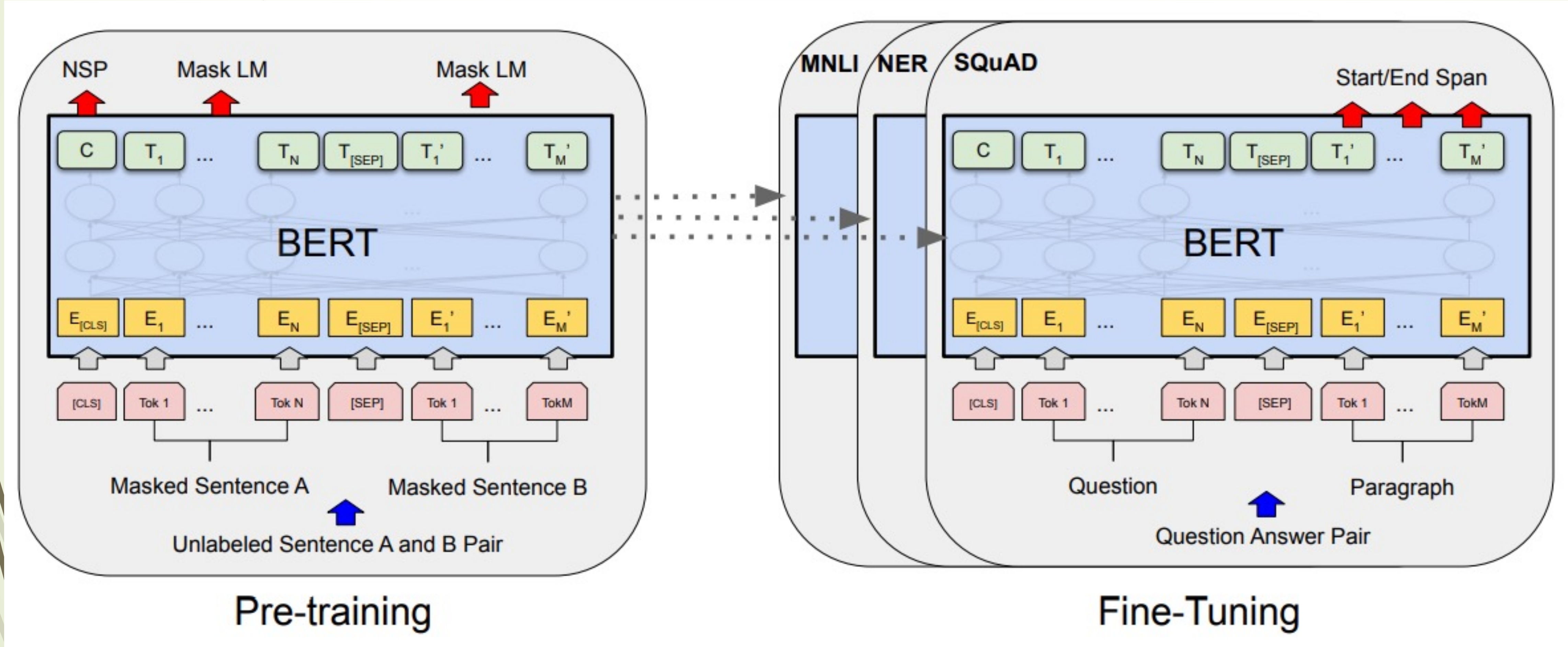
LORA and **QLORA**: Parameter-Efficient Fine-tuning of LLMs

Low-Rank Adaption

PEFT

Quantized LLMs

Pretraining and Fine-tuning



Language understanding

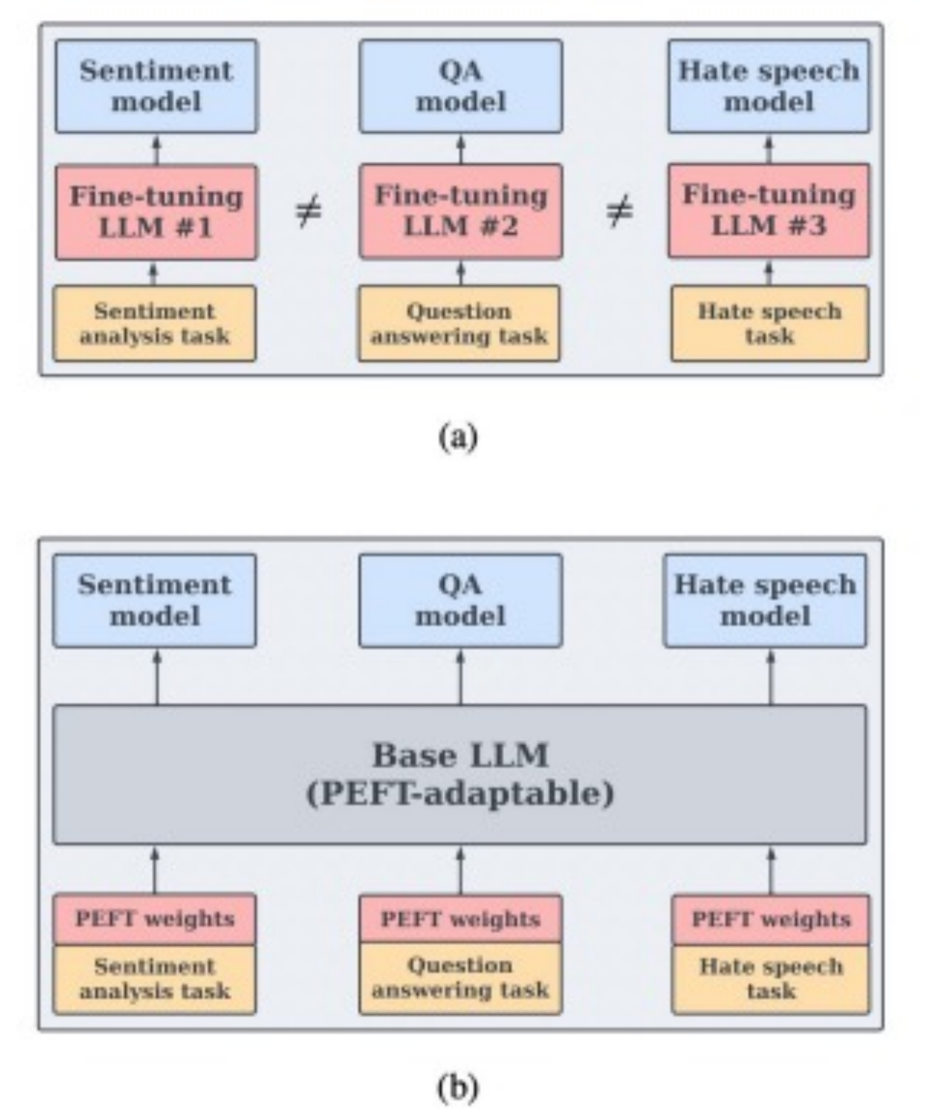
Adapting to different tasks

Pretraining v.s. Fine-tuning

Stage	Pretraining	Supervised Fine-tuning
Algorithm	Language modeling predict the next token	
Dataset	Raw internet text ~trillions of words low-quality, large quantity	Carefully curated text ~10-100K (prompt, response) low quantity, high quality
Resource	**1000s of GPUs months of training** ex: GPT LLaMA, PaLM	**1-100 GPUs days of training** ex: Vicuna-13B

Parameter-Efficient Fine-tuning (PEFT):

- A class of methods that adapt LLMs by updating only a small subset of model parameters.



PEFT Overview

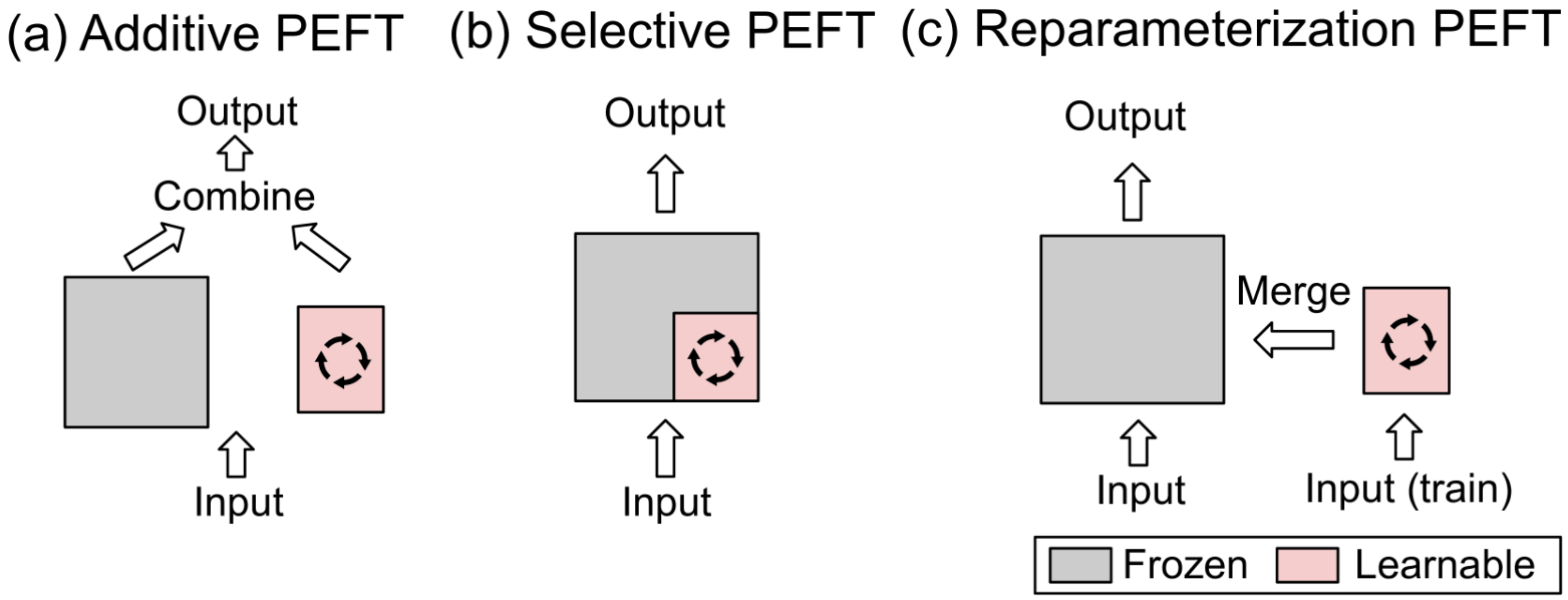
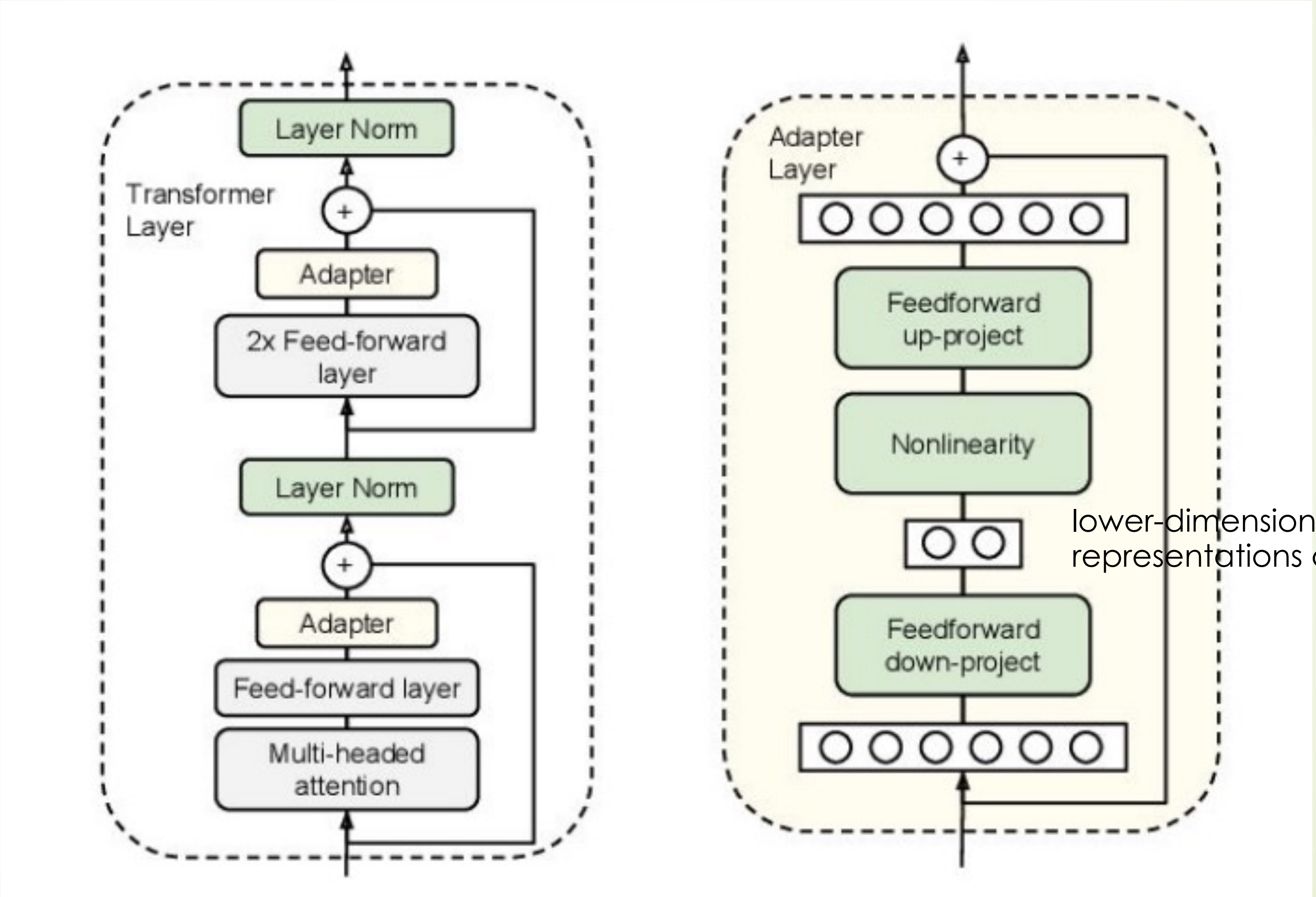


Figure 4: Different types of PEFT algorithms.

Addictive: Adapters



lower-dimensional representations of the new task

Reparametrization-based: LoRA

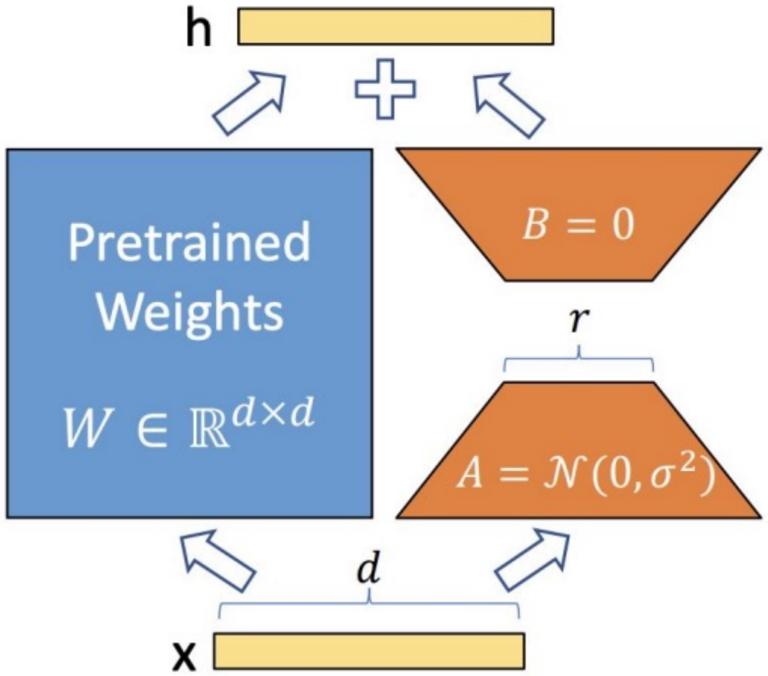


Figure 1: Our reparametrization. We only train A and B.

$$h = W_0x + \Delta W x = W_0 x + BAx$$

- Only update the low-rank matrix
- 10000x less trainable parameter
- 3x less GPU memory requirement
- Apply to any linear layer
- No inference overhead

LoRA: Key advantages

- A pre-trained model can be shared and used to **build many small LoRA modules for different tasks**.
- LoRA makes **training more efficient** and lowers the hardware barrier by up to 3 times when using adaptive optimizers
- Allows us to merge the trainable matrices with the frozen weights when deployed, **introducing no additional inference latency** compared to a fully fine-tuned model

LoRA formulation

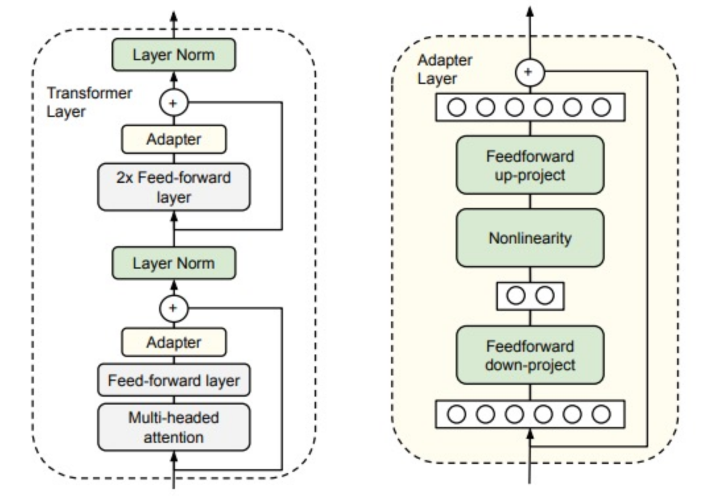
$$\max_{\Theta} \sum_{(x,y)\in\mathcal{Z}} \sum_{t=1}^{|y|} \log\left(p_{\Phi_0+\Delta\Phi(\Theta)}(y_t|x, y_{<t})\right)$$

$\Delta\Phi = \Delta\Phi(\Theta)$ Task-specific increment

$|\Theta| \ll |\Phi_0|$ Smaller-sized set of parameters

Results: Baselines

- FT: Fully fine-tuned
- BitFit: we only train the bias vectors while freezing everything else
- PreEmbed: applying learnable prefixes input tokens.
- PreLayer: applying learnable prefixes to selected layers
- Adapter tuning: inserts adapter layers between the self-attention module (and the MLP module) and the subsequent residual connection
  - Adaptor_H: original design, two fully connected layers with a nonlinearity in between
  - Adaptor_L/P
  - Adapter_D: AdapterDrop which drops some adapter layers for greater efficiency



Multi-Genre Natural Language Inference

Stanford Sentiment Treebank

Corpus of Linguistic Acceptability

Question Natural Language Inference

Quora Question Pairs

Semantic Textual Similarity Benchmark

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$_{\pm.3}$	**95.1**$_{\pm.2}$	89.7$_{\pm.7}$	63.4$_{\pm1.2}$	**93.3**$_{\pm.3}$	90.8$_{\pm.1}$	**86.6**$_{\pm.7}$	**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	**90.6**$_{\pm.2}$	96.2$_{\pm.5}$	**90.9**$_{\pm1.2}$	**68.2**$_{\pm1.9}$	**94.9**$_{\pm.3}$	91.6$_{\pm.1}$	**87.4**$_{\pm2.5}$	**92.6**$_{\pm.2}$	**89.0**
$\text{RoB}_{\text{large}}$ (Adpt^{P})†	3.0M	90.2$_{\pm.3}$	96.1$_{\pm.3}$	90.2$_{\pm.7}$	**68.3**$_{\pm1.0}$	**94.8**$_{\pm.2}$	**91.9**$_{\pm.1}$	83.8$_{\pm2.9}$	92.1$_{\pm.7}$	88.4
$\text{RoB}_{\text{large}}$ (Adpt^{P})†	0.8M	**90.5**$_{\pm.3}$	**96.6**$_{\pm.2}$	89.7$_{\pm1.2}$	67.8$_{\pm2.5}$	**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	**90.6**$_{\pm.2}$	96.2$_{\pm.5}$	**90.2**$_{\pm1.0}$	68.2$_{\pm1.9}$	**94.8**$_{\pm.3}$	91.6$_{\pm.2}$	**85.2**$_{\pm1.1}$	**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	**91.9**$_{\pm.2}$	96.9$_{\pm.2}$	**92.6**$_{\pm.6}$	**72.4**$_{\pm1.1}$	**96.0**$_{\pm.1}$	**92.9**$_{\pm.1}$	**94.9**$_{\pm.4}$	**93.0**$_{\pm.2}$	**91.3**

Table 2: $\text{RoBERTa}_{\text{base}}$, $\text{RoBERTa}_{\text{large}}$, and $\text{DeBERTa}_{\text{XXL}}$ with different adaptation methods on the GLUE benchmark. We report the overall (matched and mismatched) accuracy for MNLI, Matthew's correlation for CoLA, Pearson correlation for STS-B, and accuracy for other tasks. Higher is better for all metrics. * indicates numbers published in prior works. † indicates runs configured in a setup similar to Houlsby et al. (2019) for a fair comparison.

End-to-End Natural Language Generation Challenge: Convert structured meaning representations into fluent and coherent text

Model & Method	# Trainable Parameters	Precision of n-grams		E2E NLG Challenge		Precision & Recall of n-grams
		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	**70.4**$_{\pm.1}$	**8.85**$_{\pm.02}$	**46.8**$_{\pm.2}$	**71.8**$_{\pm.1}$	**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$_{\pm.2}$	**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	**70.4**$_{\pm.1}$	**8.89**$_{\pm.02}$	**46.8**$_{\pm.2}$	**72.0**$_{\pm.2}$	2.47$_{\pm.02}$

Table 3: GPT-2 medium (M) and large (L) with different adaptation methods on the E2E NLG Challenge. For all metrics, higher is better. LoRA outperforms several baselines with comparable or fewer trainable parameters. Confidence intervals are shown for experiments we ran. * indicates numbers published in prior works.

natural language to SQL query generation

Multi-Genre Natural Language Inference

dialogue summarization

Model&Method	# Trainable Parameters	WikiSQL Acc. (%)	MNLI-m Acc. (%)	SAMSum R1/R2/RL
GPT-3 (FT)	175,255.8M	**73.8**	89.5	52.0/28.0/44.5
GPT-3 (BitFit)	14.2M	71.3	91.0	51.3/27.4/43.5
GPT-3 (PreEmbed)	3.2M	63.1	88.6	48.3/24.2/40.5
GPT-3 (PreLayer)	20.2M	70.1	89.5	50.8/27.3/43.5
GPT-3 ($\text{Adapter}^{\text{H}}$)	7.1M	71.9	89.8	53.0/28.9/44.8
GPT-3 ($\text{Adapter}^{\text{H}}$)	40.1M	73.2	**91.5**	53.2/29.0/45.1
GPT-3 (LoRA)	4.7M	73.4	**91.7**	**53.8/29.8/45.9**
GPT-3 (LoRA)	37.7M	**74.0**	**91.6**	53.4/29.2/45.1

Table 4: Performance of different adaptation methods on GPT-3 175B. We report the logical form validation accuracy on WikiSQL, validation accuracy on MultiNLI-matched, and Rouge-1/2/L on SAMSum. LoRA performs better than prior approaches, including full fine-tuning. The results on WikiSQL have a fluctuation around $\pm 0.5\%$, MNLI-m around $\pm 0.1\%$, and SAMSum around $\pm 0.2/\pm 0.2/\pm 0.1$ for the three metrics.

R1: overlap of unigrams (single words)
RL: Longest Common Subsequence

WHICH WEIGHT MATRICES IN TRANSFORMER SHOULD WE APPLY LORA TO

	# 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**

OPTIMAL RANK r FOR LORA

	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

Table 6: Validation accuracy on WikiSQL and MultiNLI with different rank r. To our surprise, a rank as small as one suffices for adapting both W_q and W_v on these datasets while training W_q alone needs a larger r. We conduct a similar experiment on GPT-2 in Section H.2.

QLoRA: **Quantized** Low-rank Adaption

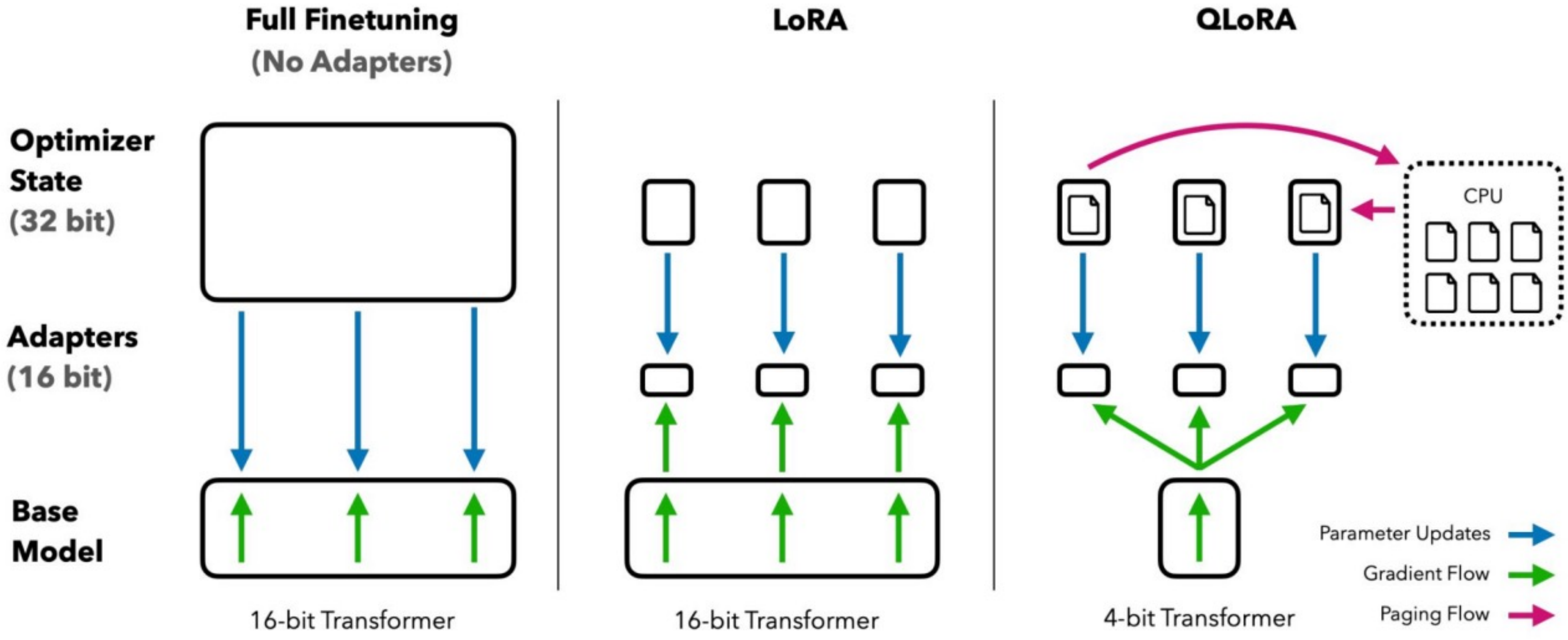
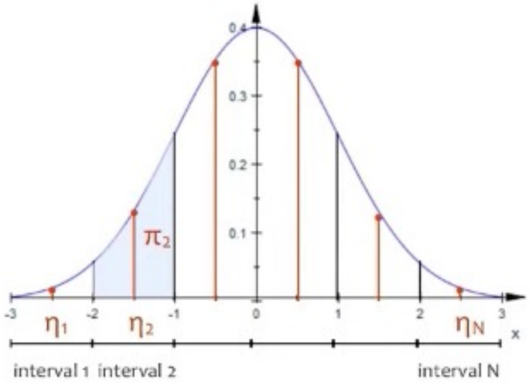


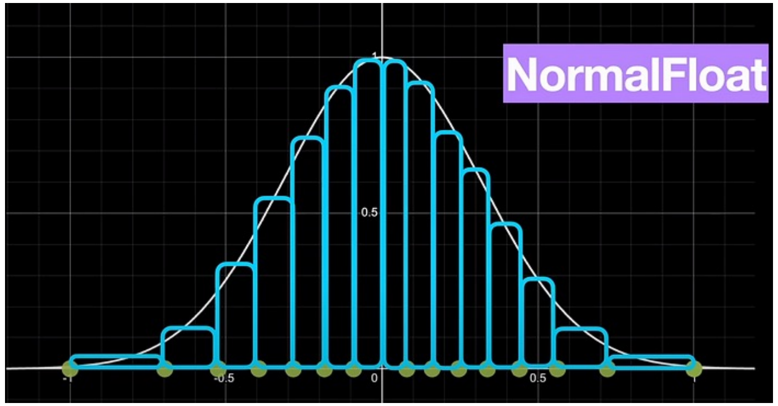
Figure 1: Different finetuning methods and their memory requirements. QLoRA improves over LoRA by quantizing the transformer model to 4-bit precision and using paged optimizers to handle memory spikes.

4-bit NormalFloat Quantization

- **Motivation** : Weights usually have a zero-centered normal distribution



Linear Method



Quantile Method

Double Quantization

Memory and compute efficiency: Save memory while speeding up computation.

Fit larger models: Allow massive models to be trained or run on limited hardware.

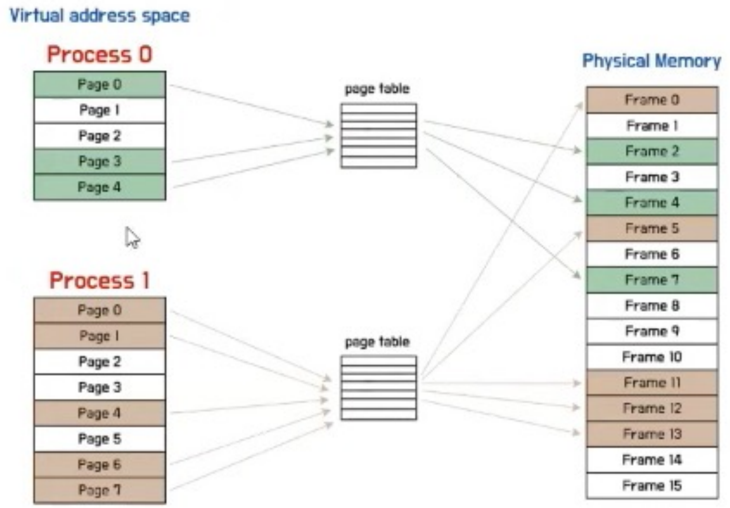
Minimal trade-offs in performance: Maintain model effectiveness while optimizing resources.

Original Value	Quantized Value	2-bit Code
0.1	0	00
0.23	1	01
0.45	3	10
0.67	5	11
0.89	7	11

This table illustrates the transformation of the original floating-point values into a compact 2-bit representation through double quantization.

Paged Optimizers

- **Utilizing NVIDIA Unified Memory Feature:**
  - Automatic page-to-page transfers between CPU and GPU memory.
- **Functionality of the Feature:**
  - Similar to regular memory paging between CPU RAM and disk storage.
  - When GPU runs out-of-memory:
    - Optimizer states are automatically evicted to CPU RAM.
  - When memory is needed in the optimizer:
    - Paged data is automatically transferred back into GPU memory.



page in CPU/GPU memory typically ranges from **4 KB to 64 KB** (though 4 KB is most common).

Float v.s. Nfloat v.s. Nfloat+DQ

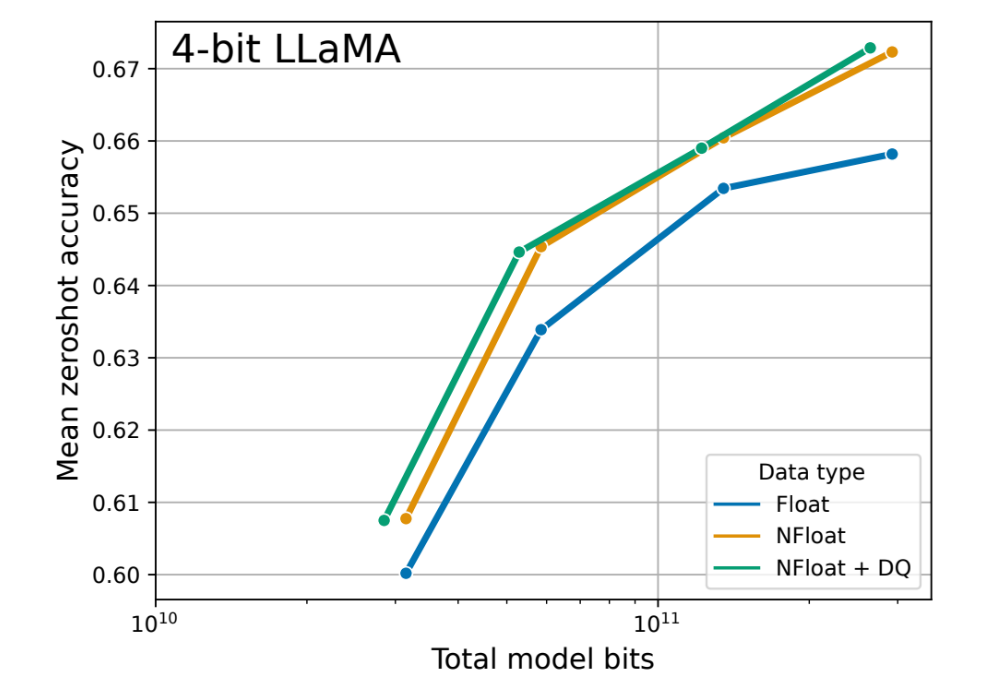


Figure 3: Mean zero-shot accuracy over Winogrande, HellaSwag, PiQA, Arc-Easy, and Arc-Challenge using LLaMA models with different 4-bit data types. The NormalFloat data type significantly improves the bit-for-bit accuracy gains compared to regular 4-bit Floats. While Double Quantization (DQ) only leads to minor gains, it allows for a more fine-grained control over the memory footprint to fit models of certain size (33B/65B) into certain GPUs (24/48GB).

WinoGrande: which noun a pronoun refers to
HellaSwag: common sense reasoning
PiQA: Physical Interaction Question Answering
Arc-Easy: AI2 Reasoning, elementary science questions
Arc-Challenge: AI2 Reasoning, complex understanding

Result benchmark

Table 3: Experiments comparing 16-bit BrainFloat (BF16), 8-bit Integer (Int8), 4-bit Float (FP4), and 4-bit NormalFloat (NF4) on GLUE and Super-NaturalInstructions. QLoRA replicates 16-bit LoRA and full-finetuning.

Dataset	GLUE (Acc.)	Super-NaturalInstructions (RougeL)				
Model	RoBERTa-large	T5-80M	T5-250M	T5-780M	T5-3B	T5-11B
BF16	88.6	40.1	42.1	48.0	54.3	62.0
LoRA BF16	88.8	40.5	42.6	47.1	55.4	60.7
QLoRA Int8	88.8	40.4	42.9	45.4	56.5	60.7
QLoRA FP4	88.6	40.3	42.4	47.5	55.6	60.9
QLoRA NF4 + DQ	-	40.4	42.7	47.7	55.3	60.9

Result benchmark

Table 4: Mean 5-shot MMLU test accuracy for LLaMA 7-65B models finetuned with adapters on Alpaca and FLAN v2 for different data types. Overall, NF4 with double quantization (DQ) matches BFloat16 performance, while FP4 is consistently one percentage point behind both.

	Mean 5-shot MMLU Accuracy								
LLaMA Size	7B		13B		33B		65B		Mean
Dataset	Alpaca	FLAN v2	Alpaca	FLAN v2	Alpaca	FLAN v2	Alpaca	FLAN v2	
BFloat16	38.4	45.6	47.2	50.6	57.7	60.5	61.8	62.5	53.0
Float4	37.2	44.0	47.3	50.0	55.9	58.5	61.3	63.3	52.2
NFloat4 + DQ	39.0	44.5	47.5	50.7	57.3	59.2	61.8	63.9	53.1

Hugging Face - PEFT library

```python
model_id = "EleutherAI/gpt-neox-20b"
bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_use_double_quant=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.bfloat16
)

tokenizer = AutoTokenizer.from_pretrained(model_id)
model = AutoModelForCausalLM.from_pretrained(model_id, quantization_config=bnb_config
```

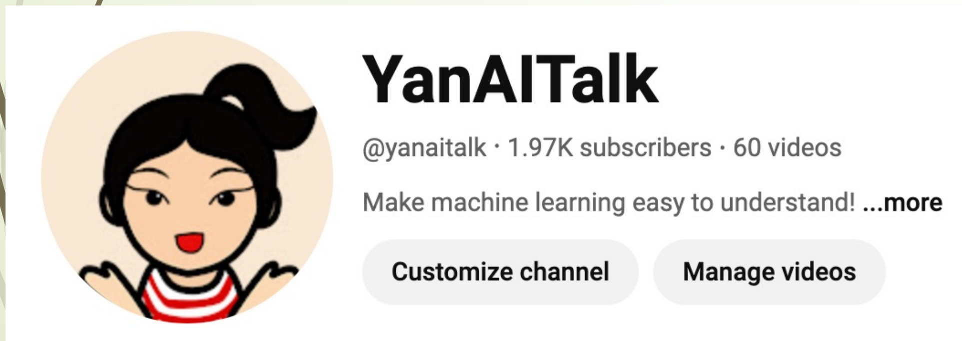
```python
from peft import LoraConfig, get_peft_model

config = LoraConfig(
    r=8,
    lora_alpha=32,
    target_modules=["query_key_value"],
    lora_dropout=0.05,
    bias="none",
    task_type="CAUSAL_LM"
)

model = get_peft_model(model, config)
print_trainable_parameters(model)
```

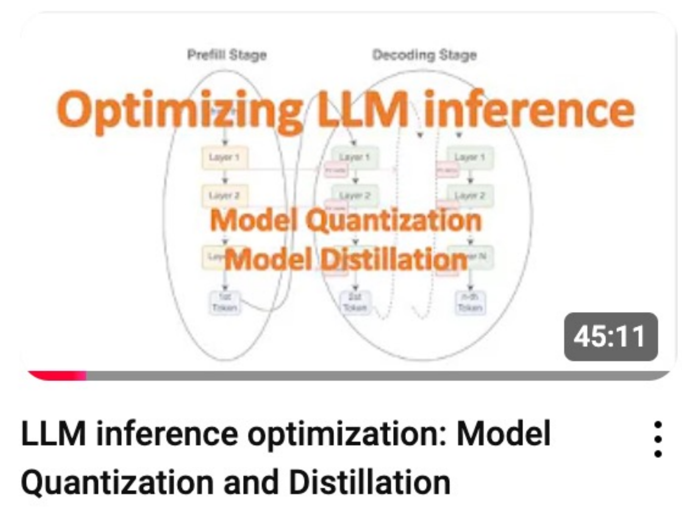
```
trainable params: 8650752 || all params: 10597552128 || trainable%: 0.08162971878329976
```

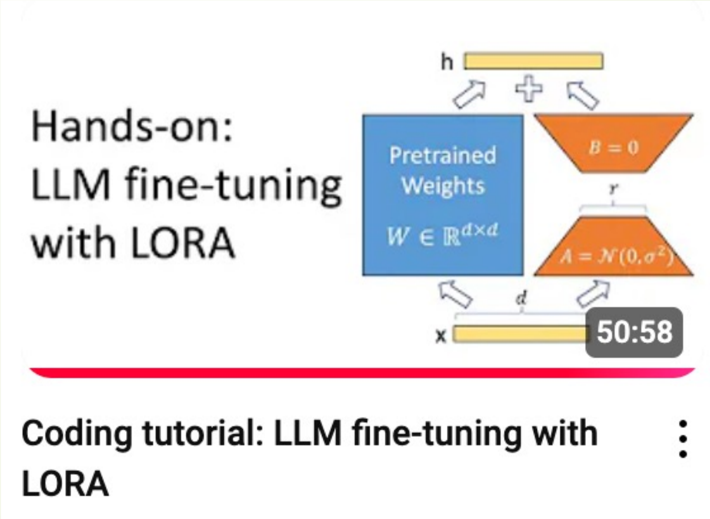
Tutorial: Bnb_4bit-training

Conclusion

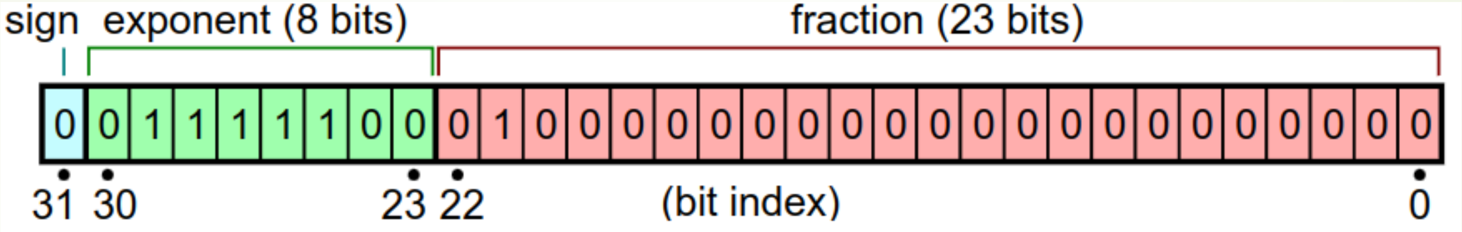
- LoRA provides a parameter-efficient fine-tuning by adding low-rank adapters to pretrained models
- QLoRA improves over LoRA with efficient fine-tuning on quantized model to save momery and compute
  - Compatible with 4-bit quantized models, enabling to fine-tune very large models like LLaMA-2-65B on a 24GB GPU
  - Match the performance of the full-fine tuning

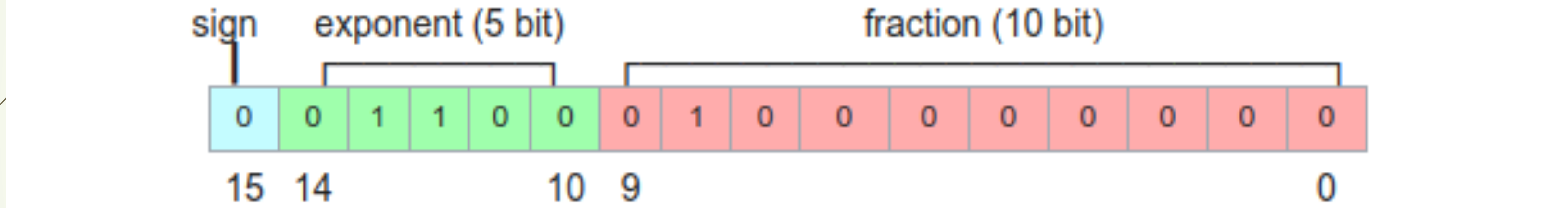


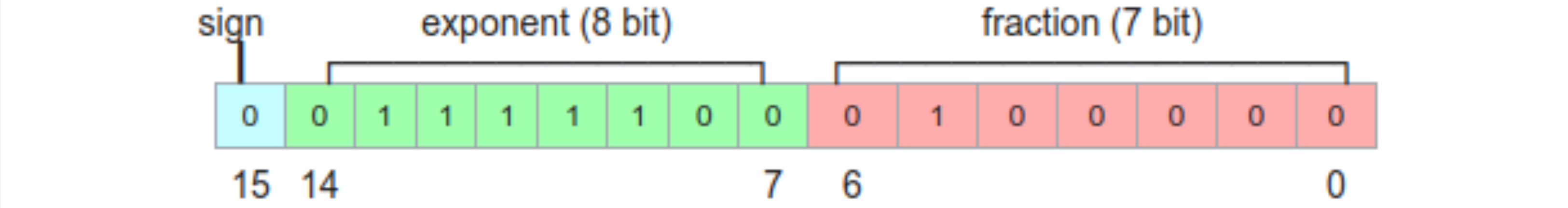




FP32, FP16, BFP16, FP4







sign	exponent		sig
0	0	0	0