LoRA/QLoRA: Profiling adapters for Language and Diffusion models

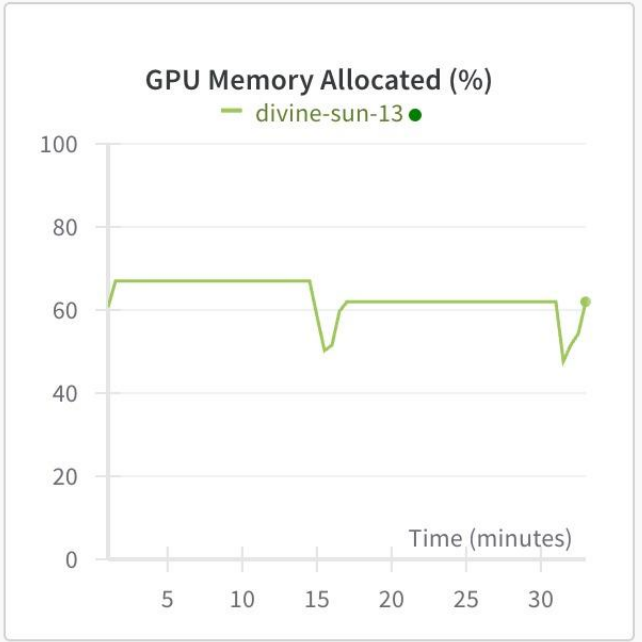
Laurent (lvb243), Pranav (pt2310), Yiyan (yn2161)

Table of Contents

- Project description + Motivation
- Model details
- Related Work
- Experiments + Results (plus visualization)
- Discussion + Future potential work

Project Description

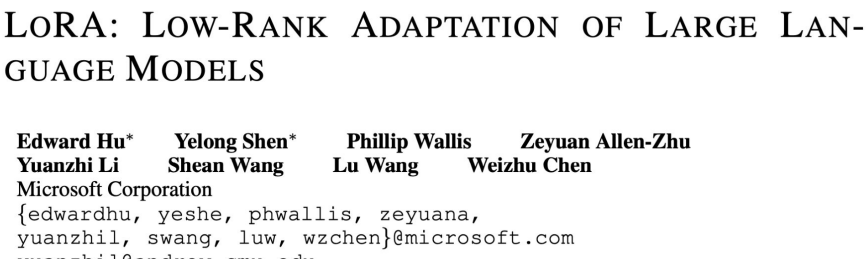
- Fine-tuning pre-trained models is a ubiquitous use case
- Can we use LoRA/QLoRA (and other adapters) to fine-tune these models on commodity hardware?
- Profile memory usage of fine-tuning for two model architectures: Llama, and Stable Diffusion



Motivation

- LoRA(2021) and QLoRA(2023) are recent papers aimed at reducing the memory footprint of fine-tuning pretrained models
- The premise: Less resource consumption when fine-tuning large models, smaller and medium models can be run on commodity hardware
- While maintaining competitive performance on benchmarks
- Both papers focus on language in their results - but there is also increasing demand for fine-tuning generative image models

---



LoRA: Low-Rank Adaptation of Large Language Models

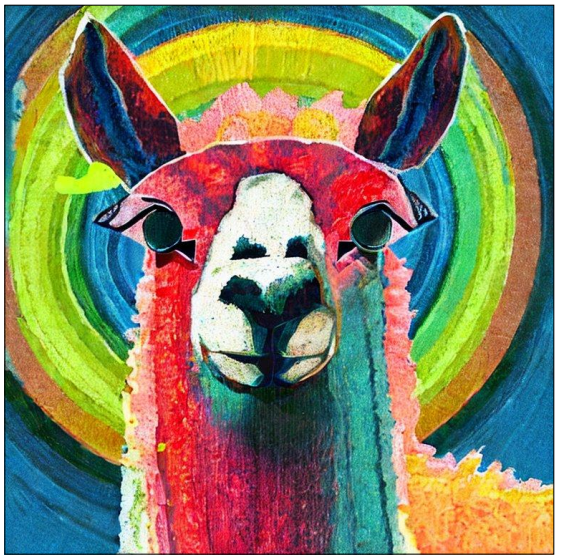
Edward Hu* **Yelong Shen*** **Phillip Wallis** **Zeyuan Allen-Zhu**
Yuanzhi Li **Shean Wang** **Lu Wang** **Weizhu Chen**
Microsoft Corporation
{edwardhu, yeshe, phwallis, zeyuana,
yuanzhil, swang, luw, wzchen}@microsoft.com

Motivation

- Generative image models are generally smaller than generative language models
  - Memory is still a bottleneck for finetuning
- Newer generative image models still scale size
  - (e.g. Stable Diffusion XL)
- Often use a mixture of Transformer-based and other parts
- Want to see if they will see similar memory efficiency gains as LLMs.

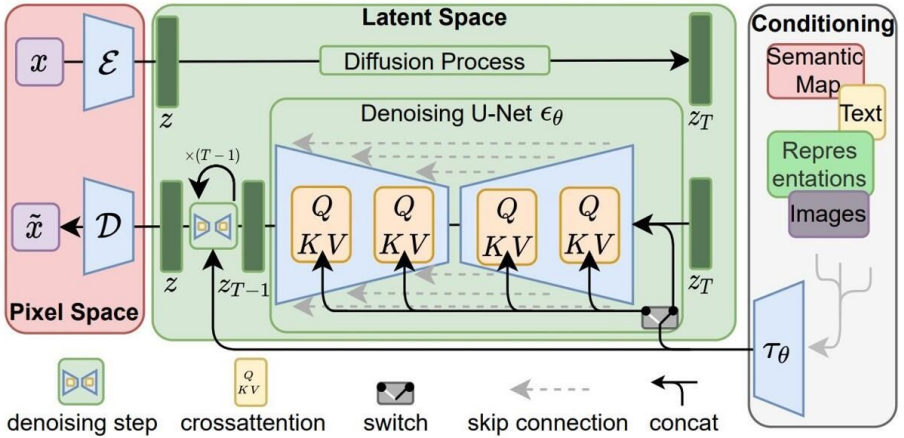
Large Language Model: LLaMA

- Meta’s open source large language model
  - Available in different sizes:
  - 7B, 13B, 33B, and 65B
- 13B model outperforms GPT3 on most benchmarks
- Transformer architecture trained on publicly available data sources
- Llama-7b: ~28Gb to load the model, ~84Gb to finetune



Text-to-image: Stable Diffusion

- Smaller model:
  - 860M parameters in the UNet, 123M parameters in the Text Encoder
- Uses an iterative denoising conditioned on input text
- Architecture:
  - UNet
  - Text Encoder (Transformer-based)
  - Conditioning cross-attention



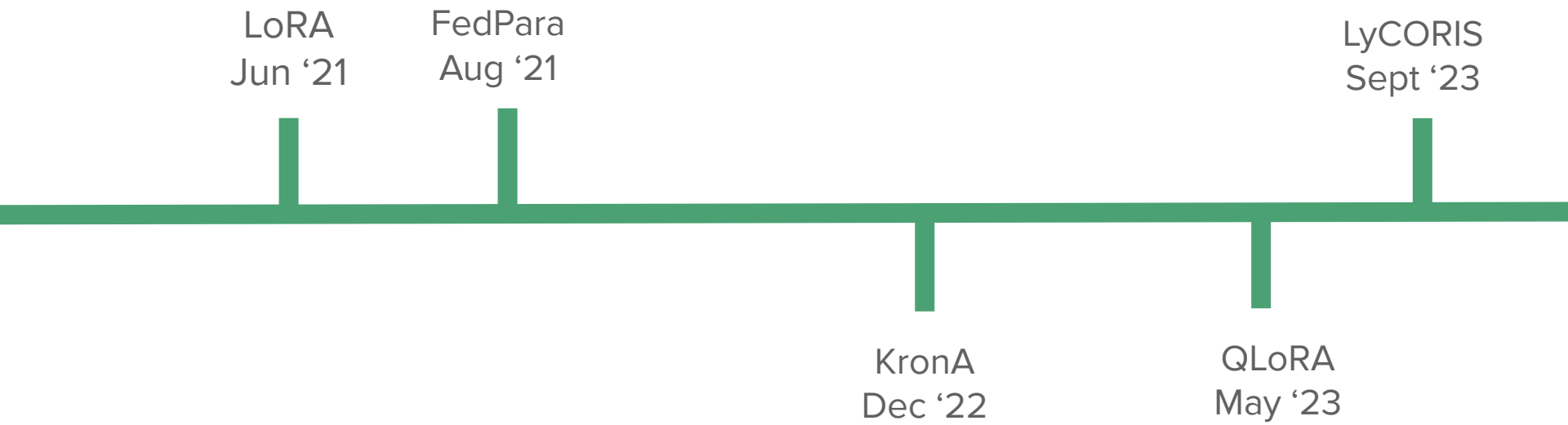
Stable Diffusion

- Iterative denoising process means weights are reused
  - So e.g. offloading strategies are not effective for memory optimization
- Transformer-based text encoder and conditioning cross-attention can benefit from any optimization for transformers
- ~5Gb to load the model, >30Gb to finetune the model

“Ramen monster in antarctica”

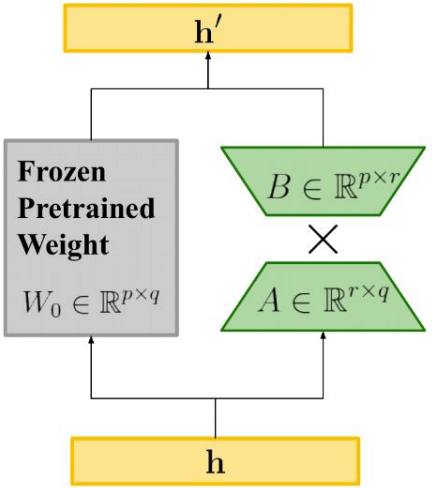
Related Work

Timeline


LoRA
Jun ‘21
FedPara
Aug ‘21
LyCORIS
Sept ‘23
KronA
Dec ‘22
QLoRA
May ‘23

LoRA / QLoRA (Recap)

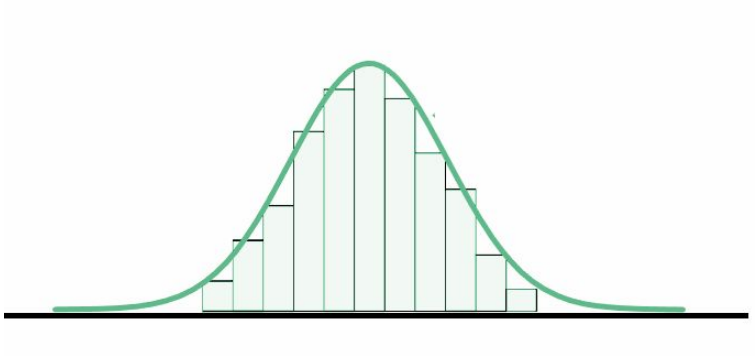
- Decomposes original weight matrix into 2 lower ranked smaller matrices (adapters)
- Only optimizes LoRA layers
- Up to 3x more memory efficient



- Easily merged with frozen weights during deployment

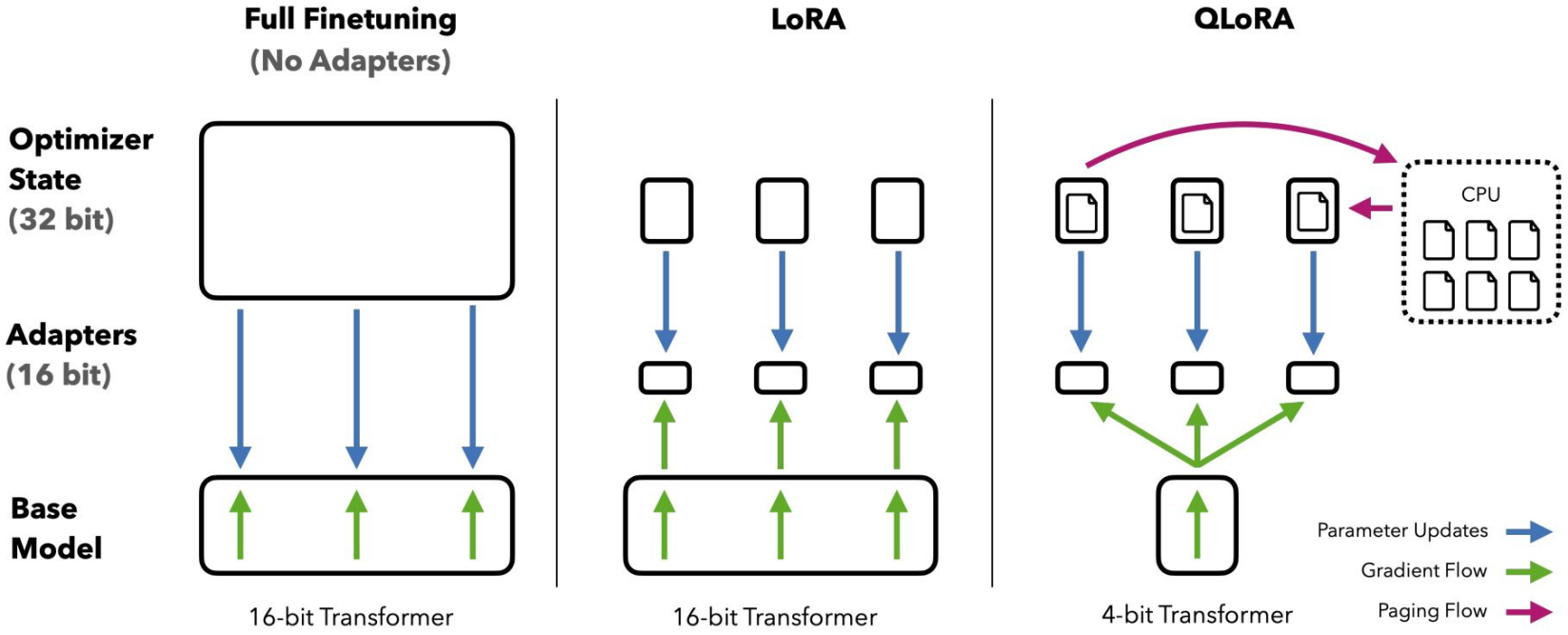
$$h = W_0 x + \Delta W x = W_0 x + BAx$$

QLoRA (recap)



- 4-bit NormalFloat (NF4)
  - Normally distributed bins
- Double Quantization:
  - Quantize quantization constants used to quantize weights
  - 0.5 bits -> 0.127 bits per param
- Paged Optimizers: Transfers to cpu when GPU out of memory
  - (paged offloading)
- **Pretrained model uses NF4 while LoRA layers kept in original precision**

LoRA / QLoRA (quick recap)



Other adapter techniques

LoKr (KronA)

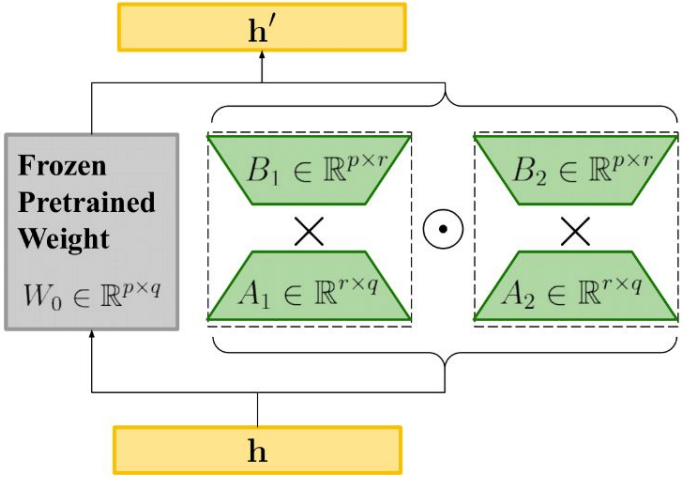
LoHa (FedPara)

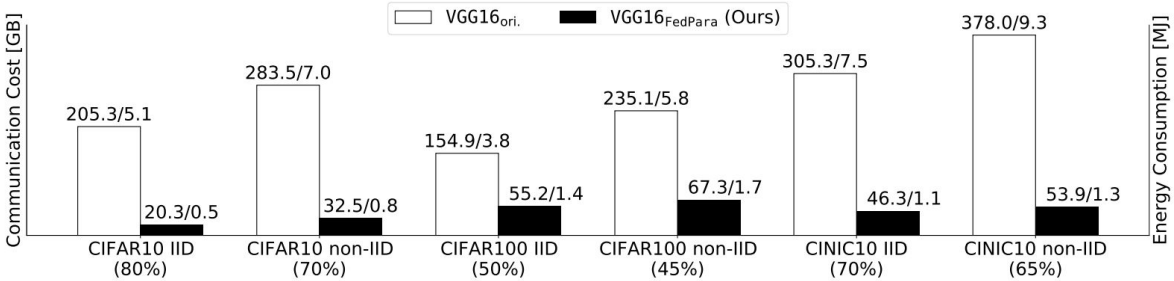
LoCon

LoHA (FedPara)

- Originally designed for federated learning, uses Hadamard product to form the new matrix
- Uses minimal parameter property achievable to **full-rank** weight matrix

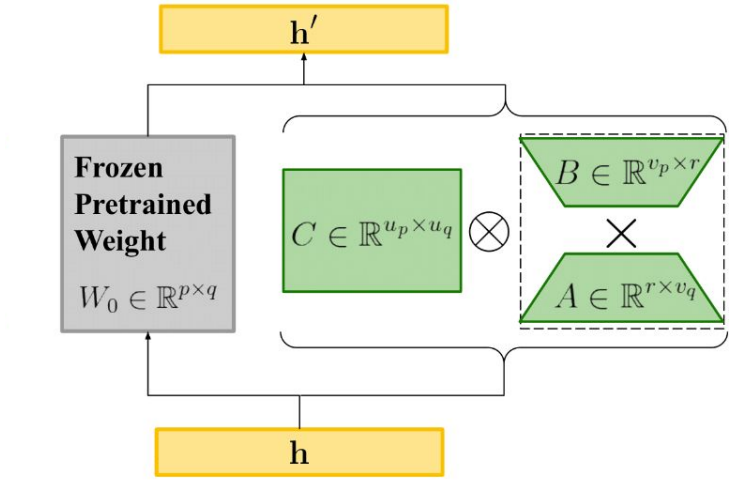
$$\mathbf{W} = (\mathbf{X}_1\mathbf{Y}_1^\top)\odot(\mathbf{X}_2\mathbf{Y}_2^\top)$$





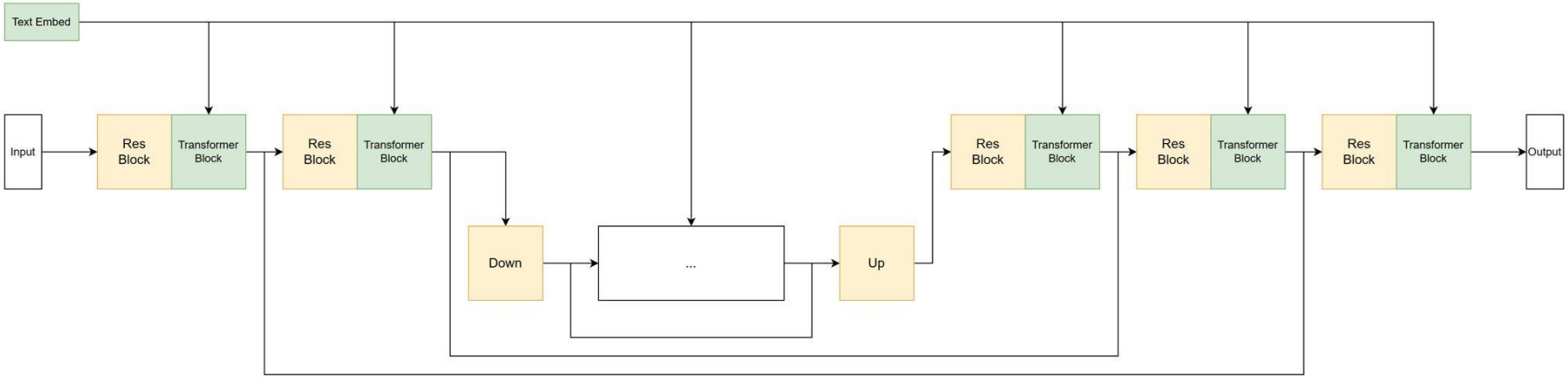
LoKr (KronA)

- Compression designed for performance and does not have low-ranked assumptions
- Outperforms low-ranked factorization methods
- 25% training speed up (no metrics on memory in the paper)



$$\mathbf{W}_{\text{tuned}} = \mathbf{W} + s[\mathbf{A}_k \otimes \mathbf{B}_k]$$

LoCon



- Original LoRA only applies the adapters and finetuning to transformers (green blocks)
- But why not also apply it to the UNet?

HuggingFace 🤗 Diffusers

- Implement LoRA/QLoRA and setup scripts
- Implement all 3 of the described adapter types based on the LyCORIS paper

Experiments & Results

LLaMA-7b

Stable Diffusion

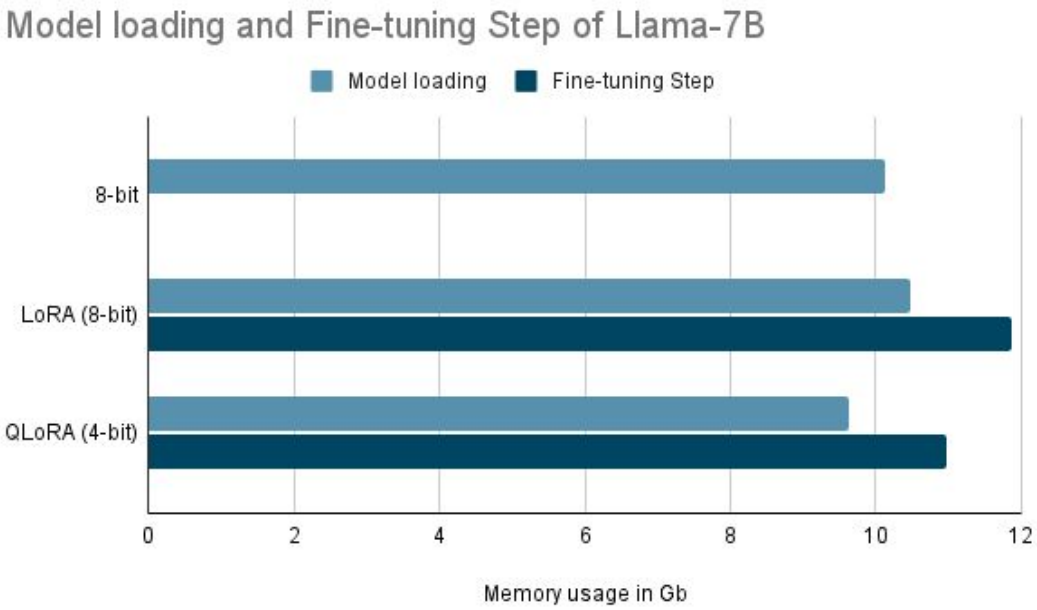
Experiment Setup

- Nvidia V100 GPU (16384MB Memory)
- HuggingFace 🤗 implementation of Diffusers (and pretrained weights)
- Representative runs on fine-tuning for memory profiling
- A longer run for seeing fine-tuning results

- HuggingFace 🤗 test dataset (Pokemon with generated descriptions)

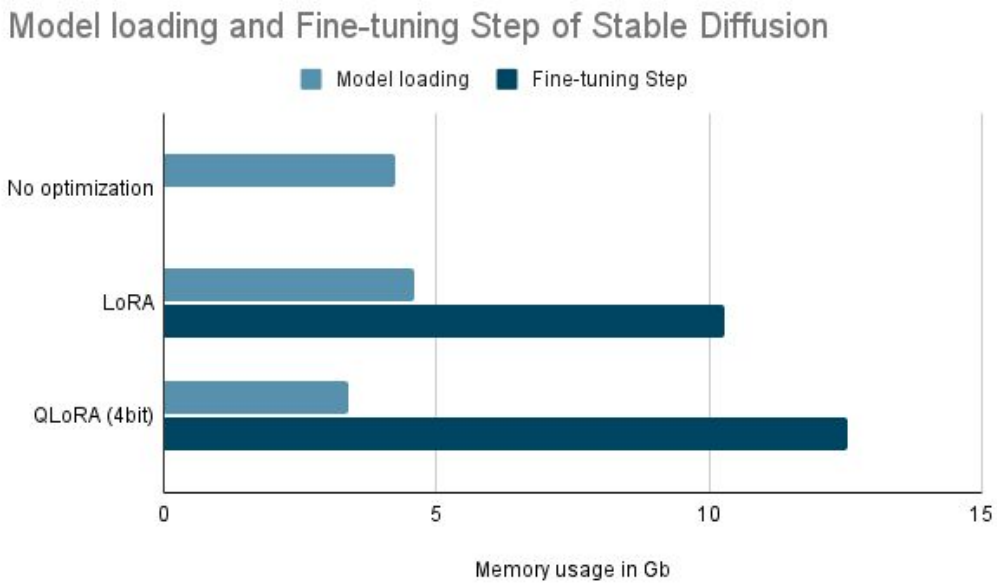
	a red and white ball with an angry look on its face

Llama-7b Results



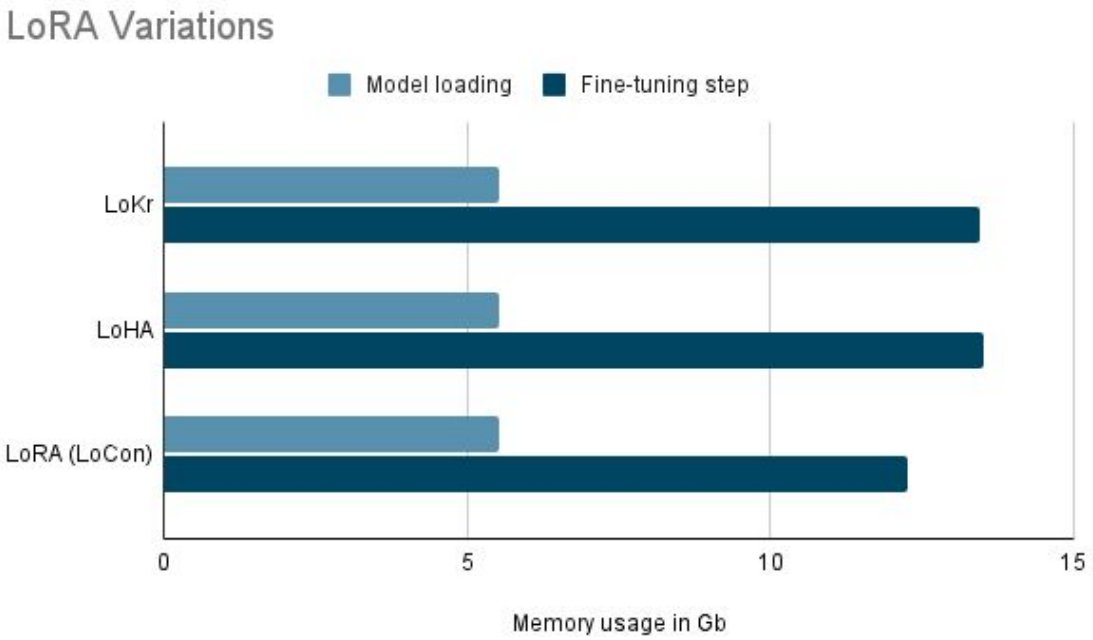
	After model creation (GB)	After 1 step (GB)
Quantization only (8bit)	10.124 (60%)	OOM
With LoRA (8bit)	10.476 (63%)	11.870 (71%)
With QLoRA (4bit)	9.616 (57%)	10.976 (66%)

Stable Diffusion LoRA/QLoRA Results



	After model creation	After 1 step
No optimizations	4.238 (26.4%)	OOM
LoRA	4.577 (27.9%)	10.281 (62.7%)
QLoRA (4bit)	3.389 (20.7%)	12.539 (76.5%)

LoRA Variations (LoCon) Results



	After model creation	After 1 step
LoKr	5.542 (32%)	13.456 (81%)
LoHA	5.540 (32%)	13.512 (81%)
LoRA (LoCon)	5.540 (32%)	12.252 (73%)

Visualizations

Discussion

Discussion

- Successfully loaded and fine-tuned of Stable Diffusion and Llama-7b
- Results seem somewhat ambivalent: regular quantization provides gains on a similar scale - advantages of LoRA in benchmark performance?
- Quantization-only LLM uses less memory to load the model, possibly due to lack of LoRA layers.
    - After 1 step however, large number of trainable params/gradients result in a lot of mem used

Discussion

- We can see that the fine-tuning memory usage makes a larger difference for image generation
- Interesting that even though image generation models are smaller than LLMs, a lot of adapter implementations are coming from the image generation community

Challenges

- Dealing with data types and how they interact with specific hardware
- Applying LLM-oriented libraries and tools in non-LLM settings
- Accessing compute resources to run larger model baseline profiling

Possible further work

- LoRA/QLoRA promise competitive performance on benchmarks - here we focused on Memory profiling. How can we best evaluate potential image quality degradation?
- Other adapter-based methods (LoHa, LoKr) have similar memory usage. Differences between them on model output should be evaluated (eg. LoKr focuses on maintaining fidelity compared to LoRA)
- What is the largest model we can fine-tune with hyperparameter tuning?

Questions?