

# Efficient Tuning of Large Language Models

## A Practical Guide

Presenter: Yaowei Zheng

Beihang University

June 6, 2023

# Auto-Regressive Language Model

An auto-regressive language model (like GPT) is trained to predict a probability distribution of the  $i$ -th token given the  $i - 1$  tokens:

$$P(w_i | w_1, w_2, \dots, w_{i-1}; \theta) \quad (1)$$

The optimization uses a causal language modeling objective:

$$\mathcal{L}_{\text{GPT}} = - \sum_{i=1}^n \log P(w_i | w_1, w_2, \dots, w_{i-1}; \theta) \quad (2)$$

# Training Cost of LLMs

Model	Model Creator	Modality	# Parameters	Tokenizer	Window Size	Access	Total Tokens	Total Queries	Total Cost
J1-Jumbo v1 (178B)	AI21 Labs	Text	178B	AI21	2047	limited	327,443,515	591,384	\$10,926
J1-Grande v1 (17B)	AI21 Labs	Text	17B	AI21	2047	limited	326,815,150	591,384	\$2,973
J1-Large v1 (7.5B)	AI21 Labs	Text	7.5B	AI21	2047	limited	342,616,800	601,560	\$1,128
Anthropic-LM v4-s3 (52B)	Anthropic	Text	52B	GPT-2	8192	closed	767,856,111	842,195	-
BLOOM (176B)	BigScience	Text	176B	BLOOM	2048	open	581,384,088	849,303	4,200 GPU hours
T0++ (11B)	BigScience	Text	11B	T0	1024	open	305,488,229	406,072	1,250 GPU hours
Cohere xlarge v20220609 (52.4B)	Cohere	Text	52.4B	Cohere	2047	limited	397,920,975	597,252	\$1,743
Cohere large v20220720 (13.1B) <sup>58</sup>	Cohere	Text	13.1B	Cohere	2047	limited	398,293,651	597,252	\$1,743
Cohere medium v20220720 (6.1B)	Cohere	Text	6.1B	Cohere	2047	limited	398,036,367	597,252	\$1,743
Cohere small v20220720 (410M) <sup>59</sup>	Cohere	Text	410M	Cohere	2047	limited	399,114,309	597,252	\$1,743
GPT-J (6B)	EleutherAI	Text	6B	GPT-J	2048	open	611,026,748	851,178	860 GPU hours
GPT-NeoX (20B)	EleutherAI	Text	20B	GPT-NeoX	2048	open	599,170,730	849,830	540 GPU hours
T5 (11B)	Google	Text	11B	T5	512	open	199,017,126	406,072	1,380 GPU hours
UL2 (20B)	Google	Text	20B	UL2	512	open	199,539,380	406,072	1,570 GPU hours
OPT (66B)	Meta	Text	66B	OPT	2048	open	612,752,867	851,178	2,000 GPU hours
OPT (175B)	Meta	Text	175B	OPT	2048	open	610,436,798	851,178	3,400 GPU hours
TNLG v2 (6.7B)	Microsoft/NVIDIA	Text	6.7B	GPT-2	2047	closed	417,583,950	590,756	-
TNLG v2 (530B)	Microsoft/NVIDIA	Text	530B	GPT-2	2047	closed	417,111,519	590,756	-
GPT-3 davinci v1 (175B)	OpenAI	Text	175B	GPT-2	2048	limited	422,001,611	606,253	\$8,440
GPT-3 curie v1 (6.7B)	OpenAI	Text	6.7B	GPT-2	2048	limited	423,016,414	606,253	\$846
GPT-3 babbage v1 (1.3B)	OpenAI	Text	1.3B	GPT-2	2048	limited	422,123,900	606,253	\$211
GPT-3 ada v1 (350M)	OpenAI	Text	350M	GPT-2	2048	limited	422,635,705	604,253	\$169
InstructGPT davinci v2 (175B*)	OpenAI	Text	175B*	GPT-2	4000	limited	466,872,228	599,815	\$9,337
InstructGPT curie v1 (6.7B*)	OpenAI	Text	6.7B*	GPT-2	2048	limited	420,004,477	606,253	\$840
InstructGPT babbage v1 (1.3B*)	OpenAI	Text	1.3B*	GPT-2	2048	limited	419,036,038	604,253	\$210
InstructGPT ada v1 (350M*)	OpenAI	Text	350M*	GPT-2	2048	limited	418,915,281	604,253	\$168
Codex davinci v2	OpenAI	Code	Unknown	GPT-2	4000	limited	46,272,590	57,051	\$925
Codex cushman v1	OpenAI	Code	Unknown	GPT-2	2048	limited	42,659,399	59,751	\$85
GLM (130B)	Tsinghua University	Text	130B	ICE	2048	open	375,474,243	406,072	2,100 GPU hours
YaLM (100B)	Yandex	Text	100B	Yandex	2048	open	378,607,292	405,093	2,200 GPU hours

Image source: Liang *et al.*, “Holistic Evaluation of Language Models”.

# Anatomy of Model's Memory

## Model:

- Parameters (float32):  $4 \text{ bytes} * \# \text{params}$
- Gradients (float32):  $4 \text{ bytes} * \# \text{params}$

## Optimizer:

- Master weight (float32):  $4 \text{ bytes} * \# \text{params}$
- Adam's m (float32):  $4 \text{ bytes} * \# \text{params}$
- Adam's v (float32):  $4 \text{ bytes} * \# \text{params}$

## Activations:

- Activations (float32):  $4 \text{ bytes} * \text{batch size} * \text{hidden size} * \text{sequence length}$

We want to reduce the memory cost of LLaMA-65B from 780GB to 48GB.

# Mixed Precision Training

Mixed precision training [1] employs a combination of single- and half-precision floating point representations to reduce memory overheads<sup>1</sup>.

bfloat16: Brain Floating Point Format

Range:  $\sim 1e^{-38}$  to  $\sim 3e^{38}$



fp32: Single-precision IEEE Floating Point Format

Range:  $\sim 1e^{-38}$  to  $\sim 3e^{38}$



fp16: Half-precision IEEE Floating Point Format

Range:  $\sim 5.96e^{-8}$  to 65504



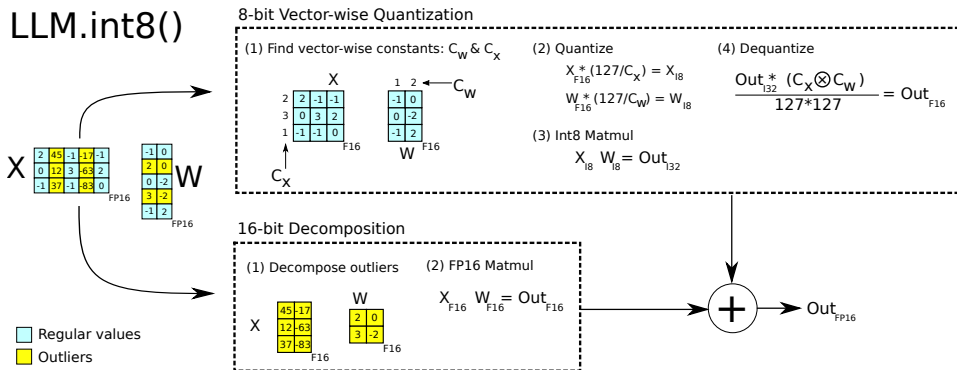
Image source: <https://cloud.google.com/tpu/docs/bfloat16>.

<sup>1</sup>bfloat16 is only compatible with 3080, A100 and TPUs.

# Model Quantizing

LLMs can be loaded in **8-bit integer mode** without performance degradation. [2]

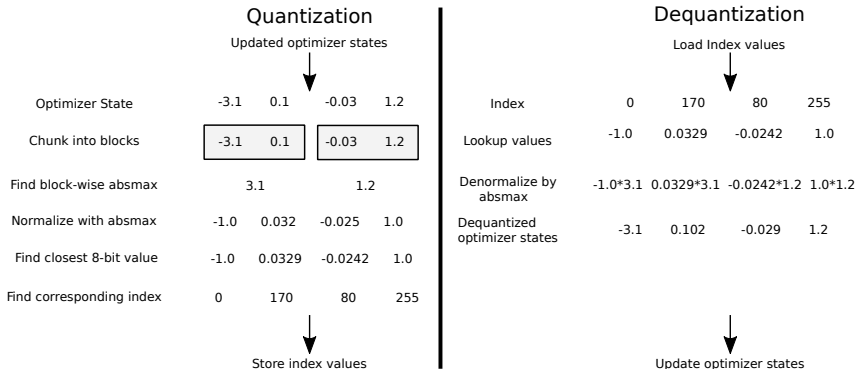
LLM.int8()



Mixed-precision decomposition scheme in LLM.int8().

# 8-bit Optimizers

Similarly, an 8-bit optimizer is developed for reduce the memory costs. [3]



Block-wise dynamic quantization in 8-bit optimizers.

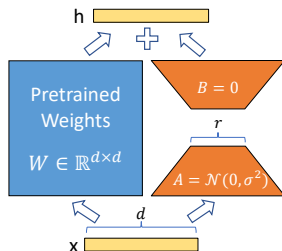
# Low-Rank Adapters

## Intrinsic Dimensionality

The *intrinsic dimension* [4] of an objective function is the **minimum dimension** required to solve the problem.

With the low-rank adapters [5], we can fine-tune LLMs by train a small portion of parameters by reparameterizing the dense layers with rank decomposition matrices.

LoRA can reduce the GPU memory requirement by **3 times** by training only **0.01% parameters**.





# Gradient Checkpointing

Gradient checkpointing [6] saves strategically selected activations in the computational graph and only re-computes a fraction of the activations.

For feed-forward networks, the optimal choice is to mark every  $\sqrt{n}$ -th node as a checkpoint, giving  $\sqrt{n}$  memory usage for feed-forward networks.

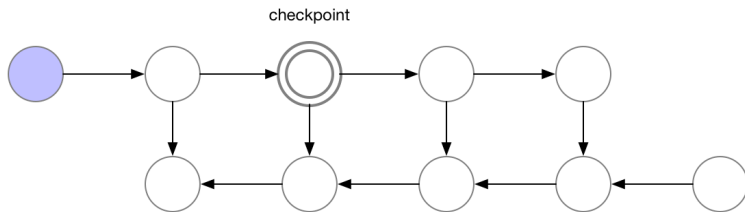


Image source: <https://github.com/cybertronai/gradient-checkpointing>.

# Other Practical Strategies

- Gradient accumulation
- Decreasing batch size
- CPU Offloading [7]
- All-reduce operation (aka distributed data parallelism)
- Megatron-LM [8] (pipeline model parallelism)
- DeepSpeed ZeRO [9] (aka fully shared data parallelism)

# Challenge: Training Stability [11]

- Compute attention scores with float32 softmax to avoid overflow.

$$\text{softmax} \left( \frac{Q_i K_i^\top}{\sqrt{d}} \right) = \text{softmax} \left( \left( \frac{Q_i K_i^\top}{\alpha \sqrt{d}} - \max \left( \frac{Q_i K_i^\top}{\alpha \sqrt{d}} \right) \right) \times \alpha \right) \quad (3)$$

$$= \text{FP16} \left( \text{softmax} \left( \text{FP32} \left( \frac{Q_i K_i^\top}{\alpha \sqrt{d}} \right) \times \alpha \right) \right) \quad (4)$$

- Decrease the gradients of the embedding layer.

```
1 word_emb = word_emb * alpha + word_emb.detach() * (1 - alpha)
```

- Use DeepNorm [10] instead of PreLN.

# High-Performance Training Framework

## HuggingFace Accelerate

- Provides APIs to easily download and train pre-trained LLMs.
- Supports quantization, distributed training, and adapters.

## Colossal AI

- Provides a collection of parallel components.
- Supports parallel training and ZeRO optimizer.

## DeepSpeed

- Provides distributed training and inference.
- Supports ZeRO training and inference.

## OneFlow

# Efficient Tuning of LLMs: a Case

We implemented QLoRA [12], which combines low-rank adapters and 4-bit quantized LLMs to fine-tune a 65B parameter model on a single 48GB GPU.

Dataset Model	GLUE (Acc.)	Super-NaturalInstructions (Rouge-l)				
	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
BF16 replication	88.6	40.0	42.2	47.3	54.9	-
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

QLoRA adopts **4-bit NormalFloat (NF4)** quantization and Double Quantization (DQ).

# Efficient Tuning of LLMs: a Case

The peft library enables **parameter-efficient fine-tuning** of pre-trained language models on **consumer GPUs** with the following few lines.

```
1 from transformers import AutoTokenizer, AutoModelForCausalLM
2 from peft import LoraConfig, TaskType, get_peft_model
3
4 peft_config = LoraConfig(
5     task_type=TaskType.CAUSAL_LM, inference_mode=False,
6     r=8, lora_alpha=32, lora_dropout=0.1
7 )
8
9 tokenizer = AutoTokenizer.from_pretrained("name/or/path/to/your/model")
10 model = AutoModelForCausalLM.from_pretrained("name/or/path/to/your/model")
11 model = get_peft_model(model, peft_config)
```

# Efficient Tuning of LLMs: a Case

QLoRA can be easily implemented using the `bitsandbytes` library.

```
1 from transformers import AutoModelForCausalLM, BitsAndBytesConfig
2
3 model = AutoModelForCausalLM.from_pretrained(
4     model_name_or_path="name/or/path/to/your/model",
5     load_in_4bit=True,
6     device_map="auto",
7     max_memory=max_memory,
8     torch_dtype=torch.bfloat16,
9     quantization_config=BitsAndBytesConfig(
10         load_in_4bit=True,
11         bnb_4bit_compute_dtype=torch.bfloat16,
12         bnb_4bit_use_double_quant=True,
13         bnb_4bit_quant_type="nf4"
14     )
15 )
```

# Efficient Tuning of LLMs: a Case

We use the **Trainer API** provided by transformers to fine-tune the models.

```
1 from transformers import TrainingArguments, Trainer
2
3 training_args = TrainingArguments(
4     output_dir="path/to/save/folder",
5     learning_rate=5e-5
6 )
7
8 trainer = Trainer(
9     model=model,
10    args=training_args,
11    train_dataset=dataset["train"],
12    eval_dataset=dataset["test"],
13    tokenizer=tokenizer
14 )
15 trainer.train()
```



# Experimental Setup

We adopted QLoRA to fine-tune **LLaMA-30B** model using the Stanford Alpaca dataset [13] on **a single A100 40GB GPU**.

Hyper-parameters	Value
lora rank	8
lora alpha	32
lora dropout	0.1
lora target	q_proj, v_proj
train batch size	4
eval batch size	8
gradient accumulation steps	4
learning rate	1e-4
max grad norm	0.3
num train epochs	1.0

# Training Dynamics

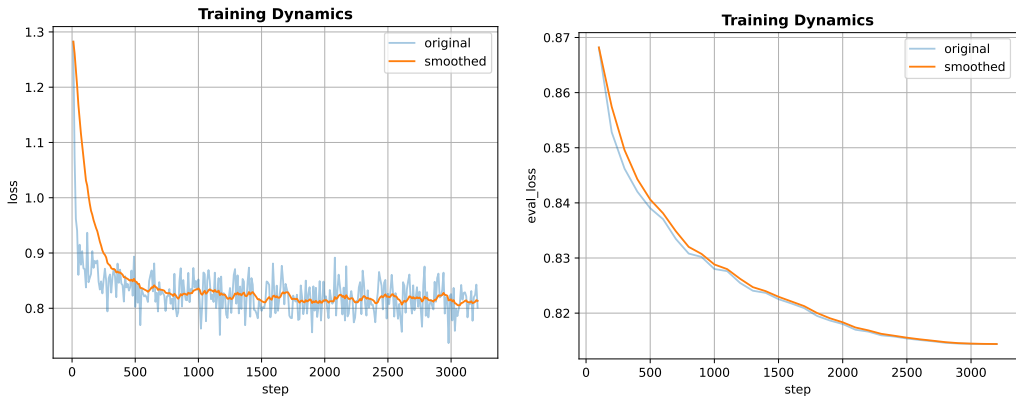


Figure: Evolution of the cross-entropy loss in training (left) and evaluation (right).

# Experimental Results

Input: Please explain machine learning to me.

LLaMA: Machine learning is a field of artificial intelligence (AI) that enables systems to automatically improve their performance through experience. It involves training algorithms to recognize patterns and make predictions or decisions based on data. The key concept behind machine learning is that rather than explicitly programming a computer to perform a specific task, you can instead provide it with data, a set of rules, and let it learn on its own.

There are several different types of machine learning algorithms, including supervised learning, unsupervised learning, and reinforcement learning. Supervised learning involves training a model with labeled data, where the model is given examples of the correct output for each input. This is commonly used for classification and regression tasks. Unsupervised learning, on the other hand, involves training a model with unlabeled data, where the model is tasked with finding patterns and structure in the data. This is commonly used for clustering and dimensionality reduction tasks. Reinforcement learning is a type of learning where the model is trained through trial and error, with the goal of maximizing a reward function. This is commonly used for decision-making and planning tasks.

# Experimental Results

Input: Make a tour plan of Beijing.

LLaMA: Here is a suggested tour plan for Beijing:

Day 1: Visit the Forbidden City and Tiananmen Square, which are two of the most iconic attractions in Beijing. Afterwards, head to the Temple of Heaven, which is a beautiful and peaceful place to spend the afternoon.

Day 2: Start the day with a visit to the Summer Palace, a stunning imperial garden with beautiful architecture and views. In the afternoon, head to the Great Wall of China, which is one of the most impressive sights in the world.

Day 3: Spend the morning at the National Museum of China, which is a great place to learn about Chinese history and culture. In the afternoon, head to the 798 Art Zone, which is a vibrant area full of galleries, cafes, and shops.

Day 4: Take a day trip to the

The whole implementation is made available at:

<https://github.com/hiyouga/LLaMA-Efficient-Tuning>

We provide:

- Scripts for easily fine-tuning of the LLaMA and BLOOM models.
- Several parameter-efficient fine-tuning methods like QLoRA.
- Whole training procedure of ChatGPT-like models including RLHF.
- Various instruction-following datasets for instruction tuning.

# References

-  Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al.  
Mixed precision training.  
*In International Conference on Learning Representations*, 2018.
-  Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer.  
Llm.int8(): 8-bit matrix multiplication for transformers at scale.  
*In Advances in Neural Information Processing Systems*, volume 35, pages 30318–30332, 2022.
-  Tim Dettmers, Mike Lewis, Sam Shleifer, and Luke Zettlemoyer.  
8-bit optimizers via block-wise quantization.  
*In International Conference on Learning Representations*, 2022.

# References

-  Chunyuan Li, Heerad Farkhor, Rosanne Liu, and Jason Yosinski.  
Measuring the intrinsic dimension of objective landscapes.  
*In International Conference on Learning Representations*, 2018.
-  Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al.  
Lora: Low-rank adaptation of large language models.  
*In International Conference on Learning Representations*, 2022.
-  Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin.  
Training deep nets with sublinear memory cost.  
*arXiv preprint arXiv:1604.06174*, 2016.

# References



-  Jie Ren, Samyam Rajbhandari, Reza Yazdani Aminabadi, Olatunji Ruwase, Shuangyan Yang, Minjia Zhang, Dong Li, and Yuxiong He.  
Zero-offload: Democratizing billion-scale model training.  
In *USENIX Annual Technical Conference*, pages 551–564, 2021.
-  Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro.  
Megatron-lm: Training multi-billion parameter language models using model parallelism.  
*arXiv preprint arXiv:1909.08053*, 2019.



# References

-  Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16. IEEE, 2020.
-  Hongyu Wang, Shuming Ma, Li Dong, Shaohan Huang, Dongdong Zhang, and Furu Wei. Deepnet: Scaling transformers to 1,000 layers. *arXiv preprint arXiv:2203.00555*, 2022.
-  Aohan Zeng, Xiao Liu, Zhengxiao Du, Zihan Wang, Hanyu Lai, Ming Ding, Zhuoyi Yang, Yifan Xu, Wendi Zheng, Xiao Xia, et al. Glm-130b: An open bilingual pre-trained model. In *International Conference on Learning Representations*, 2023.

# References

-  Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer.  
Qlora: Efficient finetuning of quantized llms.  
*arXiv preprint arXiv:2305.14314*, 2023.
-  Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto.  
Stanford alpaca: An instruction-following llama model.  
[https://github.com/tatsu-lab/stanford\\_alpaca](https://github.com/tatsu-lab/stanford_alpaca), 2023.