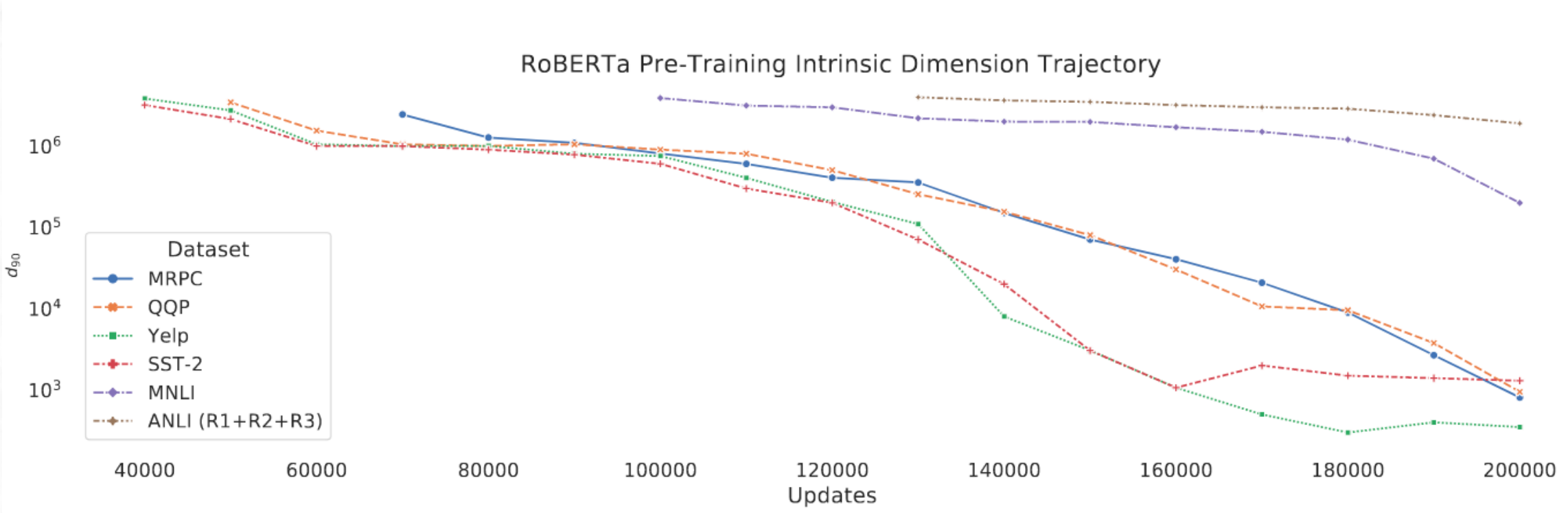




INTRINSIC DIMENSIONALITY EXPLAINS THE EFFECTIVENESS OF LANGUAGE MODEL FINE-TUNING
Aghajanyan, Zettlemoyer, Gupta, 2020

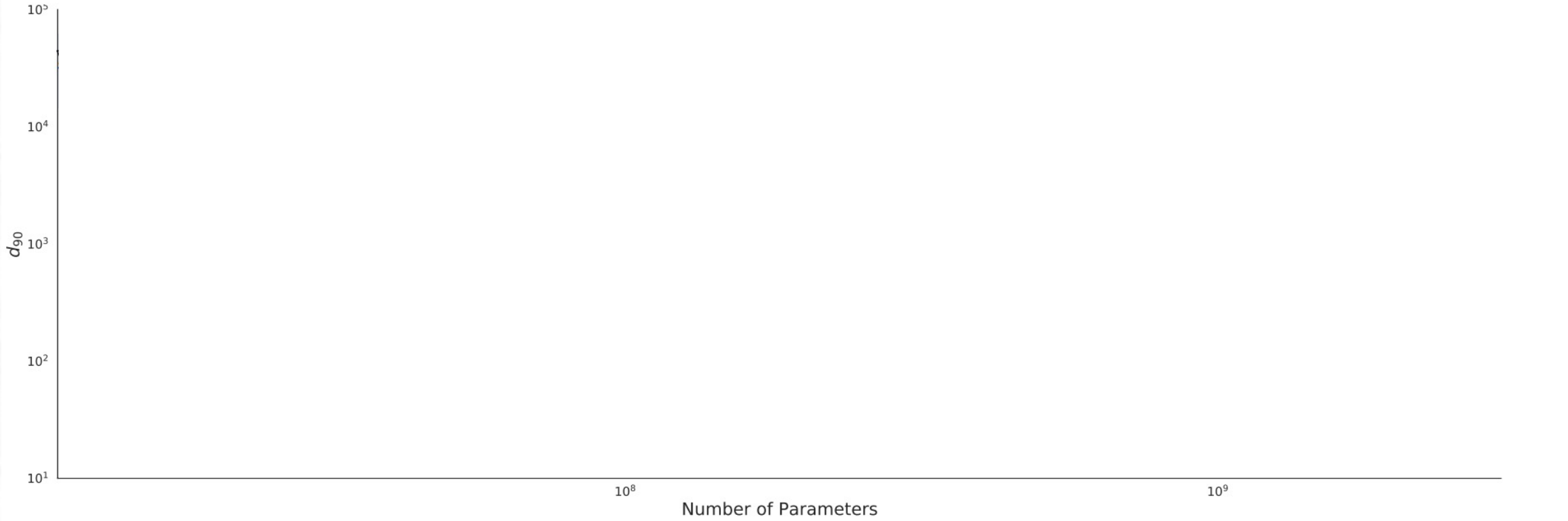


INTRINSIC DIMENSIONALITY EXPLAINS THE EFFECTIVENESS OF LANGUAGE MODEL FINE-TUNING
Aghajanyan, Zettlemoyer, Gupta, 2020



INTRINSIC DIMENSIONALITY EXPLAINS THE EFFECTIVENESS OF LANGUAGE MODEL FINE-TUNING
Aghajanyan, Zettlemoyer, Gupta, 2020

Intrinsic Dimensionality (by Model Size)

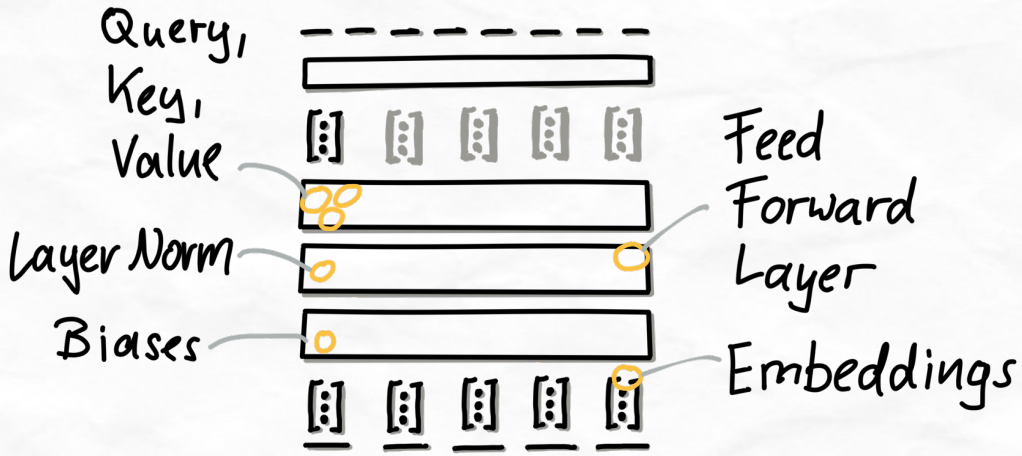


INTRINSIC DIMENSIONALITY EXPLAINS THE EFFECTIVENESS OF LANGUAGE MODEL FINE-TUNING
Aghajanyan, Zettlemoyer, Gupta, 2020

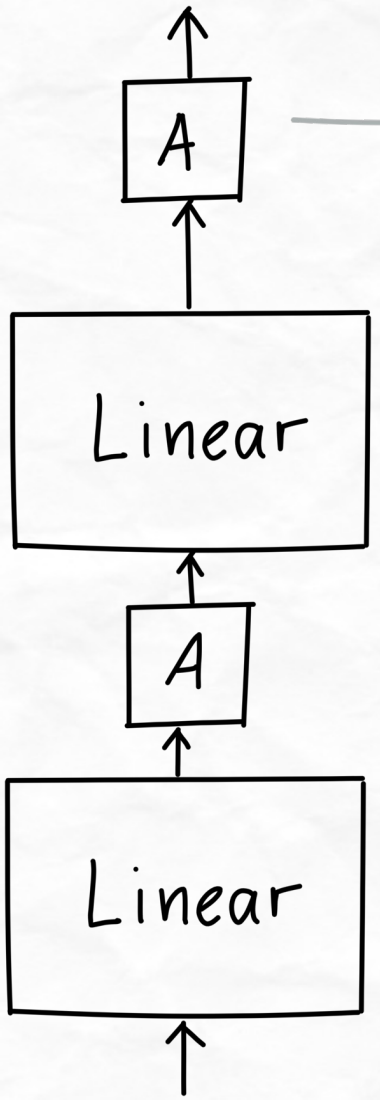
Parameter Efficient Fine-Tuning

(PEFT)

WHERE TO APPLY?



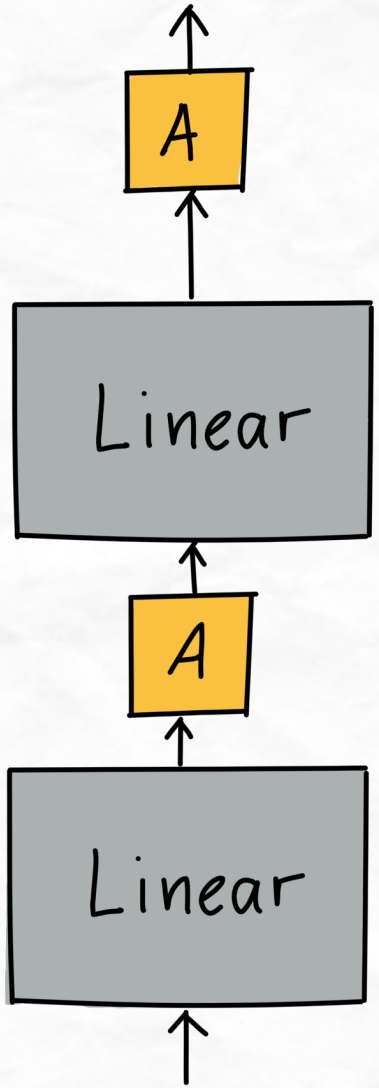
Adapter

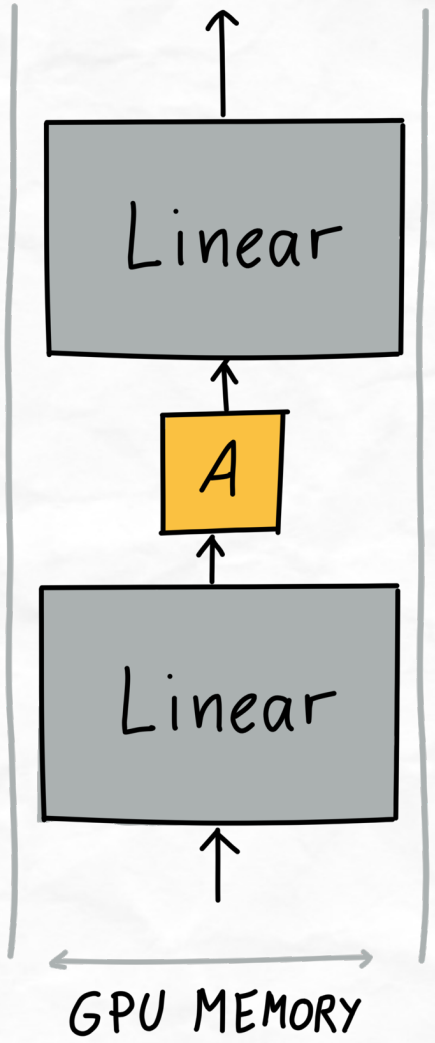


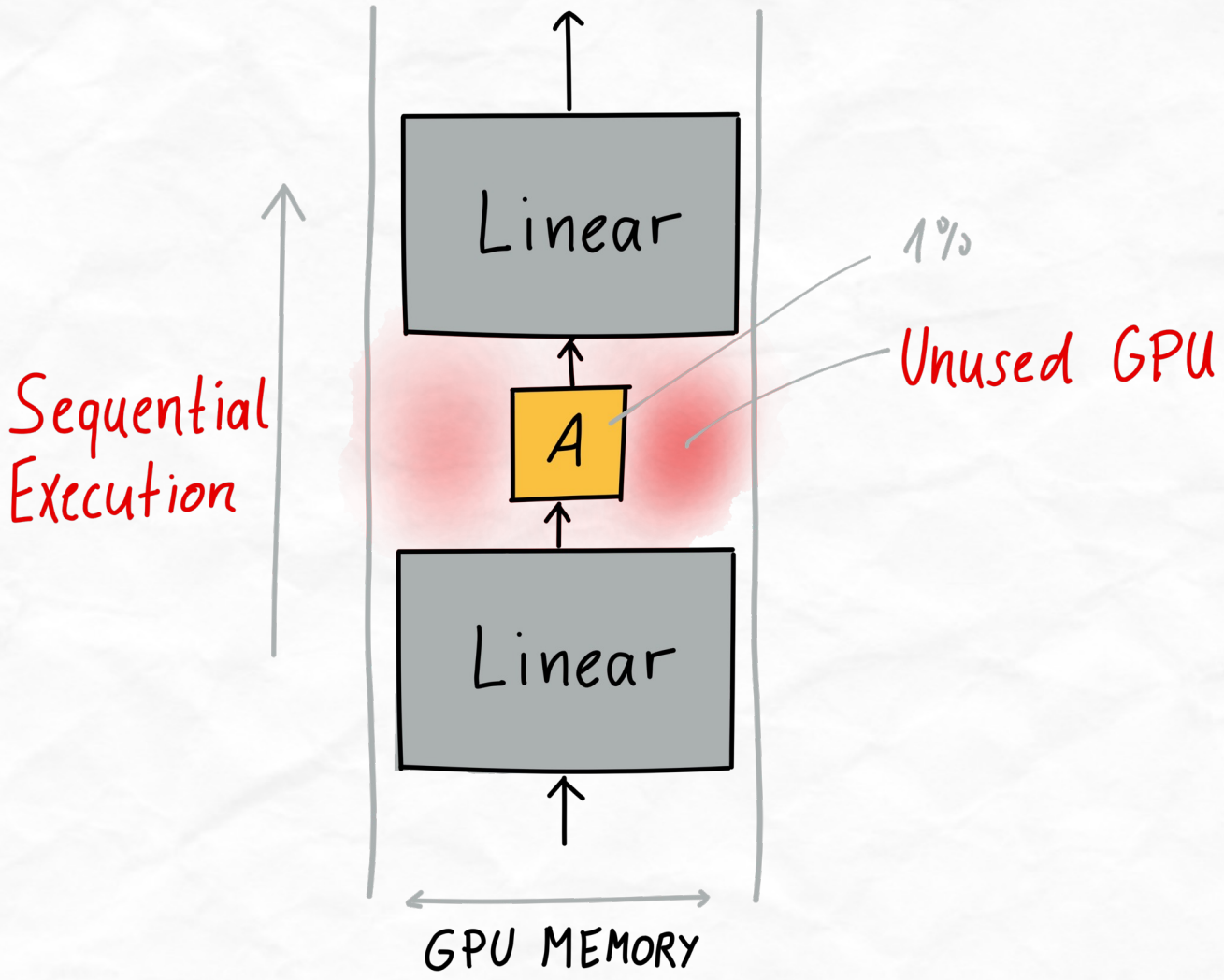
Adapters are trained

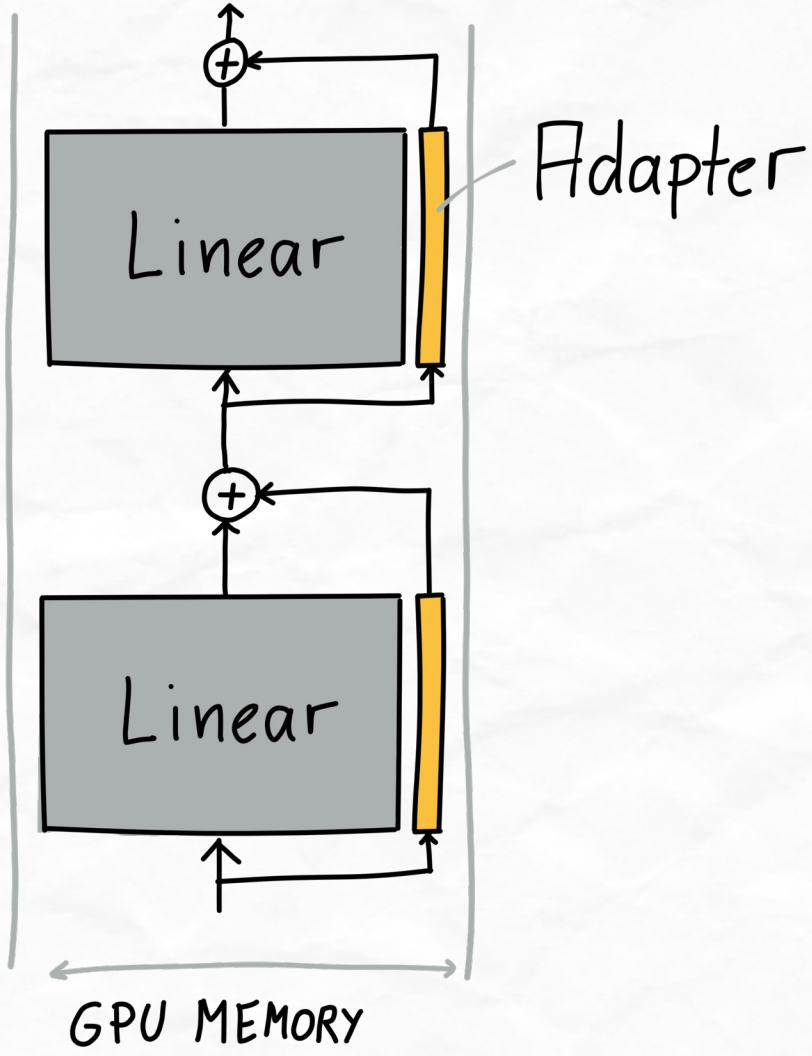
Adapted modules are frozen

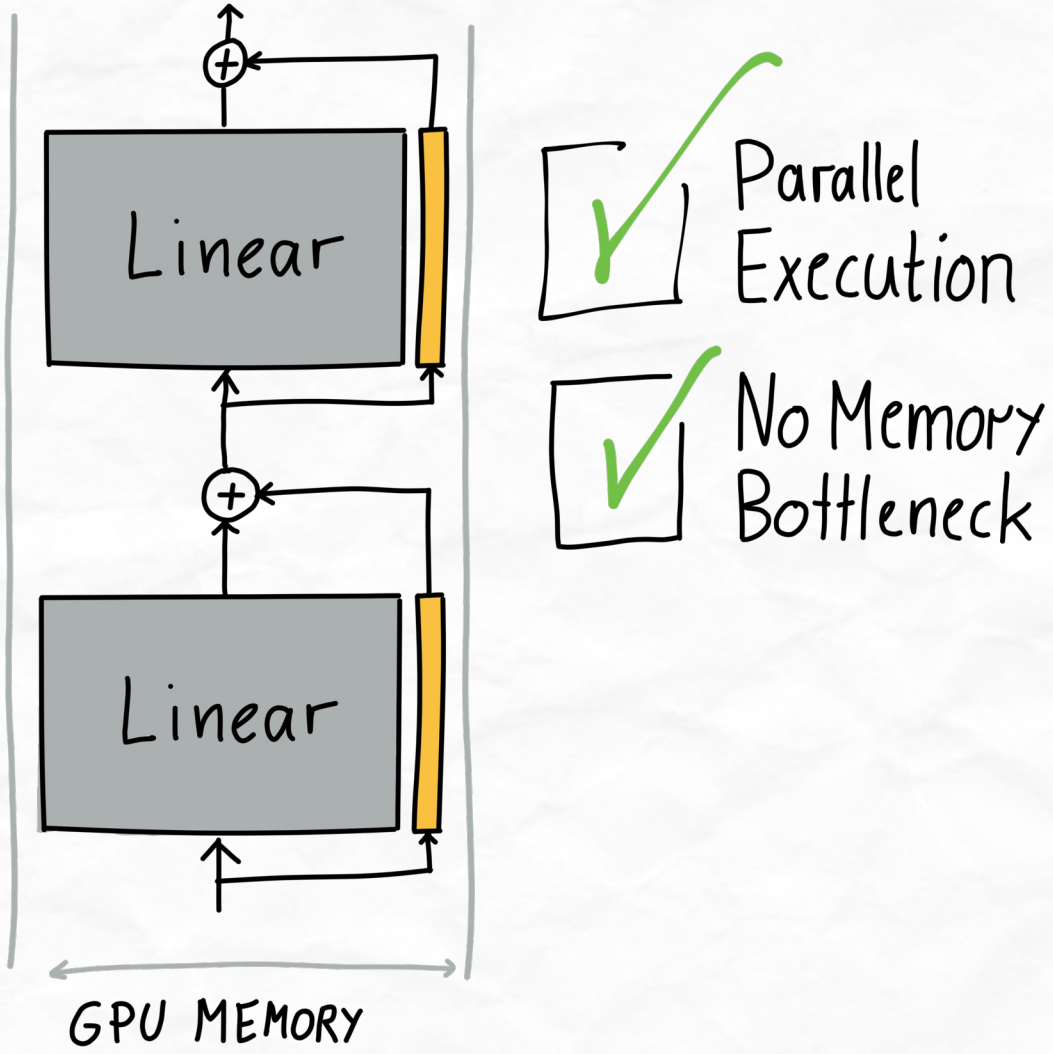
Adapter

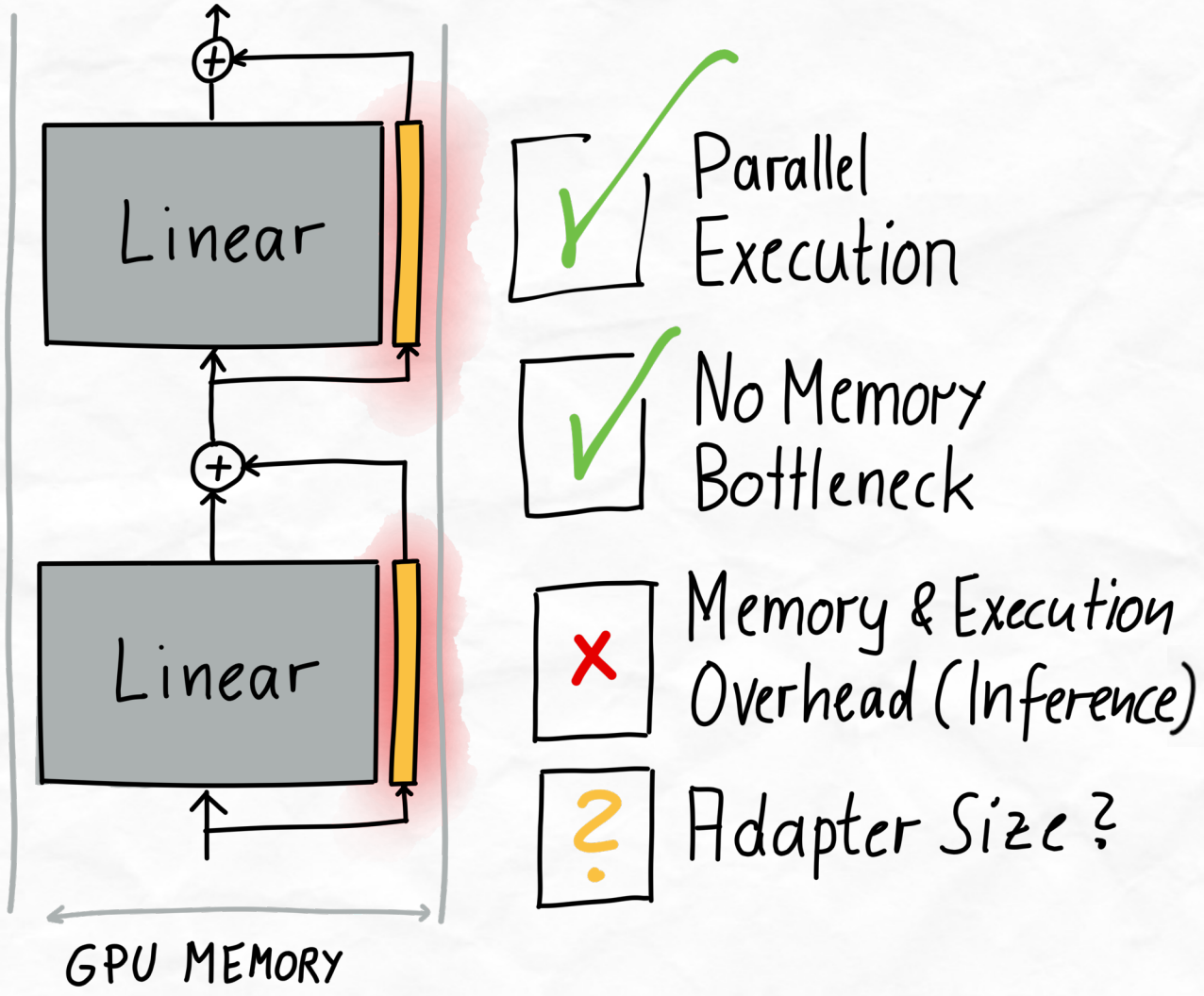


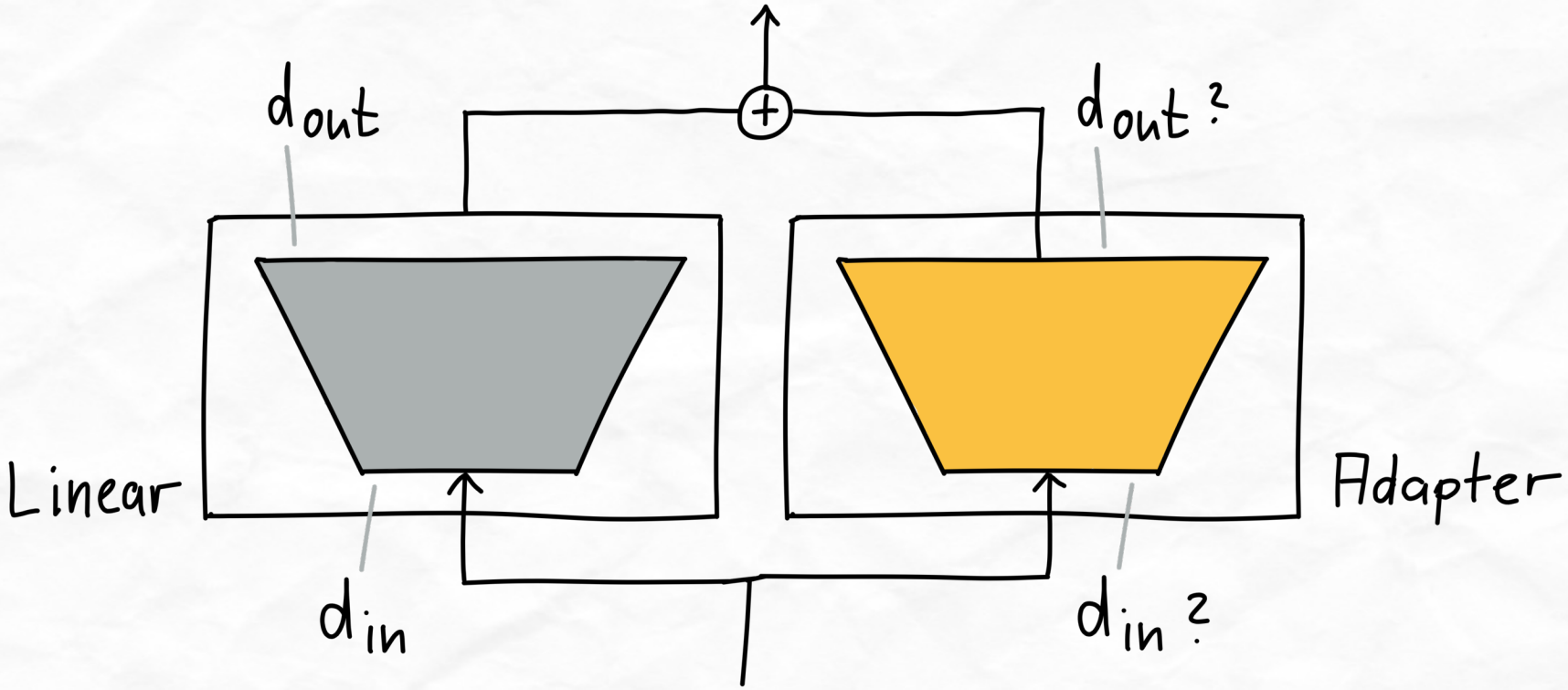


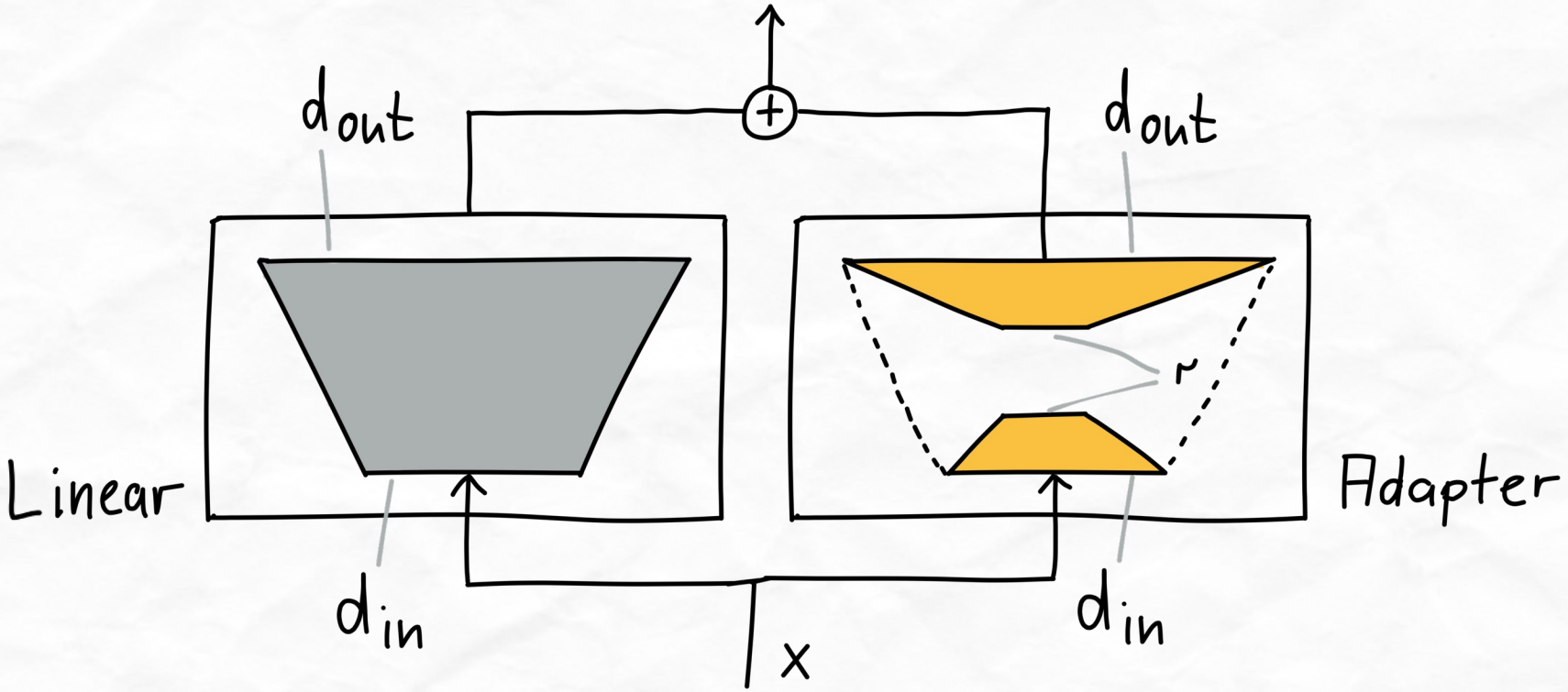




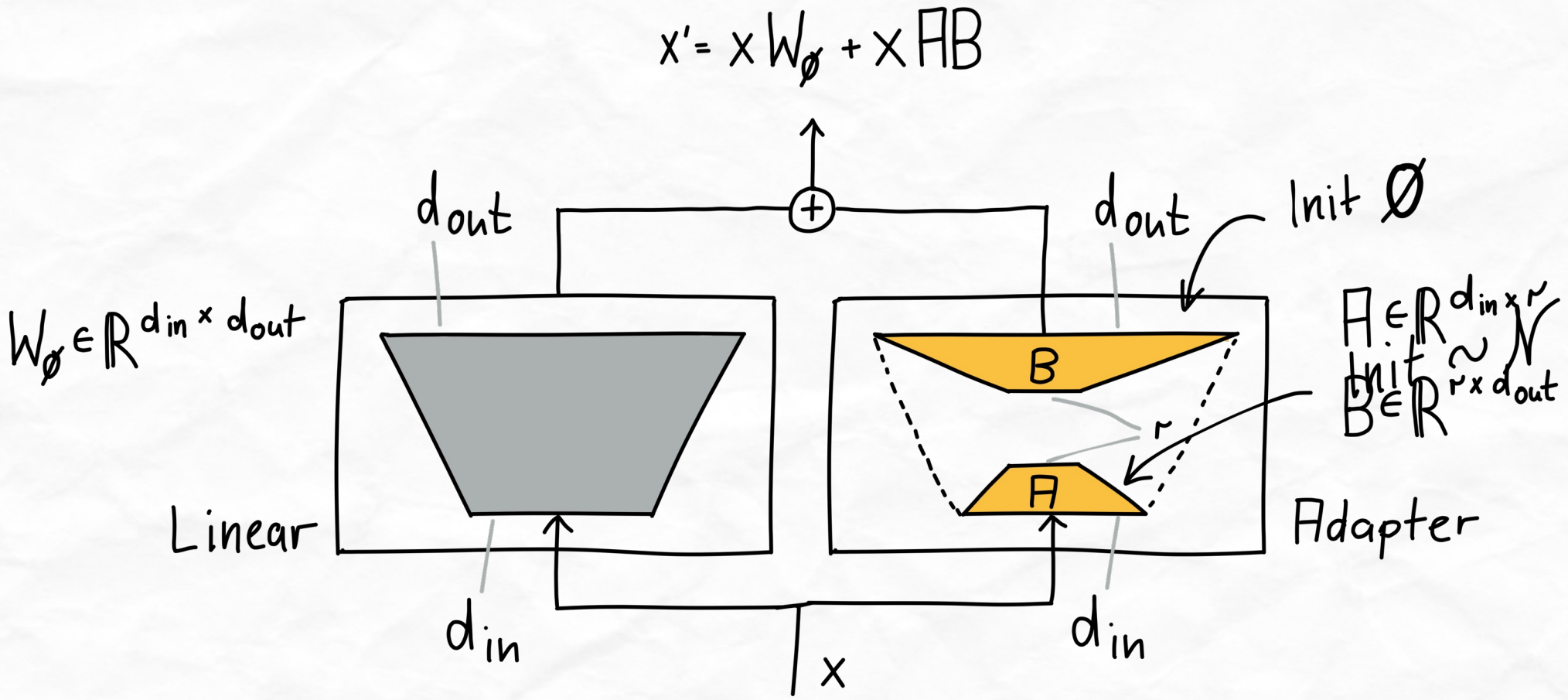








LoRA



Hu, Chen et al. 2021

```
class LoRAAdapter(nn.Module):
    def __init__(self, adaptee, r):
        super().__init__()

        self.adaptee = adaptee
        self.r = r

        self.orig_forward = adaptee.forward
        adaptee.forward = self.forward

        adaptee.lora_A = nn.Parameter(
            torch.randn(adaptee.in_features, r) / math.sqrt(adaptee.in_features)
        )
        adaptee.lora_B = nn.Parameter(torch.zeros(r, adaptee.out_features))

    def forward(self, x, *args, **kwargs):
        return (
            self.orig_forward(x, *args, **kwargs)
            + x @ self.adaptee.lora_A @ self.adaptee.lora_B
        )
```

```python
class LoRAAdapter(nn.Module):
    def __init__(self, adaptee, r):
        super().__init__()

        self.adaptee = adaptee
        self.r = r

        self.orig_forward = adaptee.forward
        adaptee.forward = self.forward

        adaptee.lora_A = nn.Parameter(
            torch.randn(adaptee.in_features, r) / math.sqrt(adaptee.in_features)
        )
        adaptee.lora_B = nn.Parameter(torch.zeros(r, adaptee.out_features))

    def forward(self, x, *args, **kwargs):
        return (
            self.orig_forward(x, *args, **kwargs)
            + x @ self.adaptee.lora_A @ self.adaptee.lora_B
        )
```

```
class LoRAAdapter(nn.Module):
    def __init__(self, adaptee, r):
        super().__init__()

        self.adaptee = adaptee
        self.r = r

        self.orig_forward = adaptee.forward
        adaptee.forward = self.forward

        adaptee.lora_A = nn.Parameter(
            torch.randn(adaptee.in_features, r) / math.sqrt(adaptee.in_features)
        )
        adaptee.lora_B = nn.Parameter(torch.zeros(r, adaptee.out_features))

    def forward(self, x, *args, **kwargs):
        return (
            self.orig_forward(x, *args, **kwargs)
            + x @ self.adaptee.lora_A @ self.adaptee.lora_B
        )
```

```
class LoRAAdapter(nn.Module):
    def __init__(self, adaptee, r):
        super().__init__()

        self.adaptee = adaptee
        self.r = r

        self.orig_forward = adaptee.forward
        adaptee.forward = self.forward

        adaptee.lora_A = nn.Parameter(
            torch.randn(adaptee.in_features, r) / math.sqrt(adaptee.in_features)
        )
        adaptee.lora_B = nn.Parameter(torch.zeros(r, adaptee.out_features))

    def forward(self, x, *args, **kwargs):
        return (
            self.orig_forward(x, *args, **kwargs)
            + x @ self.adaptee.lora_A @ self.adaptee.lora_B
        )
```

```
class LoRAAdapter(nn.Module):
    def __init__(self, adaptee, r):
        super().__init__()

        self.adaptee = adaptee
        self.r = r

        self.orig_forward = adaptee.forward
        adaptee.forward = self.forward

        adaptee.lora_A = nn.Parameter(
            torch.randn(adaptee.in_features, r) / math.sqrt(adaptee.in_features)
        )
        adaptee.lora_B = nn.Parameter(torch.zeros(r, adaptee.out_features))

    def forward(self, x, *args, **kwargs):
        return (
            self.orig_forward(x, *args, **kwargs)
            + x @ self.adaptee.lora_A @ self.adaptee.lora_B
        )
```

```
class LoRAAdapter(nn.Module):
    def __init__(self, adaptee, r):
        super().__init__()

        self.adaptee = adaptee
        self.r = r

        self.orig_forward = adaptee.forward
        adaptee.forward = self.forward

        adaptee.lora_A = nn.Parameter(
            torch.randn(adaptee.in_features, r) / math.sqrt(adaptee.in_features)
        )
        adaptee.lora_B = nn.Parameter(torch.zeros(r, adaptee.out_features))

    def forward(self, x, *args, **kwargs):
        return (
            self.orig_forward(x, *args, **kwargs)
            + x @ self.adaptee.lora_A @ self.adaptee.lora_B
        )
```

```
class LoRAAdapter(nn.Module):
    def __init__(self, adaptee, r):
        super().__init__()

        self.adaptee = adaptee
        self.r = r

        self.orig_forward = adaptee.forward
        adaptee.forward = self.forward

        adaptee.lora_A = nn.Parameter(
            torch.randn(adaptee.in_features, r) / math.sqrt(adaptee.in_features)
        )
        adaptee.lora_B = nn.Parameter(torch.zeros(r, adaptee.out_features))

    def forward(self, x, *args, **kwargs):
        return (
            self.orig_forward(x, *args, **kwargs)
            + x @ self.adaptee.lora_A @ self.adaptee.lora_B
        )
```

QLoRA:
LoRA + Quantization

```
model = AutoModelForSequenceClassification.from_pretrained(
    ”tiiuae/falcon-7b-instruct”,
    num_labels=2,
    load_in_4bit=True
)

model = prepare_model_for_kbit_training(model)

peft_config = LoraConfig(
    task_type=TaskType.SEQ_CLS,
    r=args.lora_r,
    target_modules=["query_key_value","dense_h_to_4h","dense_4h_to_h"]
)

model = get_peft_model(model, peft_config)
```

QLoRA:
LoRA + Quantization

```
model = AutoModelForSequenceClassification.from_pretrained(
    "tiiuae/falcon-7b-instruct",
    num_labels=2,
    load_in_4bit=True
)

model = prepare_model_for_kbit_training(model)

peft_config = LoraConfig(
    task_type=TaskType.SEQ_CLS,
    r=args.lora_r,
    target_modules=["query_key_value","dense_h_to_4h","dense_4h_to_h"]
)

model = get_peft_model(model, peft_config)
```

QLoRA:
LoRA + Quantization

```
model = AutoModelForSequenceClassification.from_pretrained(
    "tiiuae/falcon-7b-instruct",
    num_labels=2,
    load_in_4bit=True
)

model = prepare_model_for_kbit_training(model)

peft_config = LoraConfig(
    task_type=TaskType.SEQ_CLS,
    r=args.lora_r,
    target_modules=["query_key_value","dense_h_to_4h","dense_4h_to_h"]
)

model = get_peft_model(model, peft_config)
```

QLoRA:
LoRA + Quantization

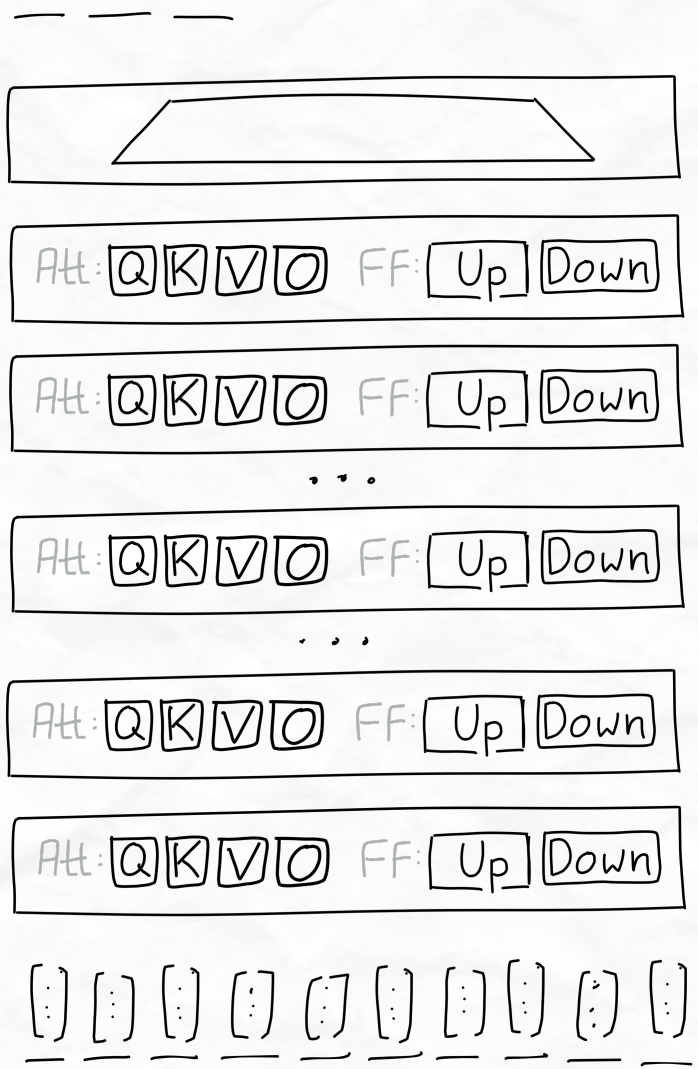
```
model = AutoModelForSequenceClassification.from_pretrained(
    "tiiuae/falcon-7b-instruct",
    num_labels=2,
    load_in_4bit=True
)

model = prepare_model_for_kbit_training(model)

peft_config = LoraConfig(
    task_type=TaskType.SEQ_CLS,
    r=8,
    target_modules=["query_key_value","dense_h_to_4h","dense_4h_to_h"]
)

model = get_peft_model(model, peft_config)
```

QLoRA:
LoRA + Quantization

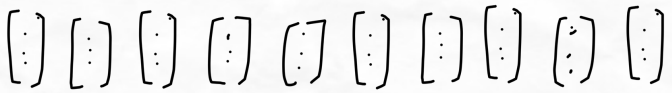
```
model = AutoModelForSequenceClassification.from_pretrained(
    "tiiuae/falcon-7b-instruct",
    num_labels=2,
    load_in_4bit=True
)

model = prepare_model_for_kbit_training(model)

peft_config = LoraConfig(
    task_type=TaskType.SEQ_CLS,
    r=8,
    target_modules=["query_key_value","dense_h_to_4h","dense_4h_to_h"]
)

model = get_peft_model(model, peft_config)
```

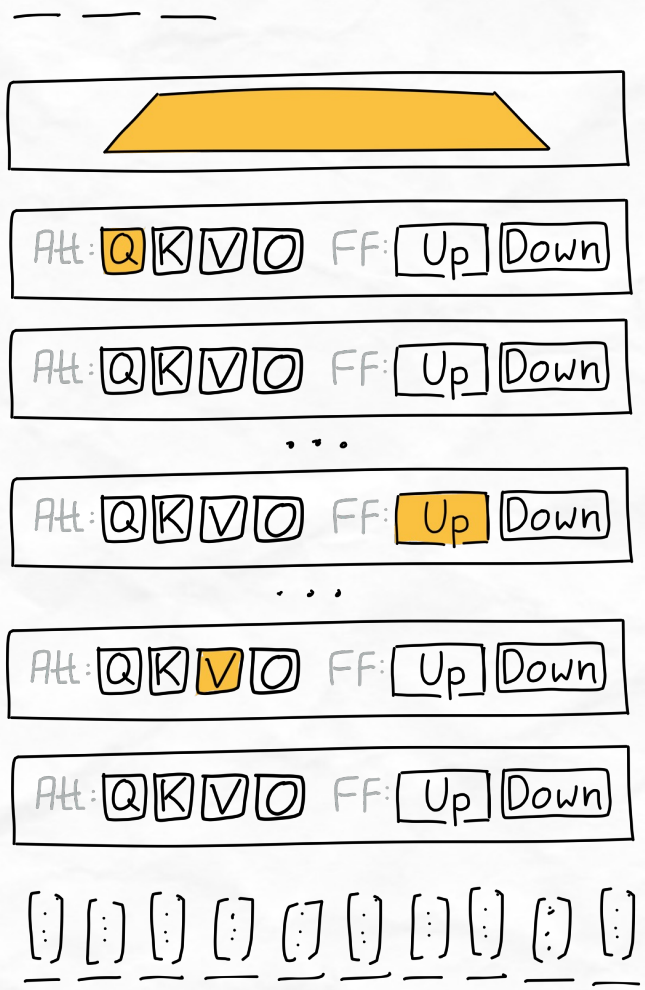
QLoRA:
LoRA + Quantization

WHERE TO APPLY?

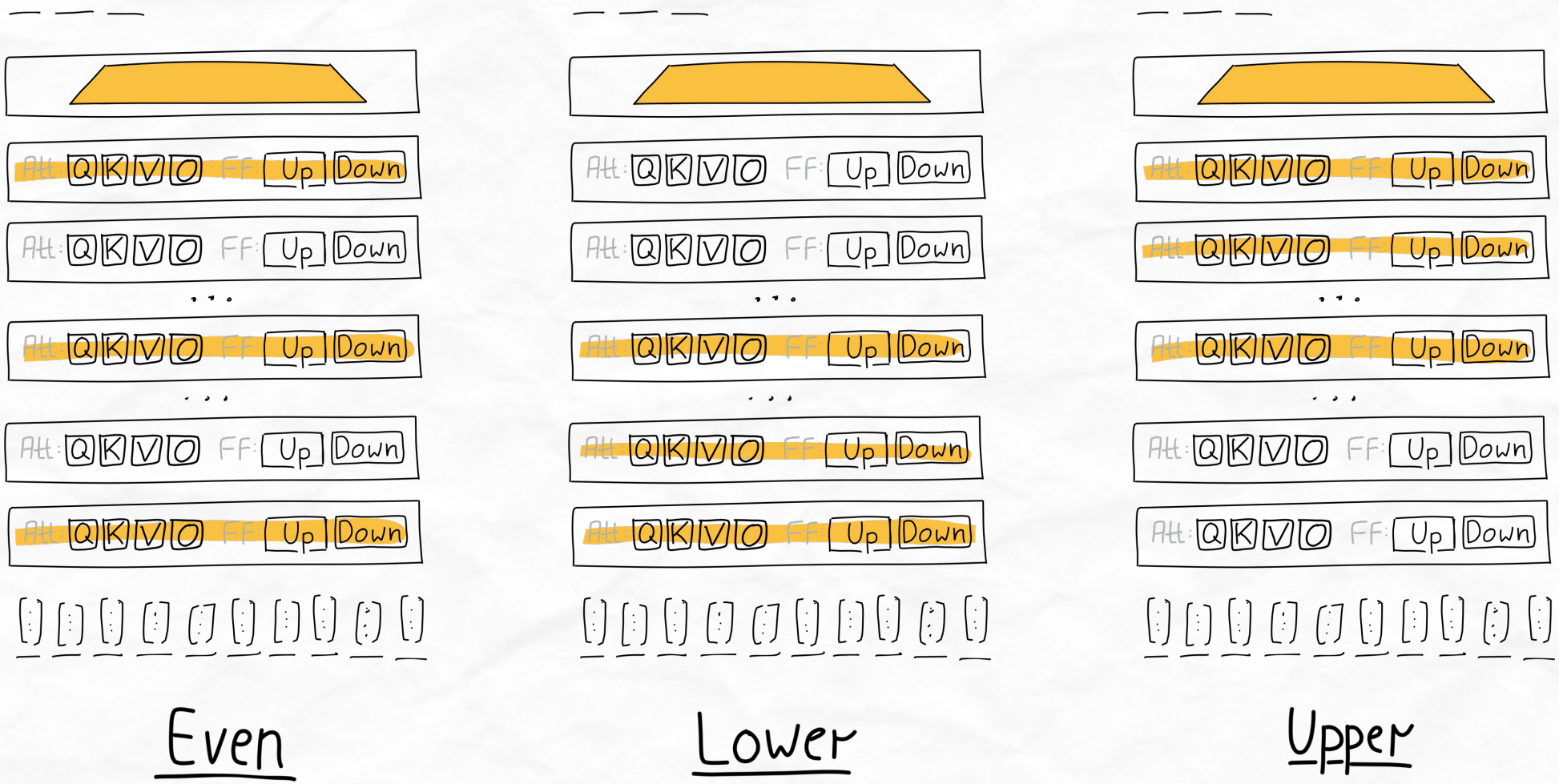


WHERE TO APPLY?

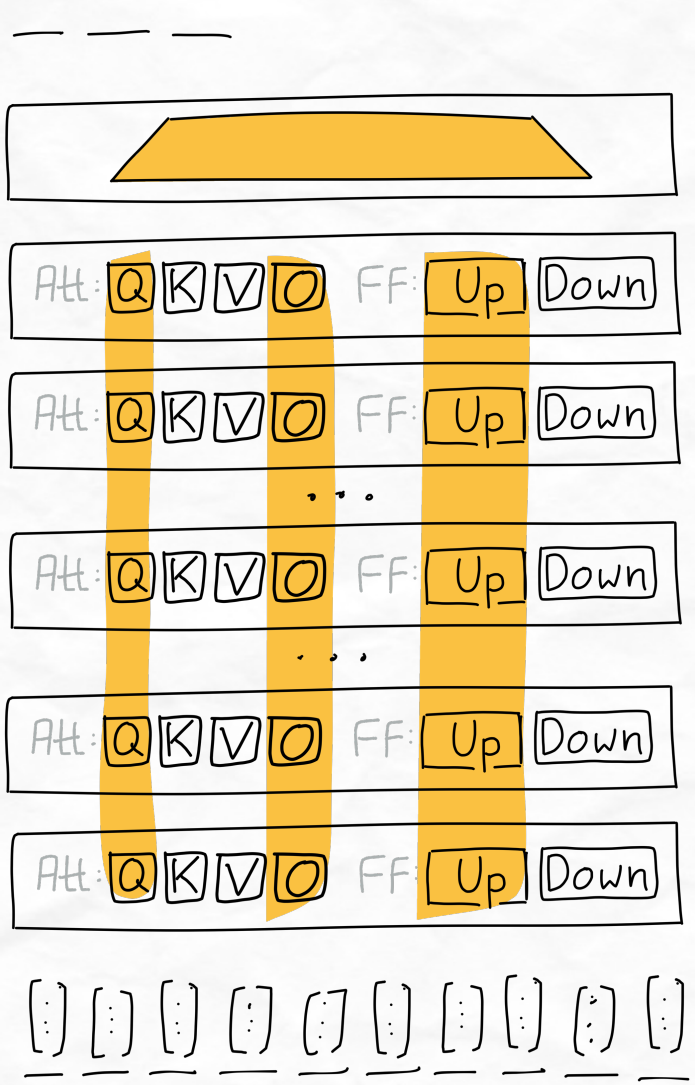


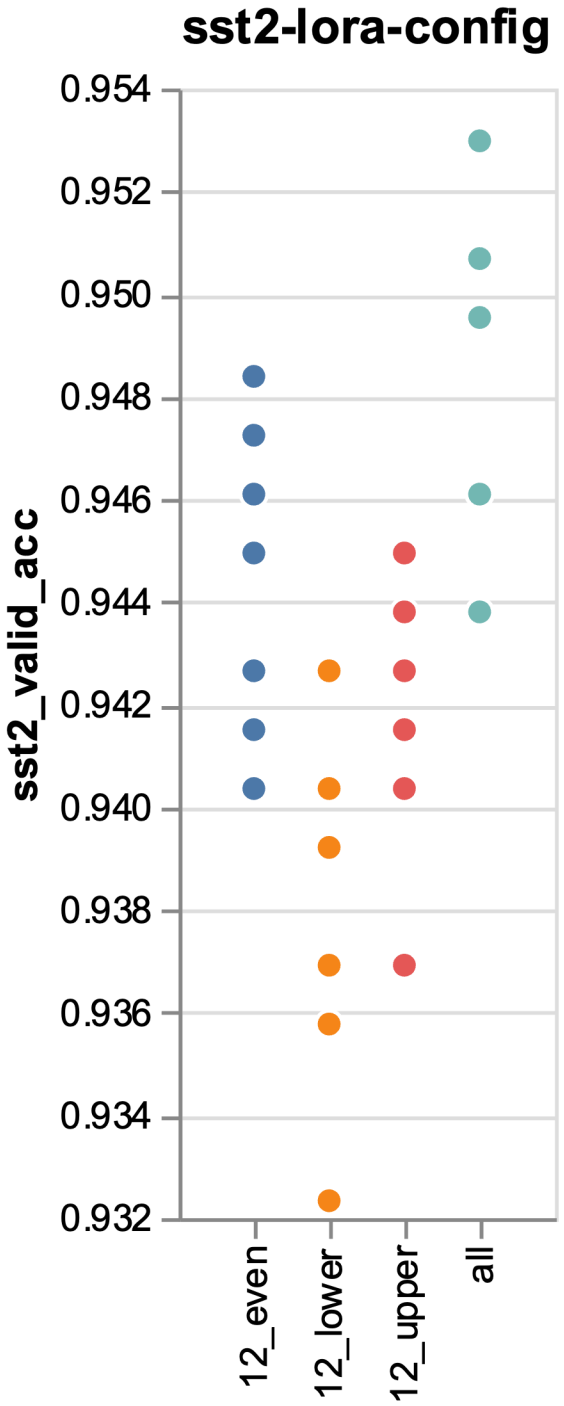


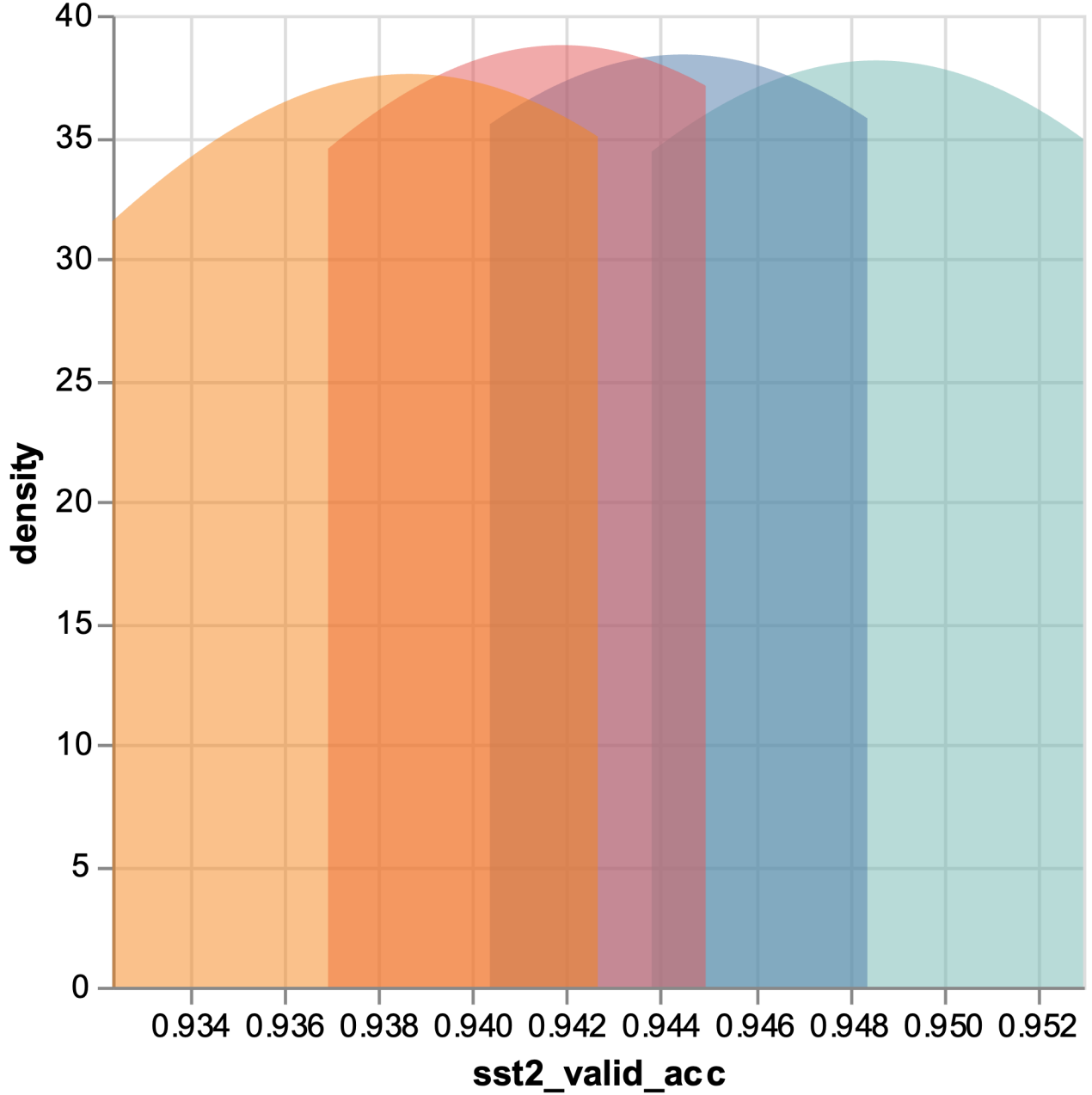
BY LAYERS

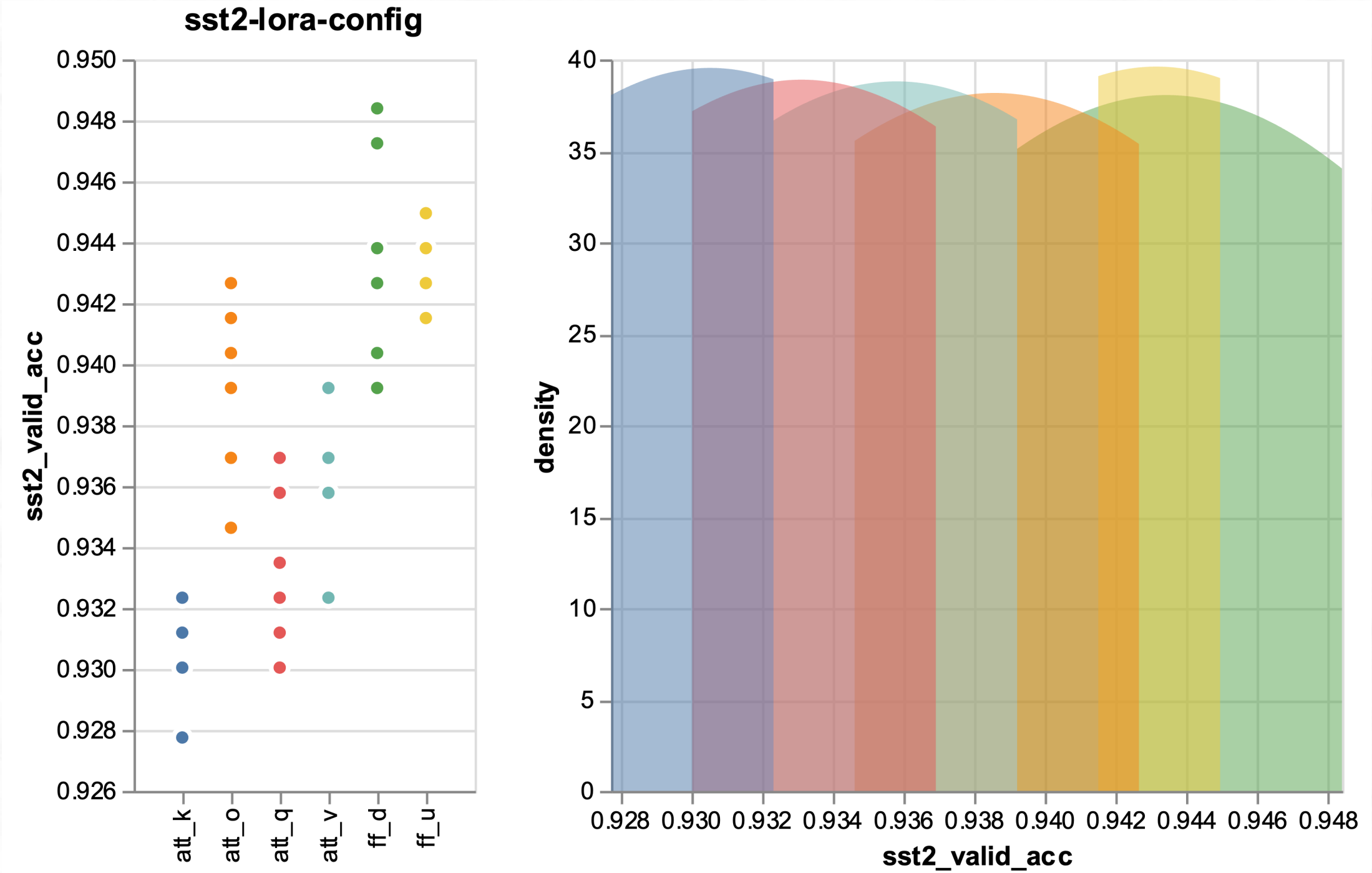


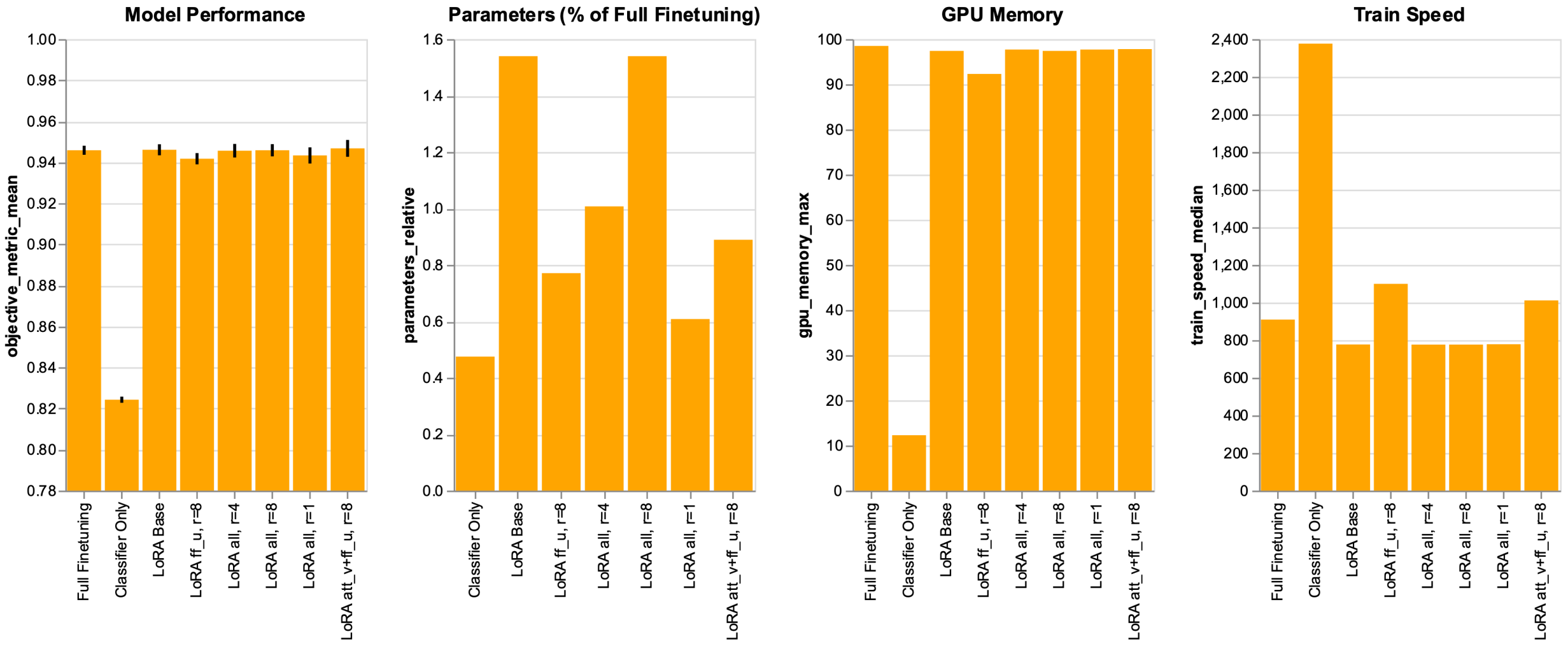
BY COMPONENT



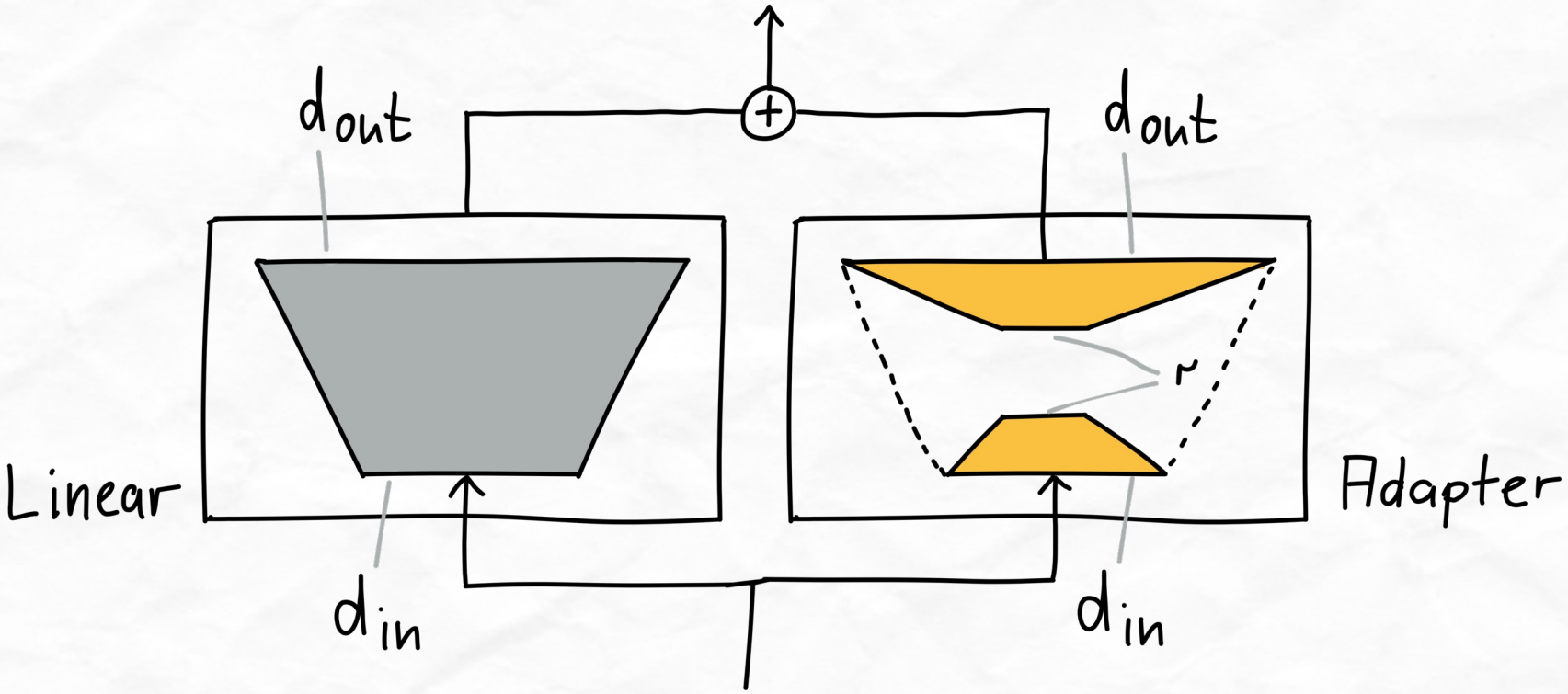

sst2-lora-config
sst2_valid_acc
0.954
0.952
0.950
0.948
0.946
0.944
0.942
0.940
0.938
0.936
0.934
0.932
12_even
12_lower
12_upper
all

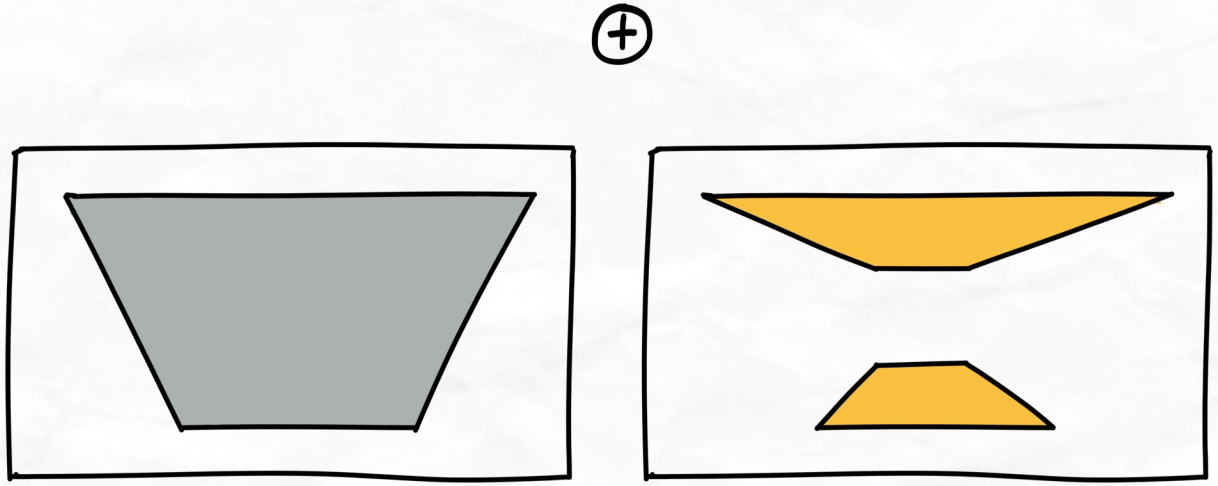

density
40
35
30
25
20
15
10
5
0
0.934
0.936
0.938
0.940
0.942
0.944
0.946
0.948
0.950
0.952
sst2_valid_acc

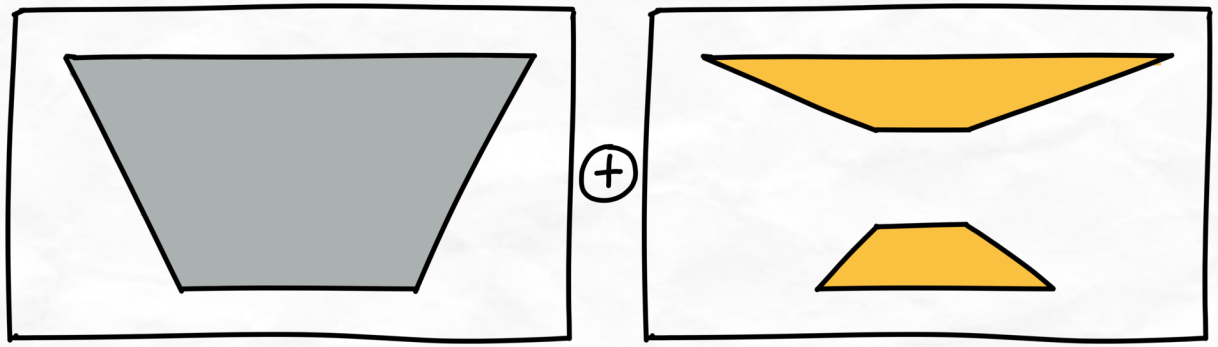

sst2-lora-config
sst2_valid_acc
0.950
0.948
0.946
0.944
0.942
0.940
0.938
0.936
0.934
0.932
0.930
0.928
0.926
att_k
att_o
att_q
att_v
ff_d
ff_u
density
40
35
30
25
20
15
10
5
0
0.928
0.930
0.932
0.934
0.936
0.938
0.940
0.942
0.944
0.946
0.948
sst2_valid_acc

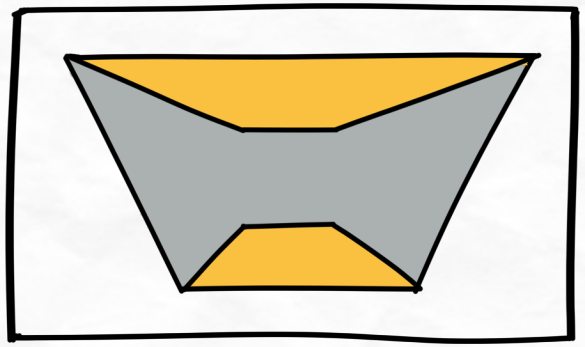


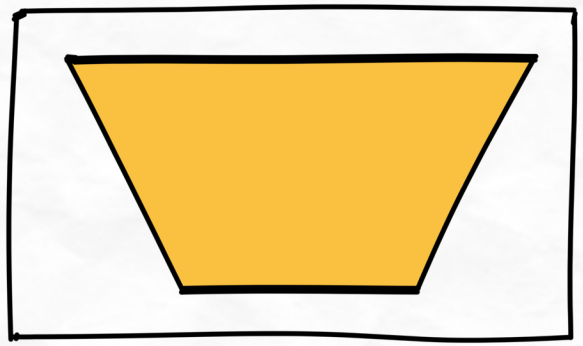
INFERENCE

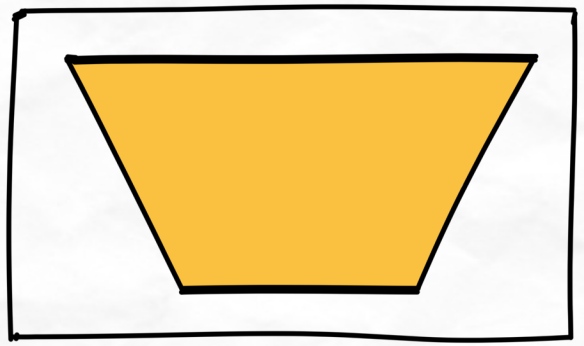








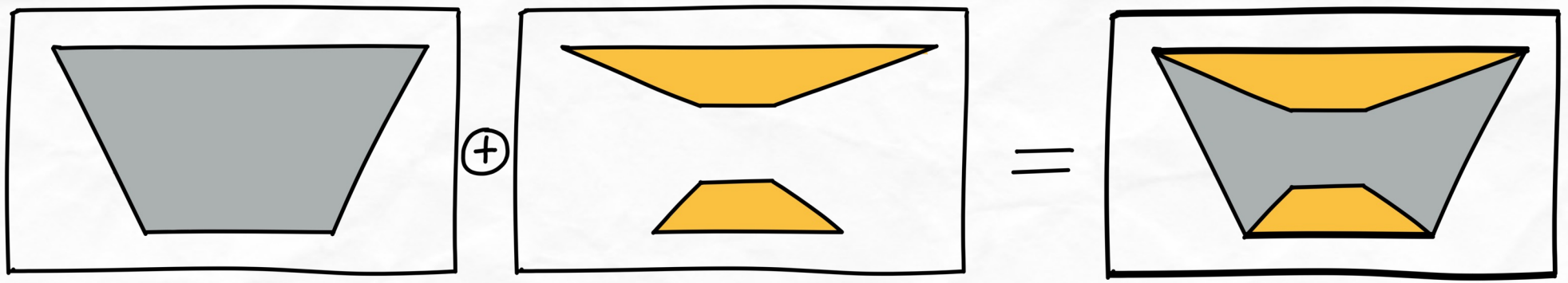




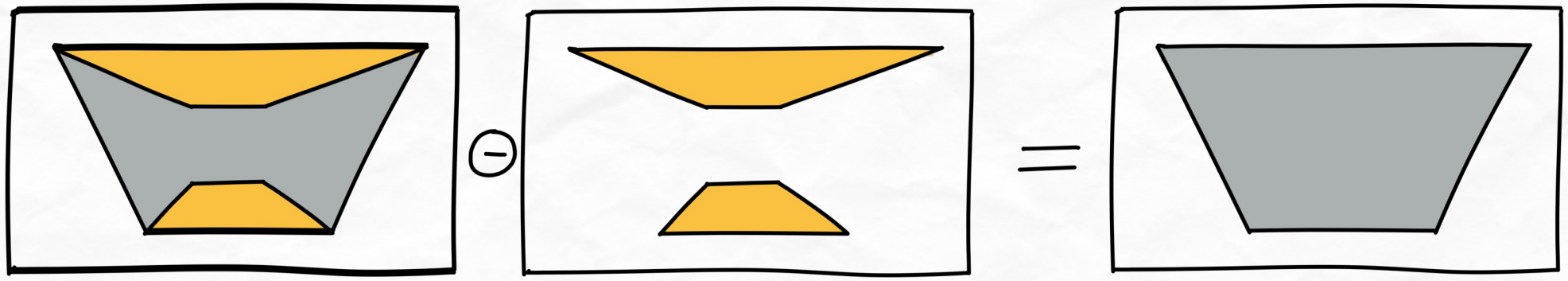
Baking in the product of A and B:

- [x] No special handling at inference
- [x] No overhead in compute or memory
- [x] Small footprint of finetuned weights

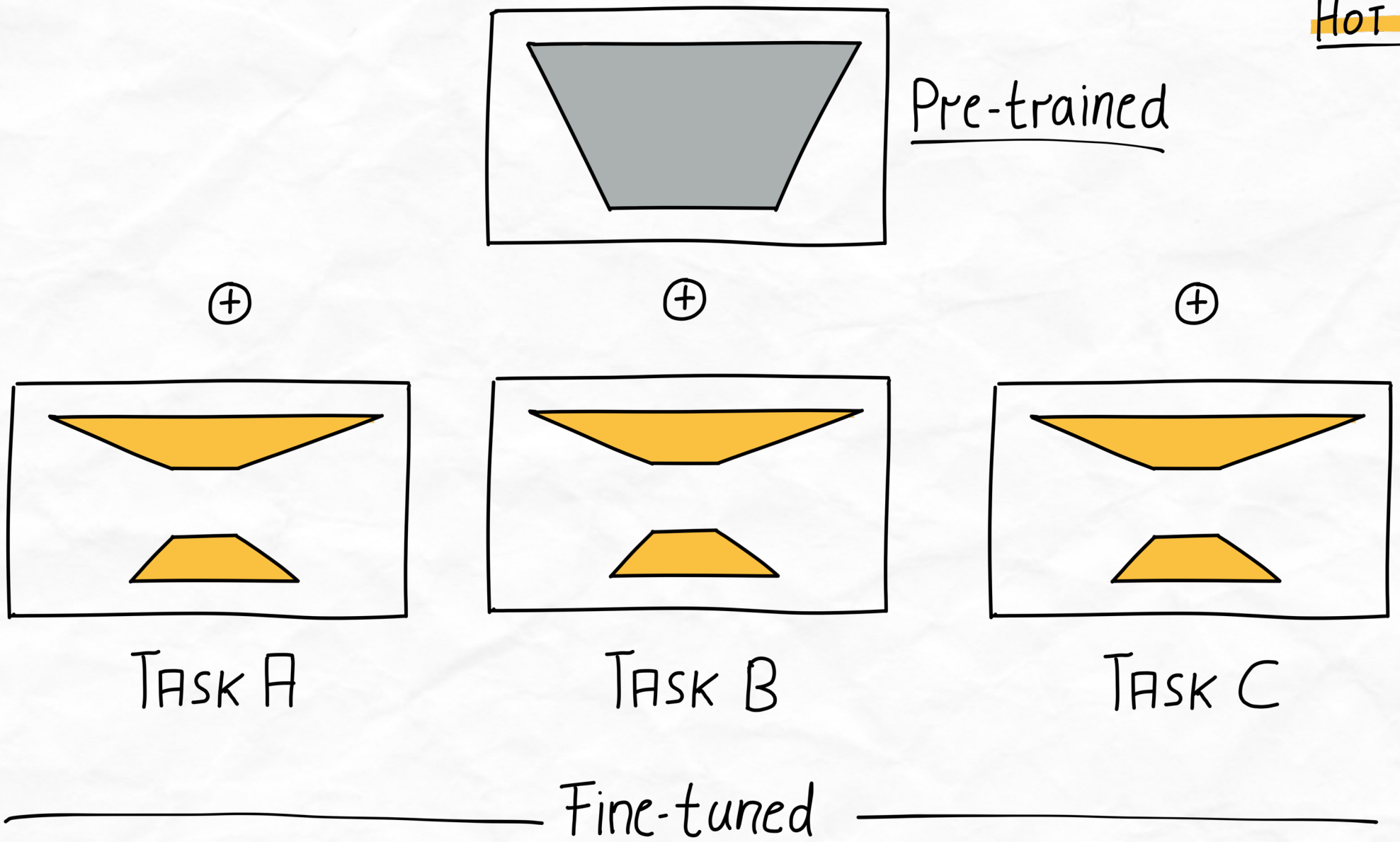
Adding Adapters



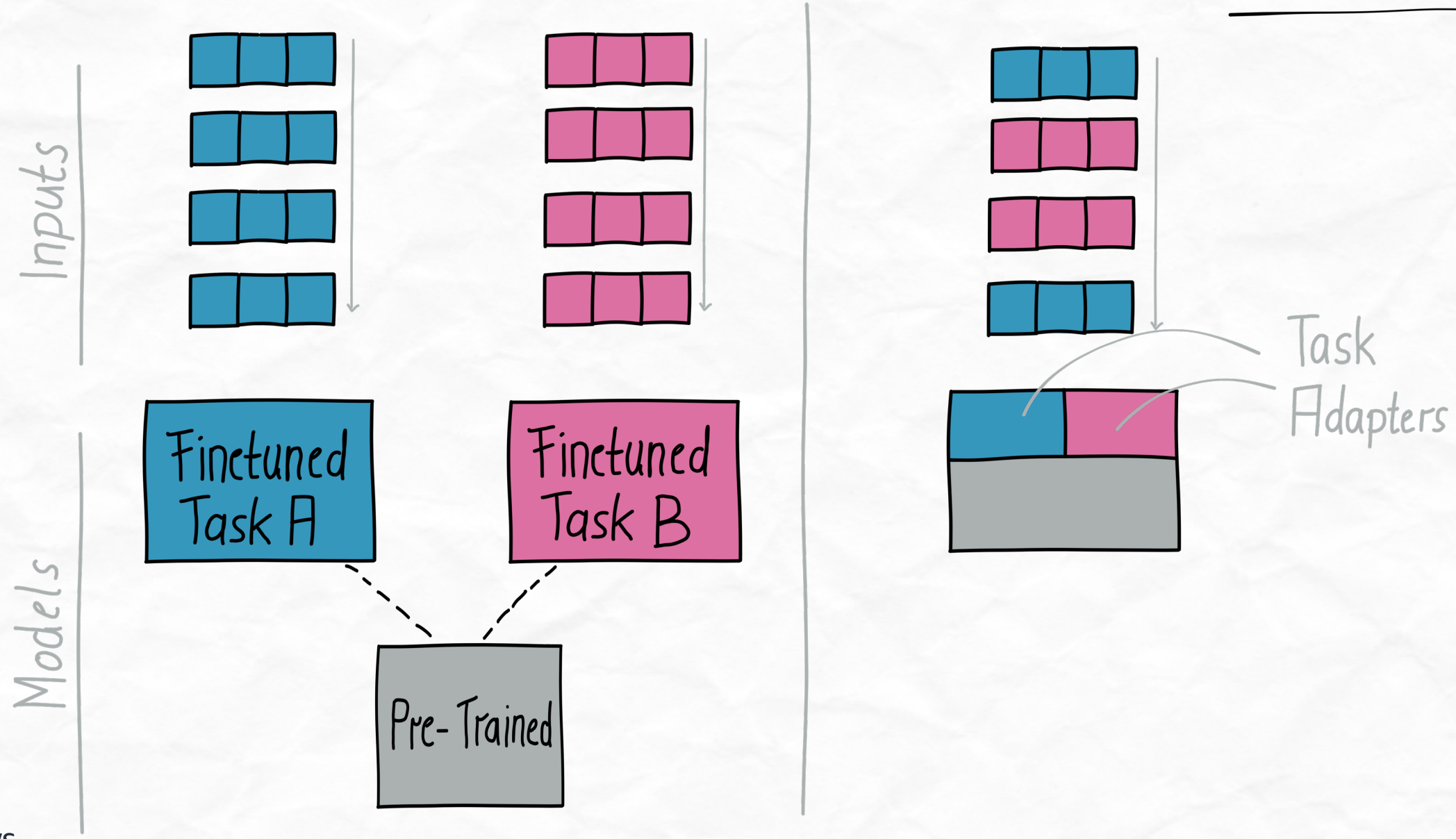
Removing Adapters



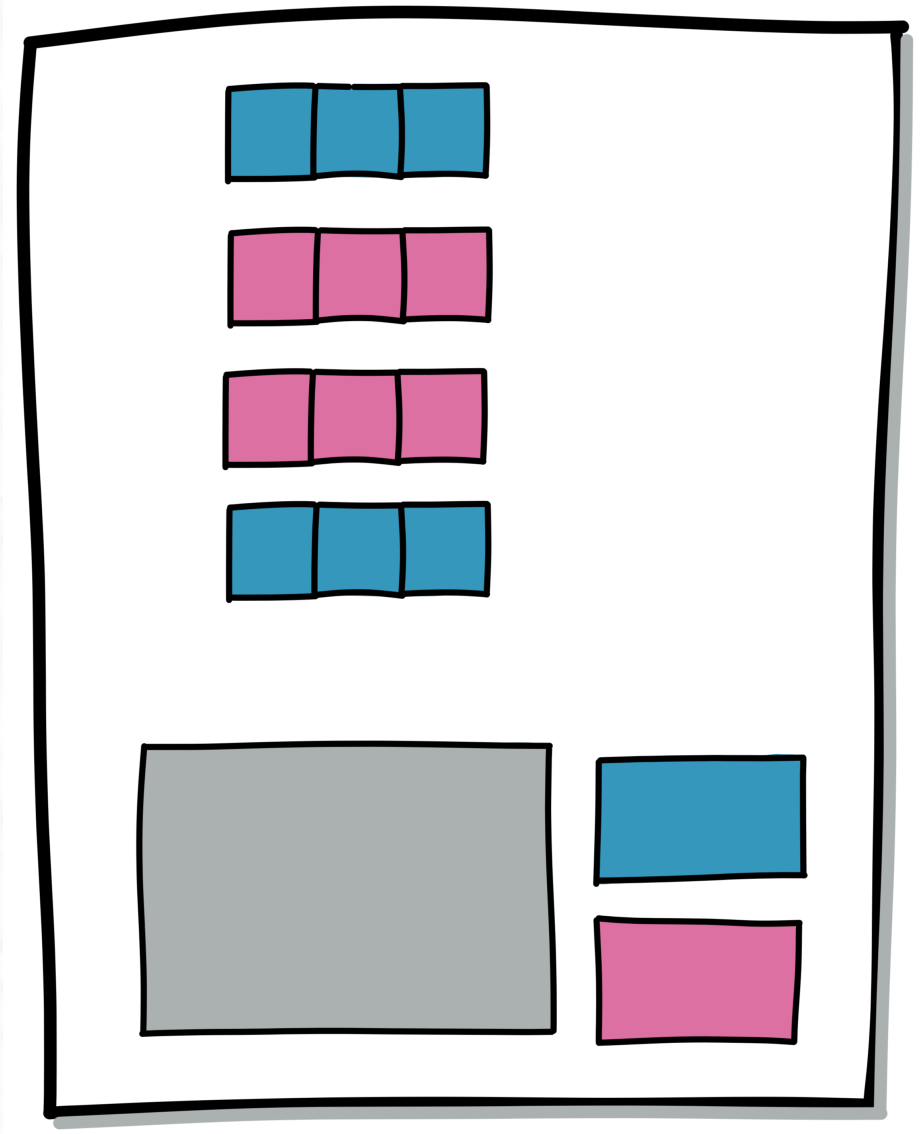
Hot Swapping

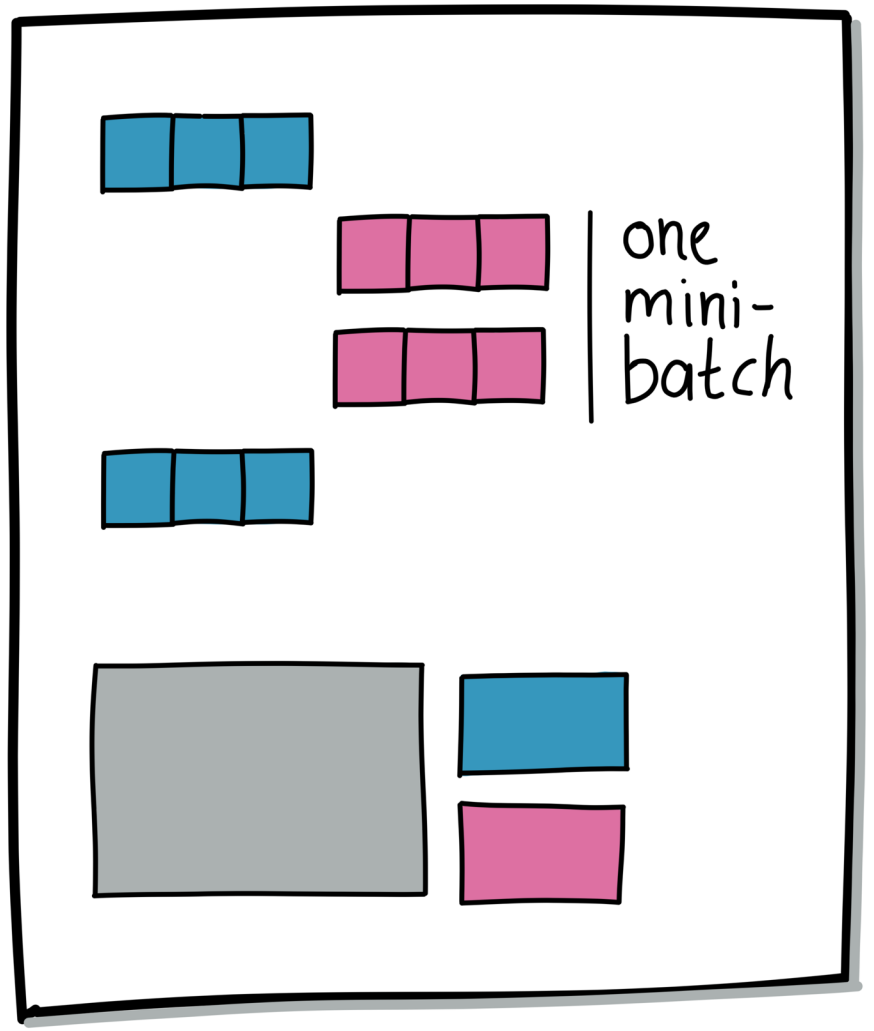


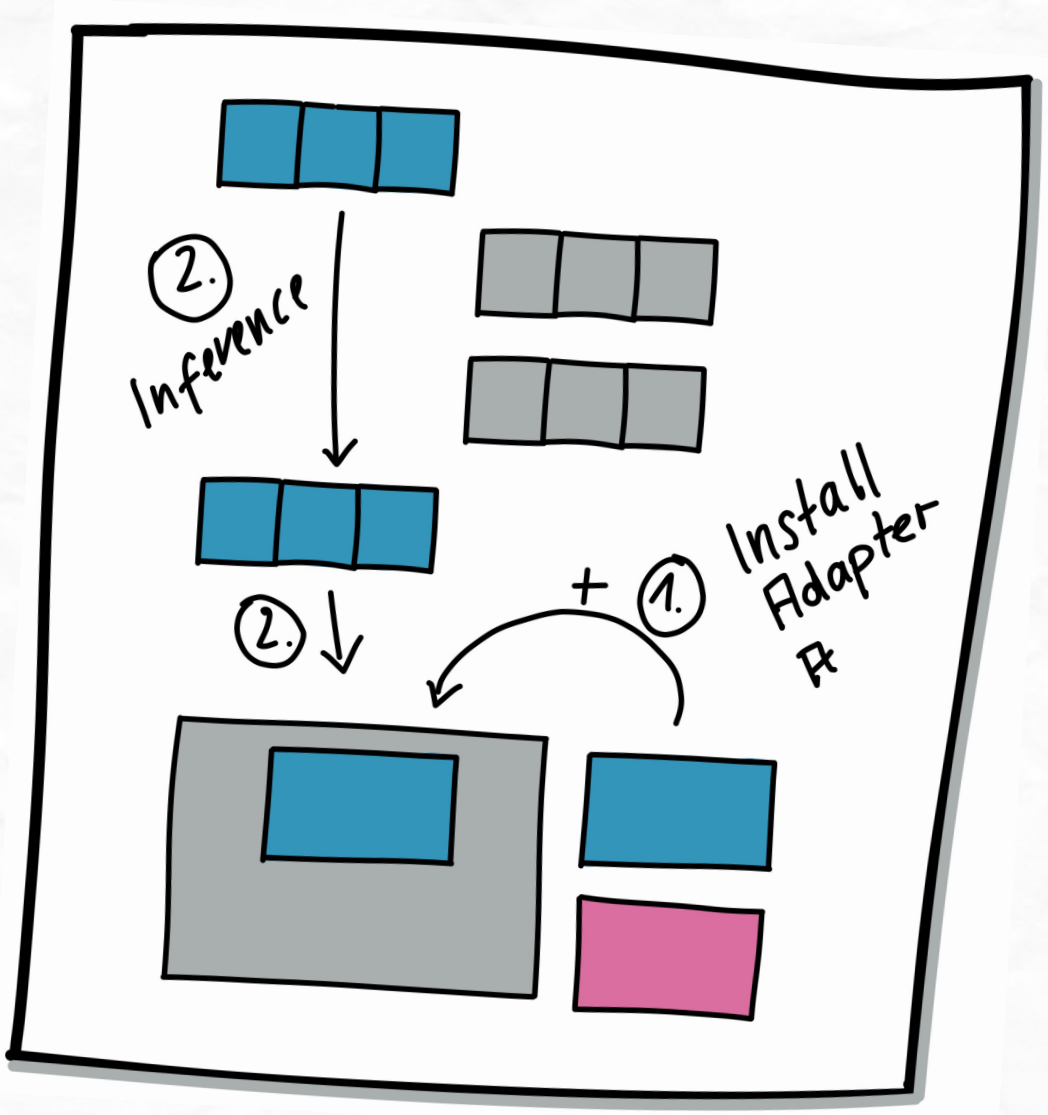
MULTI-TASK

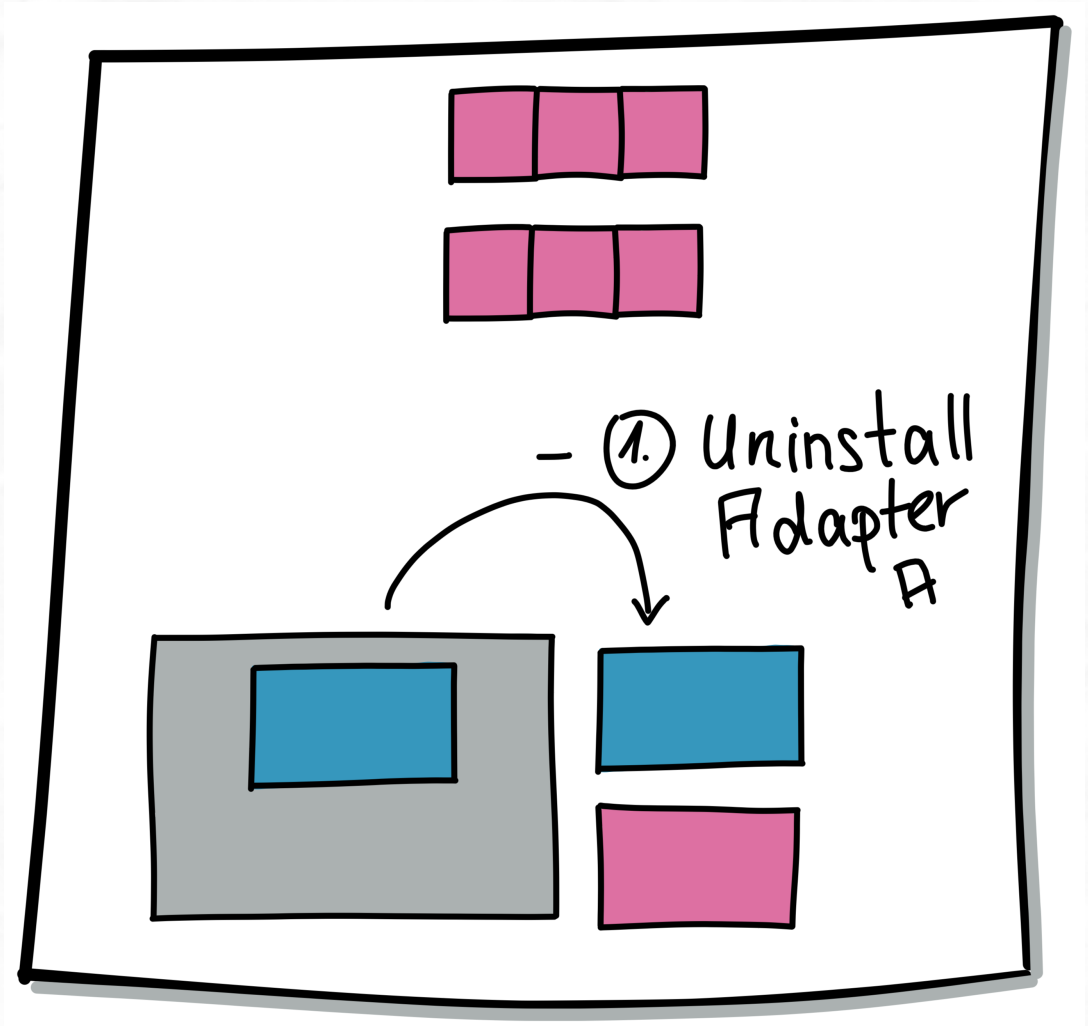


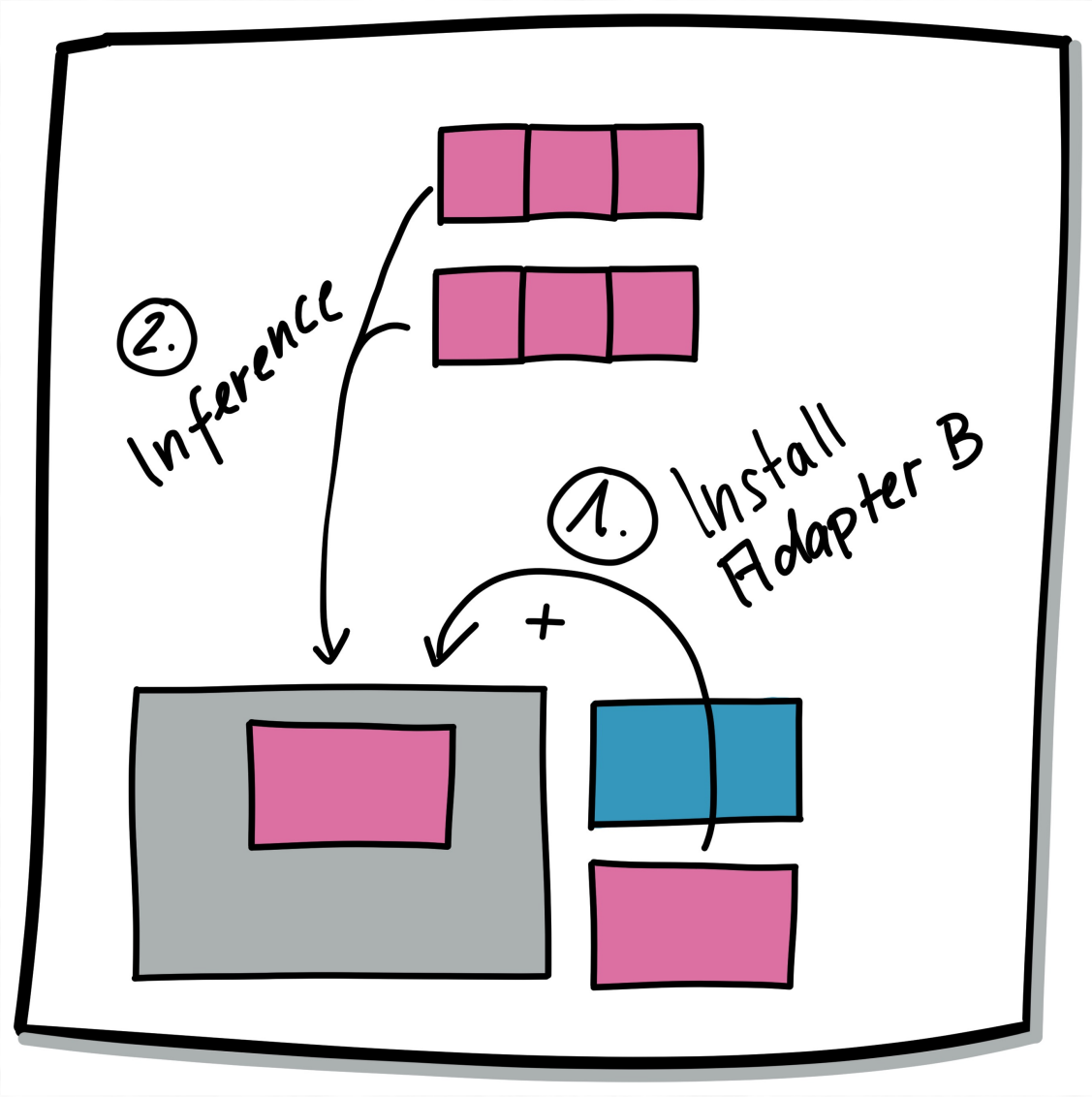
MULTI-TASK



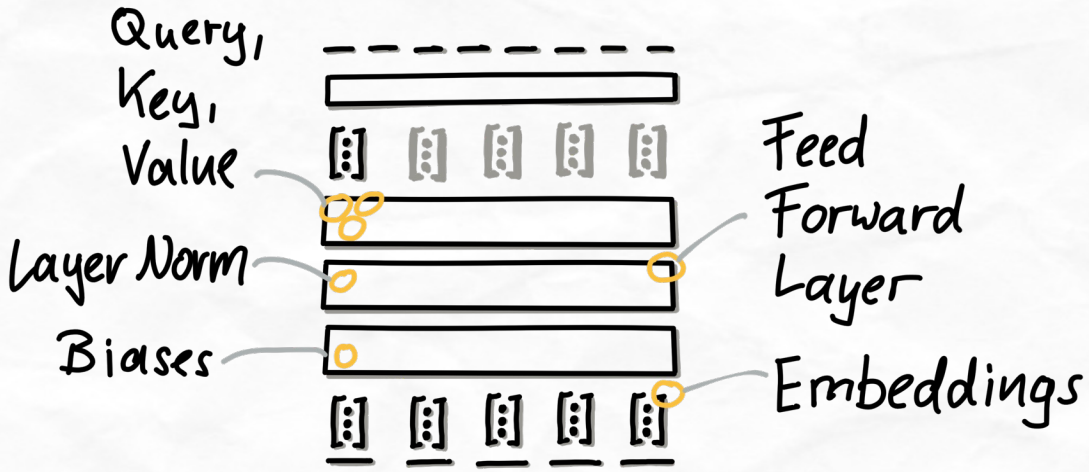




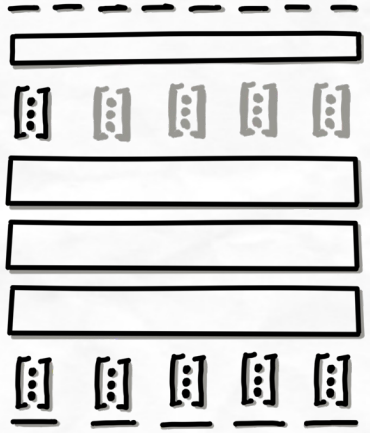




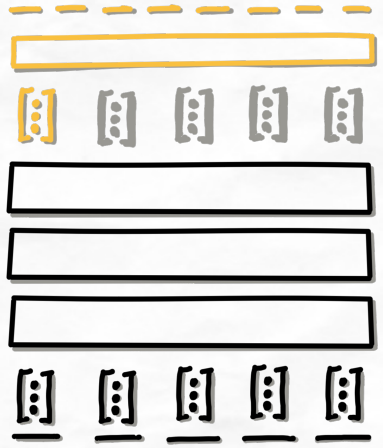
PEFT Approaches



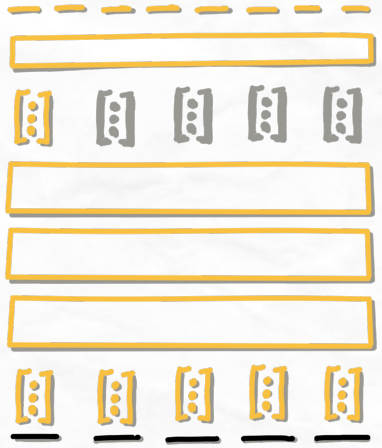
PEFT Approaches



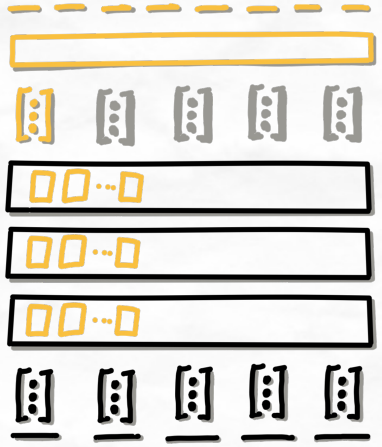
PEFT Approaches



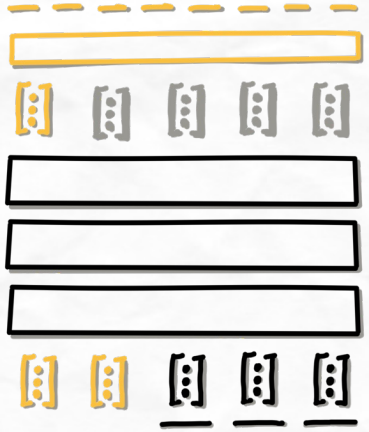
Classifier Only



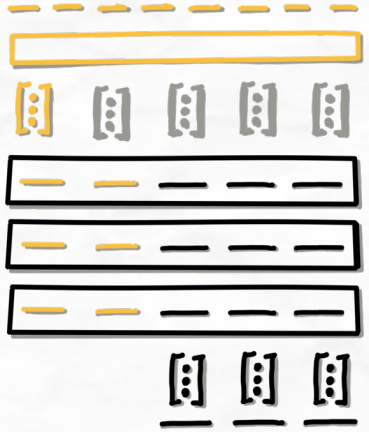
Full Finetuning



LoRA Finetuning

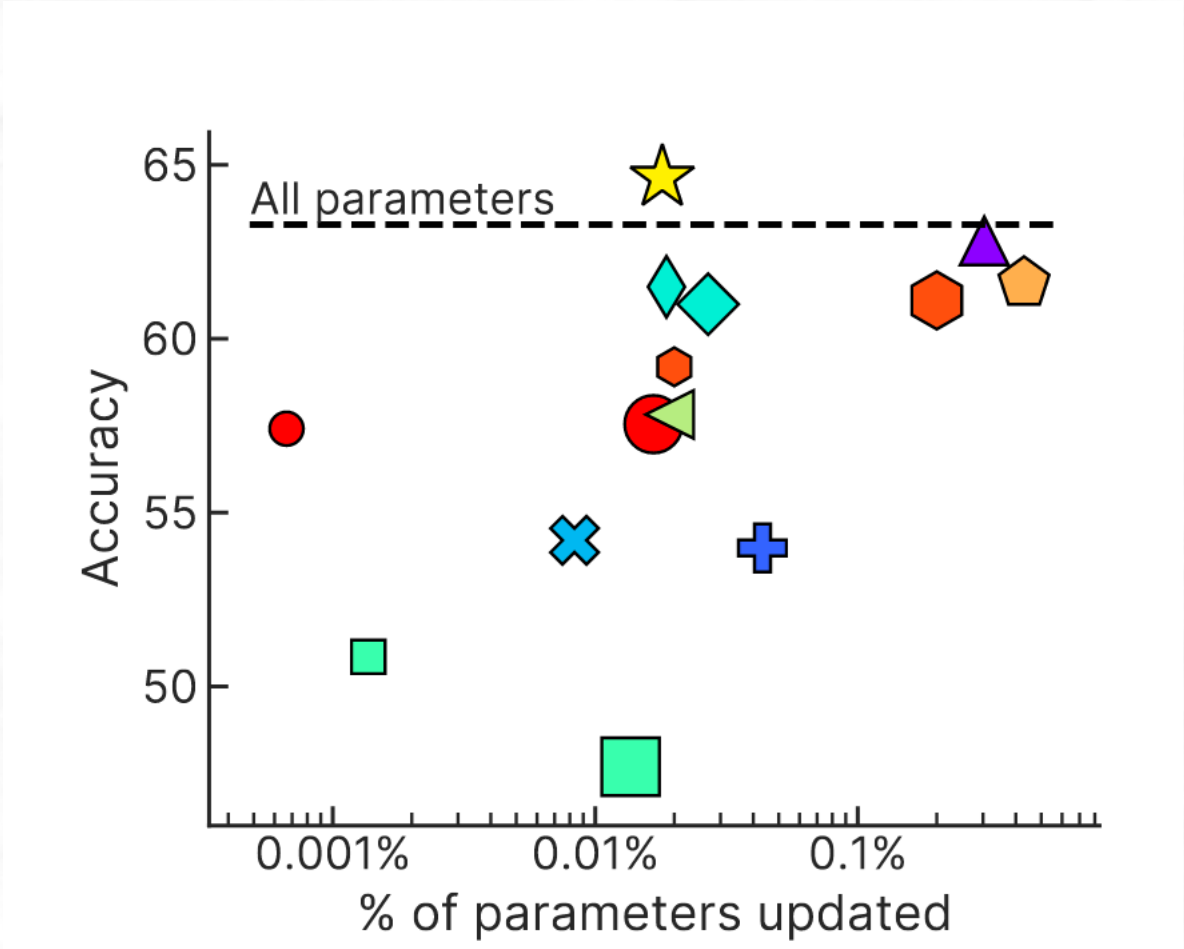


Soft Prompt Tuning



Prefix Tuning

PEFT Approaches



Liu, Tam, Muqeeth et al. 2022

Ressources

Dive Into LoRA Adapters: Intuitively Understanding Finetuning Using LoRA

Fine-tune LLaMA 2 (7-70B) on Amazon SageMaker

Hugging Face PEFT documentation

Please complete the session survey.

Thank you!

Mariano Kamp

https://www.linkedin.com/in/mariano-kamp/