Introduction to LLaMA Factory

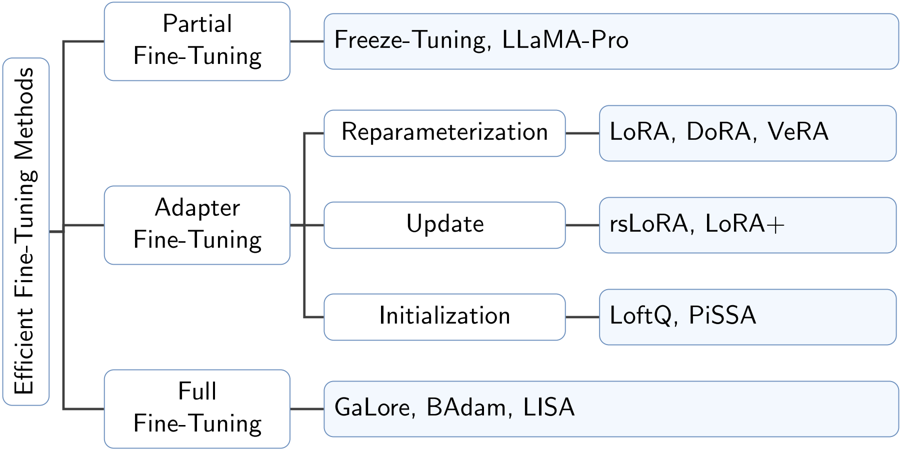
Towards Easy and Efficient LLM Fine-Tuning

Presenter: Yaowei Zheng

Beihang University

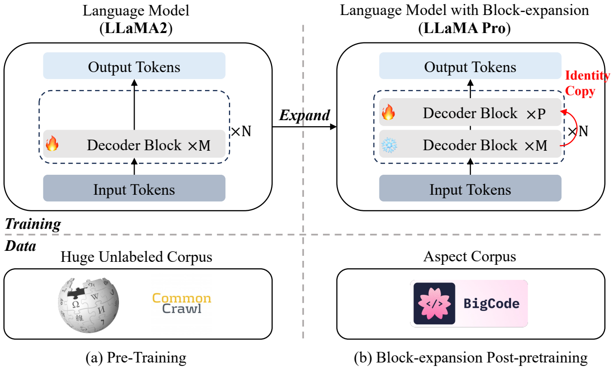
Jul 8, 2024

Overview of Efficient Fine-Tuning Methods



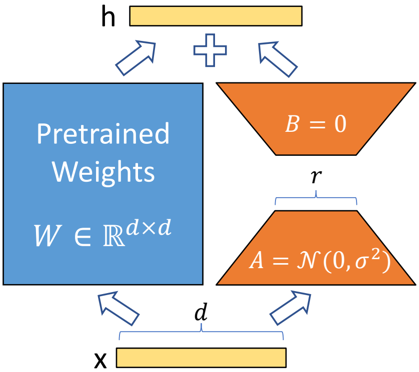
Partial Fine-Tuning: LLaMA-Pro

LLaMA-Pro replicates identical blocks every M blocks.



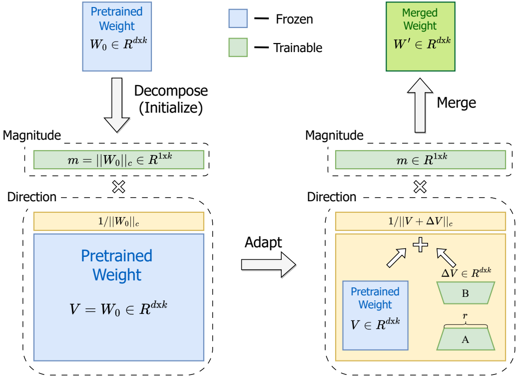
Adapter Fine-Tuning: LoRA

LoRA introduces a pair of trainable low-rank matrices and freezes pre-trained weights.



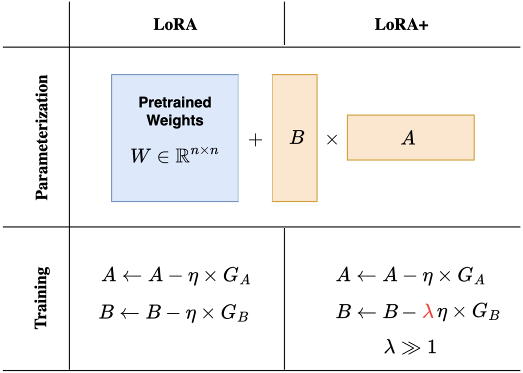
Adapter Fine-Tuning: DoRA

DoRA breaks down pre-trained weights into magnitude and direction components and updates directional components.



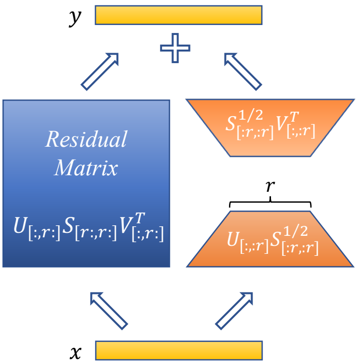
Adapter Fine-Tuning: LoRA+

LoRA+ adopts different learning rates on low-rank matrices.



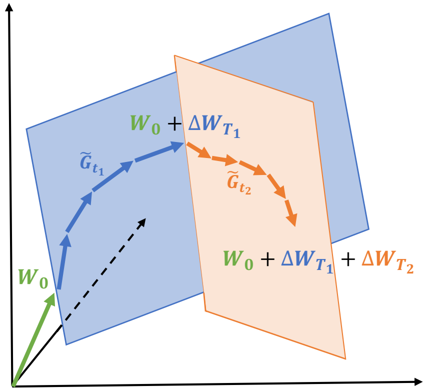
Adapter Fine-Tuning: PiSSA

PiSSA initializes adapters with the principal components.



Full Fine-Tuning: GaLore

GaLore projects gradients into lower-dimensional spaces.



Advantages of LLaMA Factory

LLaMA Factory holds merits in the following three aspects:

- **Efficiency:** By utilizing various optimization techniques, LLaMA Factory offers approximately $2\times$ speed-up and 60% memory saving in training.
- **User-friendliness:** By providing an easy-to-use web UI, LLaMA Factory enables training LLMs with just one click, without writing any code.
- **Scalability:** By leveraging the DeepSpeed framework, LLaMA Factory supports training a 180B model in a distributed environment.

How Much Efficiency Can LLaMA Factory Achieve?

LLaMA Factory combines GaLore [1] and QLoRA [2] to reduce the memory cost by 4 times for both full-parameter training and LoRA training.

Table: Minimum GPU requirements for training LLMs.

	Vanilla AMP	GaLore	LoRA	QLoRA
7B	120GB	**16GB**	16GB	**4GB**
13B	240GB	**32GB**	32GB	**8GB**
30B	600GB	**64GB**	64GB	**16GB**
70B	1200GB	**160GB**	160GB	**24GB**
8x7B	900GB	**120GB**	120GB	**18GB**

LLaMA Factory Hierarchy

Efficient Training

- Pre-Training
- Instruct Tuning (Answer)
- Policy Gradient (B, A, C)
- Direct Preference (B, A, C)

LLM Integration

- Flash Attention
- RoPE Scaling
- Quantization
- Efficient LoRA

Data Integration

- Dynamic Padding
- Function Calling
- Dataset Streaming
- Multi-Turn Dialog

Frameworks

PyTorch, Hugging Face, DeepSpeed

Accelerators

NVIDIA, HUAWEI, HYGON

Revisiting InstructGPT

InstructGPT [3] was trained with 3 different techniques:

Step 1: Make model to follow instructions via supervised fine-tuning (SFT).

- Fine-tune GPT-3 using supervised learning with prompts and responses in a sequence-to-sequence manner.

Step 2: Train a reward model with comparison data.

- Rank the outputs from the fine-tuned GPT models and train a reward model (RM) that assigns a scalar score to a given prompt-response pair.

Step 3: Align the model with human preference using reinforcement learning.

- Use the reward model as a reward function to fine-tune the SFT model using the proximal policy optimization (PPO) algorithm.

(Continuous) Pre-Training

Assuming we have an auto-regressive language model (like GPT-3) which is trained to predict a probability distribution of the i-th token given $i-1$ tokens:

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

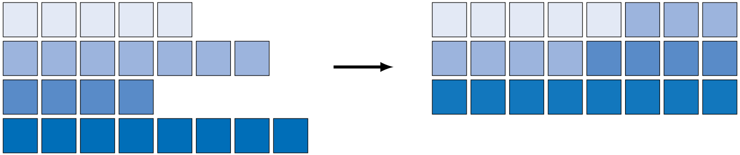
(Continuous) pre-training follows the standard causal language modelling task, which minimizes the following negative log-likelihood loss:

$$\mathcal{L}_{\mathrm{CLM}} = -\frac{1}{n}\sum_{i=1}^{n} \log P(x_i | x_1, x_2, \ldots, x_{i-1}; \theta) \tag{2}$$

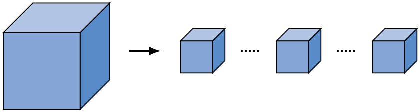
where $(x_1, x_2, \ldots, x_n)$ is the input sequence containing n words.

LLaMA Factory's Implementation

Sequence packing [4] packs sequences together and removes padding to efficiently learn sequences in variable lengths.



Dataset streaming [5] let us work with a smaller chunk of data to allocate more RAM to the model and avoid timeout errors in distributed training.



Supervised Fine-Tuning

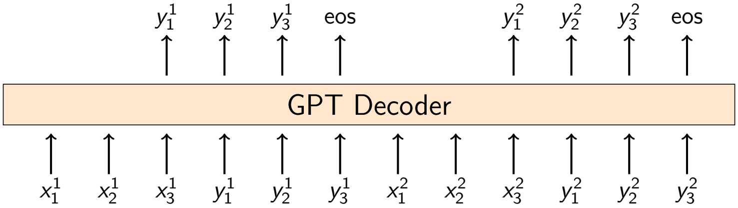
Supervised fine-tuning (SFT) adopts a sequence-to-sequence learning approach, where only the loss of the response part is calculated.

$$\mathcal{L}_{\mathrm{SFT}} = -\frac{1}{n}\sum_{i=1}^{n} \log P(y_i | x_1, x_2, \ldots, x_m, y_1, y_2, \ldots, y_{i-1}; \theta) \tag{3}$$

where $(x_1, x_2, \ldots, x_m)$ is the prompt sequence containing m words and $(y_1, y_2, \ldots, y_n)$ is the response sequence containing n words.

LLaMA Factory's Implementation

For **multi-turn dialog** dataset, we compute the loss of the response part in each round to efficiently learn the multi-turn conversation data.

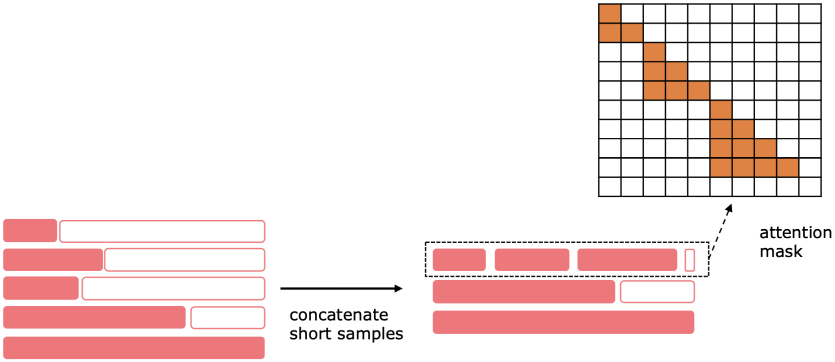


A **chat template** is applied to the input sequence to format the conversations.

[INST] Hello, how are you? [/INST] I'm doing great. How can I help you today? ⟨/s⟩ [INST] I'd like to show off how chat template works! [/INST]

LLaMA Factory's Implementation

We also supports sequence packing in supervised fine-tuning for efficiency.



Reward Modelling

The reward model is trained on a dataset of comparisons between two model outputs on the same input, using the pairwise ranking loss:

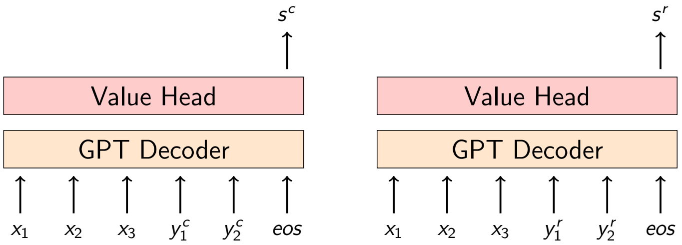
$$\mathcal{L}_{\mathrm{RM}} = -\sum_{(x,y^c,y^r)} \log\left(\sigma\left(r_\theta(x, y^c) - r_\theta(x, y^r)\right)\right) \tag{4}$$

where $r_\theta(x, y)$ is the scalar output of the reward model for prompt x and response y. Besides, y^c is the preferred response out of the pair (y^c, y^r).

LLaMA Factory's Implementation

We use the reward score on the eos token to compute the pairwise loss.

$$\mathcal{L}_{\mathrm{RM}} = \log\left(\sigma\left(s^c - s^r\right)\right) \tag{5}$$



Reinforcement Learning from Human Feedback

The SFT model can be further optimized using the **PPO algorithm** [6].

Consider a bandit problem, the model is expected to generate a response to a random prompt. The reward model assigns a score on the generated response.

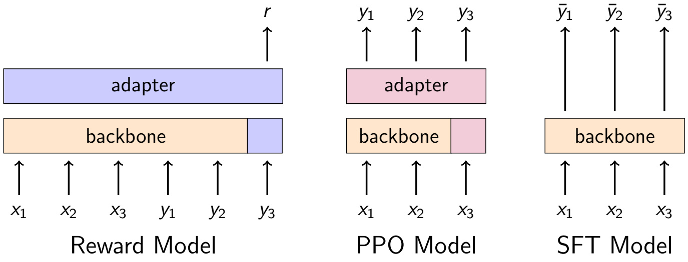
PPO maximizes the score with a per-token KL penalty between the PPO model and the SFT model, the objective function is:

$$\mathcal{J}(\phi) = \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}_{\pi_\phi^{\mathrm{PPO}}}} \left[r_\theta(x, y) - \beta \log \frac{\pi_\phi^{\mathrm{PPO}}(y|x)}{\pi^{\mathrm{SFT}}(y|x)} \right] \tag{6}$$

where π_ϕ^{PPO} is the PPO model, π^{SFT} is the SFT model.

LLaMA Factory's Implementation

We propose a multi-adapter RLHF framework that reduces memory consumption by 3 times by sharing the parameters of the backbone model.



We can apply scaling/normalization/clipping [7] to reward scores to improve the stability of training.

Direct Preference Optimization

The DPO algorithm [8] is an alternative to RLHF.

DPO aligns the model with human preferences without explicitly training a reward model, the objective function is:

$$\mathcal{L}_{\mathrm{DPO}} = - \mathop{\mathbb{E}}_{(x, y^c, y^r) \sim \mathcal{D}} \left[\log \sigma \left(\beta \log \frac{\pi^{\mathrm{DPO}}(y^c|x)}{\pi^{\mathrm{SFT}}(y^c|x)} - \beta \log \frac{\pi^{\mathrm{DPO}}(y^r|x)}{\pi^{\mathrm{SFT}}(y^r|x)} \right) \right] \tag{7}$$

where $r_\theta(x, y) = \beta \log \frac{\pi^{\mathrm{DPO}}(y|x)}{\pi^{\mathrm{SFT}}(y|x)}$ is the reward implicitly defined by the DPO model π^{DPO} and the SFT (reference) model π^{SFT}.

LLaMA Factory's Implementation

Inspired by PPO-ptx [3], we propose mixing the fine-tuning gradients into the DPO training, dubbed **DPO-ftx**, to mitigate the catastrophic forgetting.

A factor γ is added to control the ratio of fine-tuning gradients, the final loss function can be formulated as:

$$\mathcal{L}_{\mathrm{DPO-ftx}}(x, y^c, y^r) = \mathcal{L}_{\mathrm{DPO}}(x, y^c, y^r) + \gamma \mathcal{L}_{\mathrm{SFT}}(x, y^c) \tag{8}$$

Function Calling

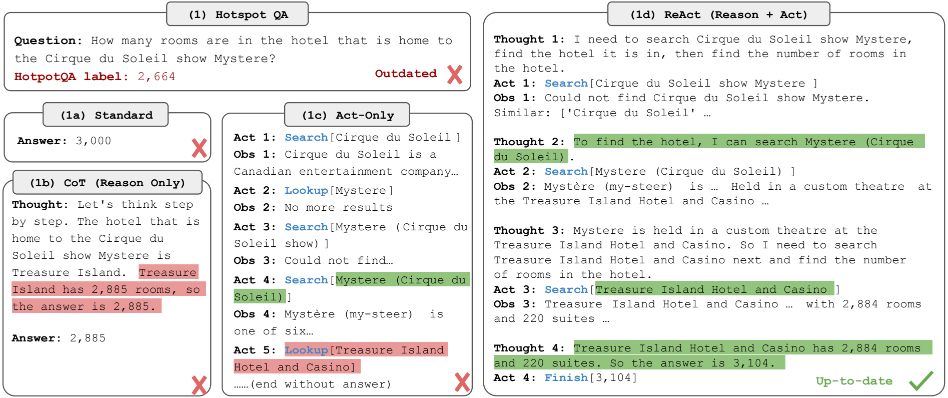


Figure: Enabling LLMs to interact with external tools by templating their output (*e.g.*, using a JSON schema). [9]

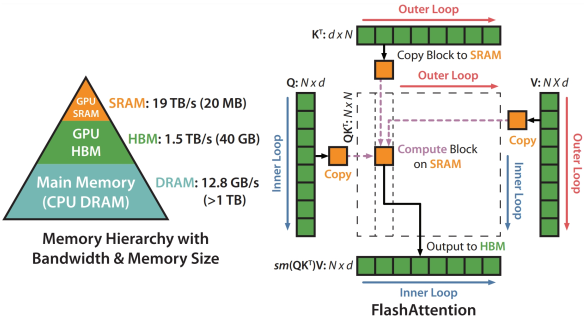
LLaMA Factory's Implementation

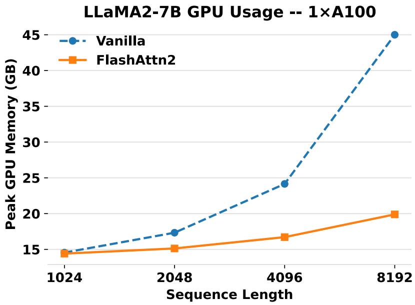
Table: Chat template for aligning LLM with function calling.

System: You are a helpful Assistant. You have access to the following tools: 〉 Tool Name: get_current_weather Tool Description: Get the current weather in a given location Tool Args: - location (string, required): The city and state, e.g. San Francisco, CA - unit (string): The unit, should be one of [celsius, fahrenheit] Use the following format to answer the question: ``` Action: the action to take, should be one of [get_current_weather] if using a tool. Action Input: the input to the action, in a JSON format representing the kwargs (e.g. ```{"input": "hello world", "num_beams": 5}```). ```
Human: What is the weather like in Boston?
Assistant: Action: "get_current_weather"\n Action Input: "{ \"location\": \"Boston, MA\"}"
Observation: "{\"temperature\": 22, \"unit\": \"celsius\", \"description\": \"Sunny\"}"
Assistant: The weather in Boston is currently sunny with a temperature of 22 degrees Celsius.

Flash Attention: Fast and Memory-Efficient Attention

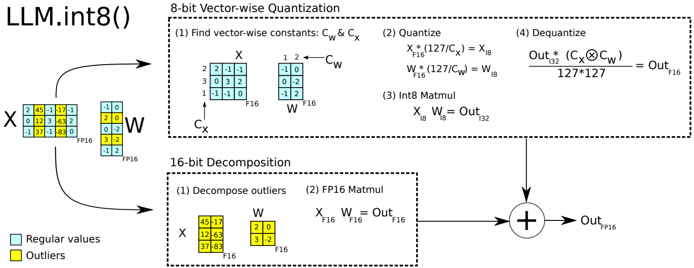
Compared to PyTorch attention, Flash Attention achieves $2 \sim 3\times$ speed-up and linear memory usage with respect to sequence length.

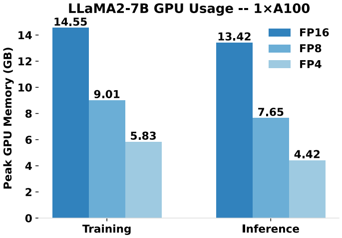




Model Quantization

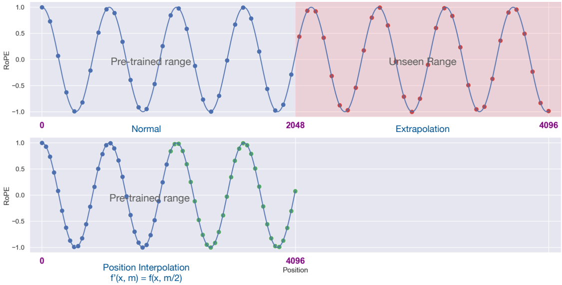
Through LLM.int8 [10] or GPTQ [11] quantization, we could train LLMs in 4 bits with Low-Rank Adapters (LoRA).

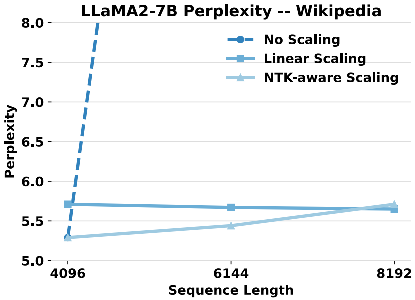




RoPE Scaling: Extend the Context Length of LLMs

Positional interpolation [12] or NTK-aware RoPE scaling extends the context length of pre-trained models.





Efficient LoRA Implementation

Recall the LoRA forward pass $Z = WX + BAX$ and the chain rule, we have the derivative of A.

$$dA = X^T_{(m,d)} \times dZ_{(m,h)} \times B^T_{(h,r)} \quad (9)$$

where m is sequence length, r is the LoRA rank, h is the MLP intermediate size and d is the attention dimension.

In PyTorch's autograd, it will compute $(X^T_{(m,d)} \times dZ_{(m,h)}) \times B^T_{(h,r)}$, where $X^T_{(m,d)} \times dZ_{(m,h)}$ takes $m \times d \times h$ FLOPs and $(X^T_{(m,d)} \times dZ_{(m,h)}) \times B^T_{(h,r)}$ takes $h \times r \times d$ FLOPs, totally $(h \times d)(m + r)$ FLOPs.

Efficient LoRA Implementation

In LLaMA Factory, we compute $X^T_{(m,d)} \times (dZ_{(m,h)} \times B^T_{(h,r)})$, where $dZ_{(m,h)} \times B^T_{(h,r)}$ takes $h \times m \times r$ FLOPs and $X^T_{(m,d)} \times (dZ_{(m,h)} \times B^T_{(h,r)})$ takes $m \times d \times r$ FLOPs, totally $(h + d)(m \times r)$ FLOPs.

Assuming that $m = 2048, r = 16$. In LLaMA 7B, $h = 11008, d = 4096$. We find our implementation getting a speed-up ratio

$$\frac{(m \times r)(h + d)}{(m + r)(h \times d)} \approx 0.5\% \tag{10}$$

Consequently, we reduce the number of FLOPs to 0.5% via manual backward, achieving better LoRA efficiency.

Training Stability

[13] suggests computing attention scores with float32 softmax to avoid overflow.

$$\mathrm{softmax}\left(\frac{Q_i K_i^\top}{\sqrt{d}}\right) = \mathrm{softmax}\left(\left(\frac{Q_i K_i^\top}{\alpha\sqrt{d}} - \max\left(\frac{Q_i K_i^\top}{\alpha\sqrt{d}}\right)\right) \times \alpha\right) \tag{11}$$

$$= \mathrm{FP16}\left(\mathrm{softmax}\left(\mathrm{FP32}\left(\frac{Q_i K_i^\top}{\alpha\sqrt{d}}\right) \times \alpha\right)\right) \tag{12}$$

Bfloat16 prevents from overflow in model training, but a bf16 model cannot switch back to fp16 training.

train a fp16 model with bf16	☺
train a bf16 model with fp16	☹

The Future of LLaMA Factory

- Improving alignment with offline-to-online RL.
- Implementing 3D parallelism (pipeline parallelism, tensor parallelism).

Code: `https://github.com/hiyouga/LLaMA-Factory`

Paper: `https://arxiv.org/abs/2403.13372`

References

Jiawei Zhao, Zhenyu Zhang, Beidi Chen, Zhangyang Wang, Anima Anandkumar, and Yuandong Tian.
Galore: Memory-efficient llm training by gradient low-rank projection.
arXiv preprint arXiv:2403.03507, 2024.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer.
Qlora: Efficient finetuning of quantized llms.
Advances in Neural Information Processing Systems, 36, 2023.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al.
Training language models to follow instructions with human feedback.
Advances in Neural Information Processing Systems, 35:27730–27744, 2022.

References

Mario Michael Krell, Matej Kosec, Sergio P Perez, and Andrew Fitzgibbon.
Efficient sequence packing without cross-contamination: Accelerating large language models without impacting performance.
arXiv preprint arXiv:2107.02027, 2021.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al.
Transformers: State-of-the-art natural language processing.
In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45, 2020.

References

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov.
Proximal policy optimization algorithms.
arXiv preprint arXiv:1707.06347, 2017.

Rui Zheng, Shihan Dou, Songyang Gao, Yuan Hua, Wei Shen, Binghai Wang, Yan Liu, Senjie Jin, Qin Liu, Yuhao Zhou, et al.
Secrets of rlhf in large language models part i: Ppo.
arXiv preprint arXiv:2307.04964, 2023.

References

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D Manning, and Chelsea Finn.
Direct preference optimization: Your language model is secretly a reward model.
Advances in Neural Information Processing Systems, 37, 2023.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao.
React: Synergizing reasoning and acting in language models.
In *International Conference on Learning Representations*, 2023.

References

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.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh.
Gptq: Accurate post-training quantization for generative pre-trained transformers.
In *International Conference on Learning Representations*, 2023.

Shouyuan Chen, Sherman Wong, Liangjian Chen, and Yuandong Tian.
Extending context window of large language models via positional interpolation.
arXiv preprint arXiv:2306.15595, 2023.

References

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.

青稞社区

青年AI研究员Idea加油站，转行码农的新能源充电。

关注公众号
获取更多干货文章

青稞知识社区
观看更多精彩视频

添加小助手
加入青稞社群

青稞技术社区主页：https://qingkelab.github.io

青稞知识社区主页：https://qingkelab.github.io/talks