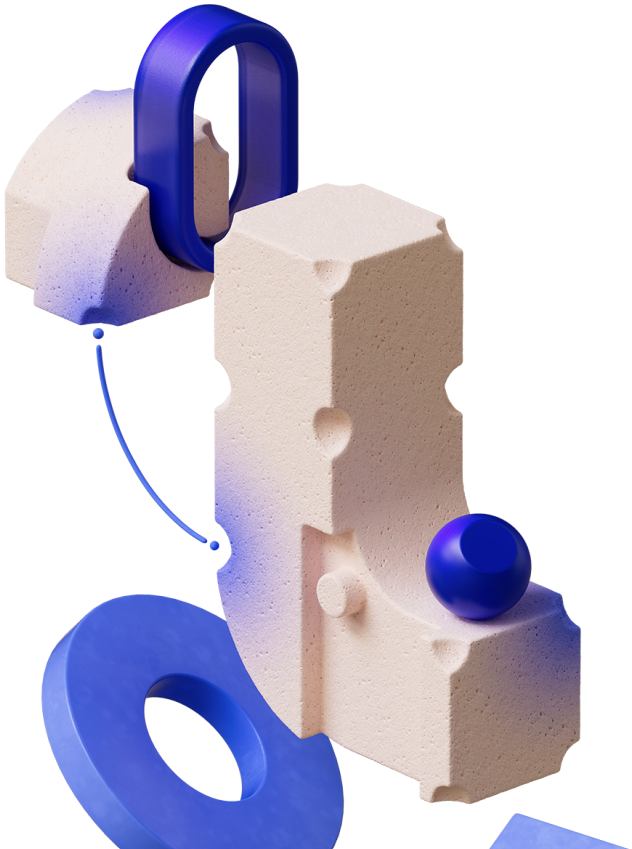
Google DeepMind



Introduction to Post-training

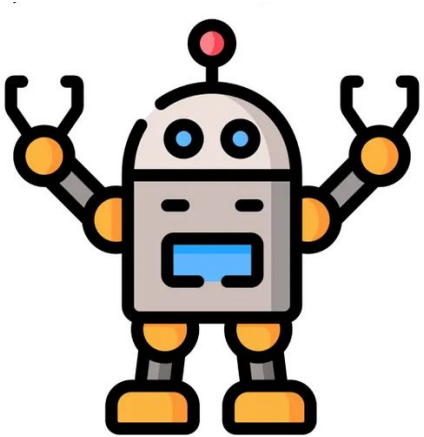
Nikita Saxena
Research Engineer

Quick Recap

We have a Pre-trained Base Model

- Vast general knowledge.
- Understands language, grammar, facts, some reasoning.

But... it's not yet a helpful, aligned assistant. It's more like a very knowledgeable text completion engine.



The Goal of Post-training

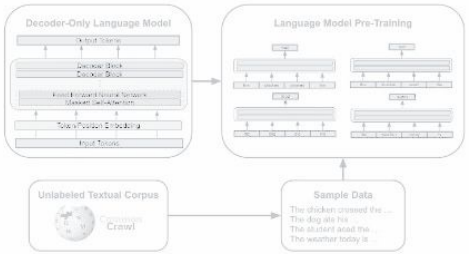
- Align the model with human intent and preferences.
- Teach it to follow instructions, answer questions, and engage in useful dialogue.

Post-training Stages We'll Cover

- Supervised Fine-Tuning (SFT)
  - Parameter-Efficient Fine-Tuning (PEFT) (like LoRA, Prompt Tuning)
- Reinforcement Learning from Human Feedback (RLHF)

Alignment

Pre-Training



SFT

A prompt is sampled from our prompt dataset.

A labeler demonstrates the desired output behavior.

This data is used to fine-tune GPT-3 with supervised learning.

RLHF

A prompt and several model outputs are sampled.

A labeler ranks the outputs from best to worst.

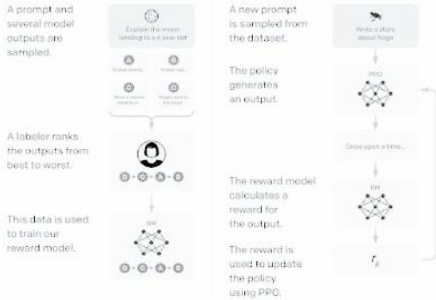
This data is used to train our reward model.

A new prompt is sampled from the dataset.

The policy generates an output.

The reward model calculates a reward for the output.

The reward is used to update the policy using PPO.



Agenda

Data

Data Flywheel

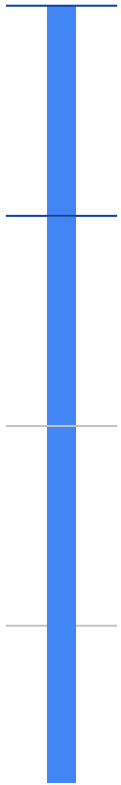
Why?

Seed data (initial dataset used to start training or fine-tuning a language model) alone is insufficient

Generating quality synthetic prompts requires extensive and diverse seed data.

Real user interactions differ from synthetic data:

Actual user queries, particularly in multi-turn conversations, are highly varied and unpredictable. Synthetic multi-turn dialogues often assume ideal conditions—such as users consistently accepting and following the model's responses.



Collect user prompts from a recent time period

Filter them with heuristic rules for quality, privacy, etc.

Label or generate high-quality answers using a more powerful model (e.g., GPT-4o).

Collect the resulting (prompt, answer) pairs

Data Synthesis

Goal

Ensure prompt diversity using various synthesis methods. Dataset collection in the order of thousands.

Prompt Synthesis

1. **Self-Instruct Methodology**: Apply task identification by tagging each task with task_type labels.
2. **Heuristic-Based Synthesis**: Perform rewriting of data through rule-based transformations and high-capacity LLMs. Generate variations in multiple formats (e.g., text, markdown, JSON)

Answer Synthesis

1. **Powerful model is all you need**: For generating high-quality answers, powerful models (e.g. GPT-4) remain the gold standard.
2. **Specialized Data**: includes specific datasets for: Long-context texts, agent or function-calling interactions, Retrieval-Augmented Generation.

Training

2

Understanding and Diagnosing Loss

Channel-Specific Loss Monitoring

1. If training on diverse task_types, monitor loss separately for each "channel" (task category).
2. Why? Different tasks (e.g., summarization vs. simple Q&A) exhibit different loss behaviors. Separate tracking aids diagnostics.

Special Token Loss

1. Tokens like <bos>, <eos>, <pad> might initially have high loss.
2. Typically drops quickly as the model learns their structural role.

Creative vs. Fixed-Answer Loss

1. Creative Tasks (e.g., story writing): Inherently higher loss due to many valid responses.
2. Fixed-Answer Tasks (e.g., factual Q&A): Typically lower loss as "correctness" is more defined.

Understanding and Diagnosing Loss

Expected Loss Ranges for General-Purpose Data (varies by model & data)

1. Initial Loss
    a. (7B/13B models): ~2.0
    b. (e.g., ~70B models): ~1.0 - 2.0.
2. Converged Loss : ~0.5 .
3. Caution: Very low loss (<0.2-0.3) might indicate overfitting.

Diagnosing Increasing Loss

1. Consistent Increase? Unlikely a "data is too hard" problem. LLMs can memorize
2. Increasing loss usually points to training code issues (bugs, hyperparameter misconfiguration).
3. Even with random noise, loss should plateau, not continuously rise.

Evaluation

How Good is the Assistant

Automated Evaluation (e.g., using LLMs like GPT-4/Claude as judges)

Pros: Scalable, fast.

Cons & Challenges:

1. Results highly sensitive to how the "judge LLM" is prompted.
2. Inherent LLM Biases:
    a. Positional bias (e.g., favoring option A).
    b. Length bias (favoring longer answers).
    c. Inconsistent scoring (e.g., same answer gets 3, 4, or 5 on different trials).
3. Mitigation Tips:
    a. Reference well-designed prompts (e.g., AlignBench).
    b. Include a "gold" reference answer for comparison (S x S Eval).

Human Evaluation

Pros: Gold standard for nuanced judgment. Better at assessing subtle qualities (e.g., tone, creativity).

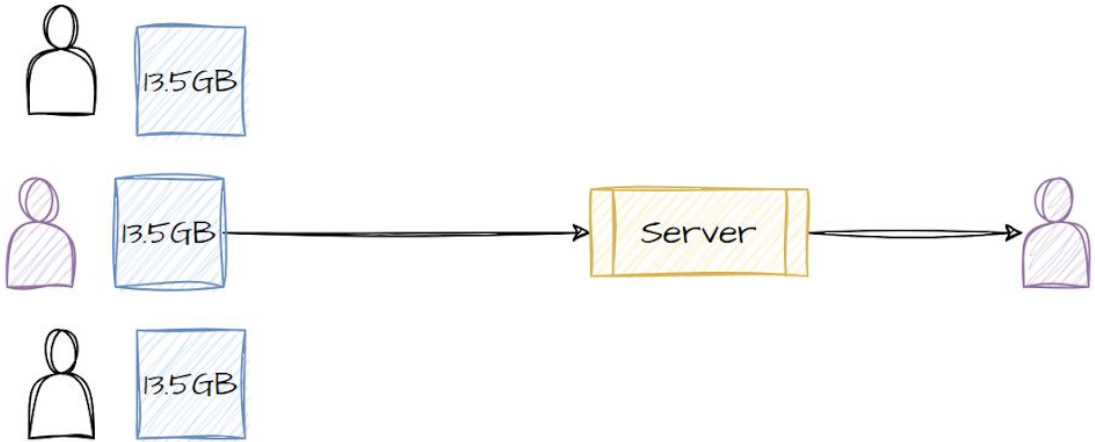
Cons: Slow, expensive, labor-intensive, requires clear rubrics and trained evaluators for consistency.

Parameter-Efficient Fine tuning

I. Overview
II. Prompt Tuning
III. LoRA Finetuning
IV. QLoRA Finetuning

01 Overview

Every User gets their own model



- ✅ High Accuracy
- ❌ Difficult to maintain
- ❌ High Costs
- ❌ High Latency

PEFT only **fine-tunes a small portion of or a small amount of additional model parameters, keeping most of the pre-trained parameters fixed.**

This greatly reduces computational and storage costs, while the most advanced PEFT techniques can achieve performance comparable to full fine-tuning.

Advantages:

- Significantly reduces memory usage and hardware resource requirements
- Faster training and shorter iteration times.
- Lower storage costs, as shared weight parameters can be reused across different tasks
- May even lead to better model performance by alleviating issues related to overfitting

02

Prompt Tuning

Overview

Fine-tune LLM by adjusting prompt templates, learnable continuous embeddings.

Method

- Prepend k learnable prompt tokens (T) to the input text (X).
- Pre-trained model parameters (θ) are FROZEN. **Only prompt token embeddings (θ_T) are updated.**
- No changes to model architecture (encoding/output layers).

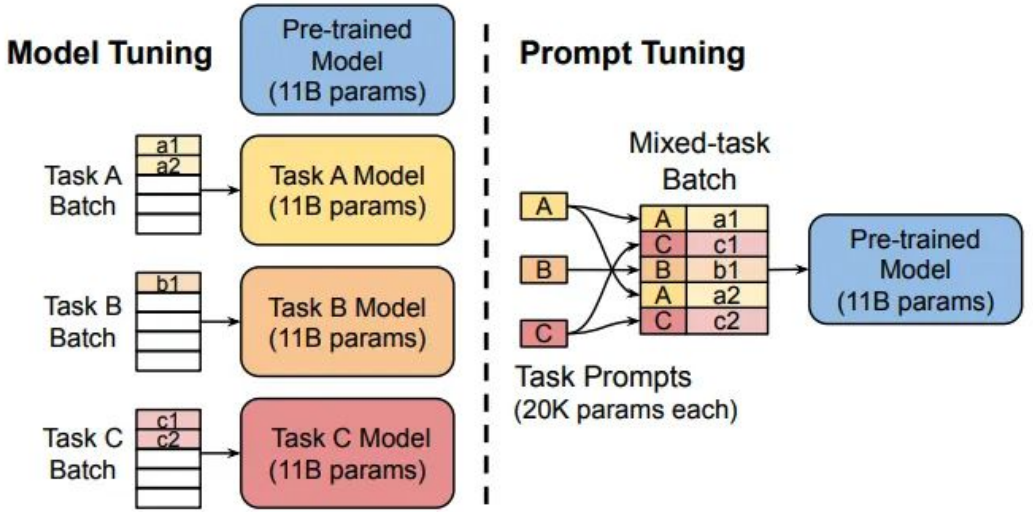
Representation

- Standard Model: $\theta\,(Y \mid X)$
  - θ: Frozen pre-trained model parameters.
- Prompt Tuning: $\theta_T\,(Y \mid [T\,;\,X])$
  - θ_T: Task-specific learnable prompt parameters (optimized during training).

Training Strategy: Prompt Ensembling

- Train multiple distinct "soft prompts" (each with its own θ_T) for the same task.
- These different prompts share the same frozen pre-trained LLM.

Prompt Tuning vs Full Finetuning

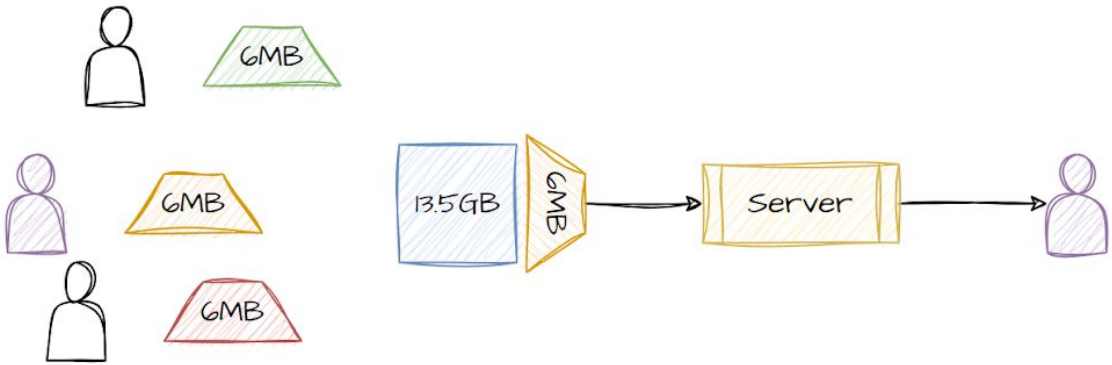


Feature	Full Fine-tuning	Prompt Tuning
Parameters Updated	Most model parameters	Just the prompt token embeddings
Storage per Task	Entire model copy (GBs/TBs)	Small prompt embedding (KBs/MBs)
Task-Specific Models	Separate model per task	Original shared model + task-specific prompts
Computational Cost	High	Very Low

03

LoRA
Fine tuning

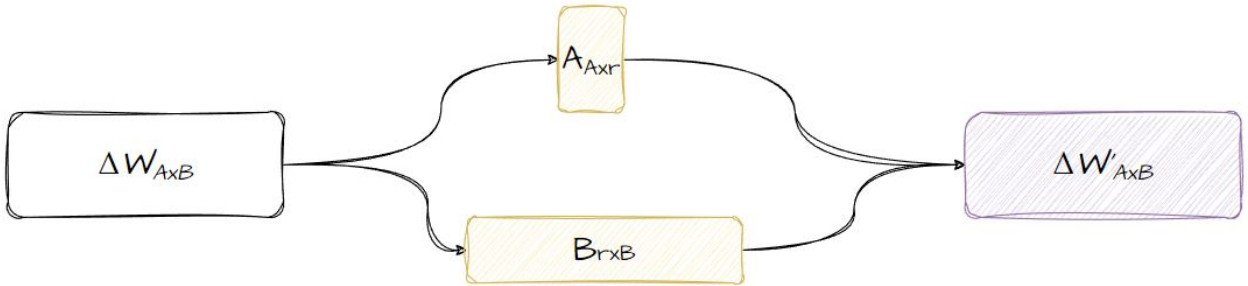
Every User gets their own LoRA Adapter



- [x] High Accuracy
- [x] Easy to maintain
- [x] Low Costs
- [x] Low Latency

How LoRA Works?

Instead of fine-tuning the original large weight matrices (W), learn their **change (ΔW) u**sing low-rank decomposition.



During fine-tuning, with the full weight matrix **W**, we optimize it with formula **W**+Δ**W**, where the update Δ**W** is a low-rank decomposition Δ**W**=**AB a**nd **A**,**B** are low-rank matrices.

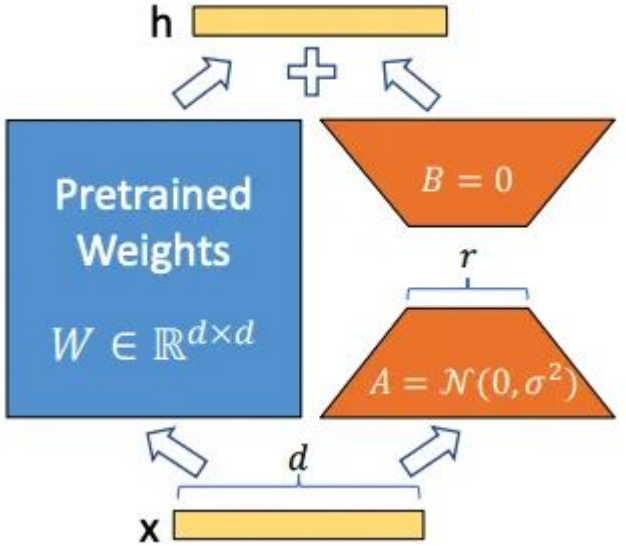
How LoRA Works?

- Pre-trained model weights (W_0) are FROZEN
- For a target layer (e.g., attention query/value matrices), we want to learn an update ΔW.
- LoRA approximates ΔW with two smaller "adapter" matrices: A (d x r) and B (r x k).

$$\begin{cases} W_0 + \Delta W = W_0 + BA \\ W_0 \in \mathbb{R}^{d \times k}, B \in \mathbb{R}^{d \times r}, A \in \mathbb{R}^{r \times k} \\ \text{and} \quad r \ll \min(d, k) \end{cases}$$

- Only parameters in matrices A and B are trained.
- The forward pass is thus transformed into:

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



Why LoRA?

Parameter Efficiency
Only A and B are trained.
$(r * (d+k)$ vs $d * k)$.

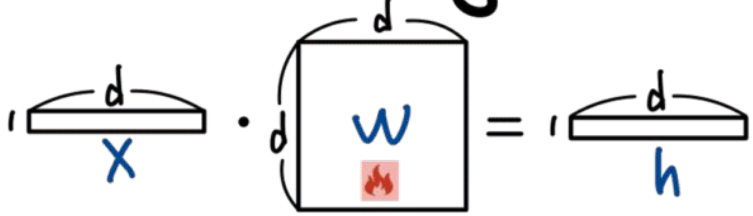
No Increase in Inference Latency
(if merged)
W + BA can be pre-calculated, resulting in a matrix of the original size.

Flexibility
Easy to switch between tasks by swapping LoRA adapters.

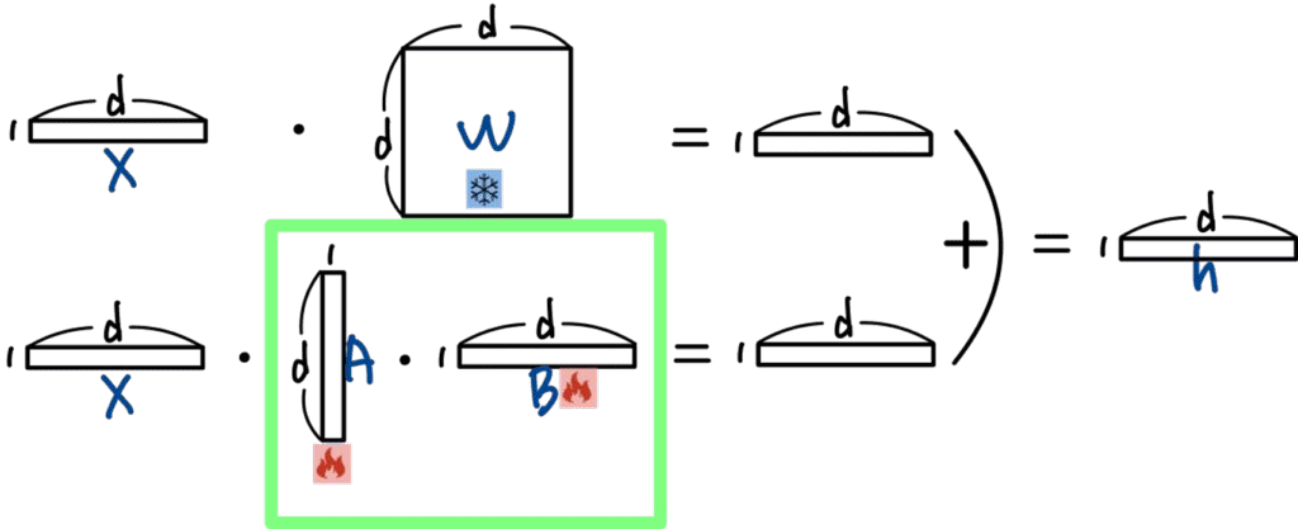
Model Size Remains the Same
(Conceptually)
The new parameters (A, B) are effectively for the training step to learn ΔW. The base model isn't permanently enlarged.

Reduced Checkpoint Size
Only need to save A and B for each task (tiny).

Full Finetuning



LoRA



04

QLoRA Finetuning

How QLoRA Works?

Combines a **high-precision computing technique with a low-precision storage method**. This helps keep the **model size small** while still making sure the model is **still highly accurate**.

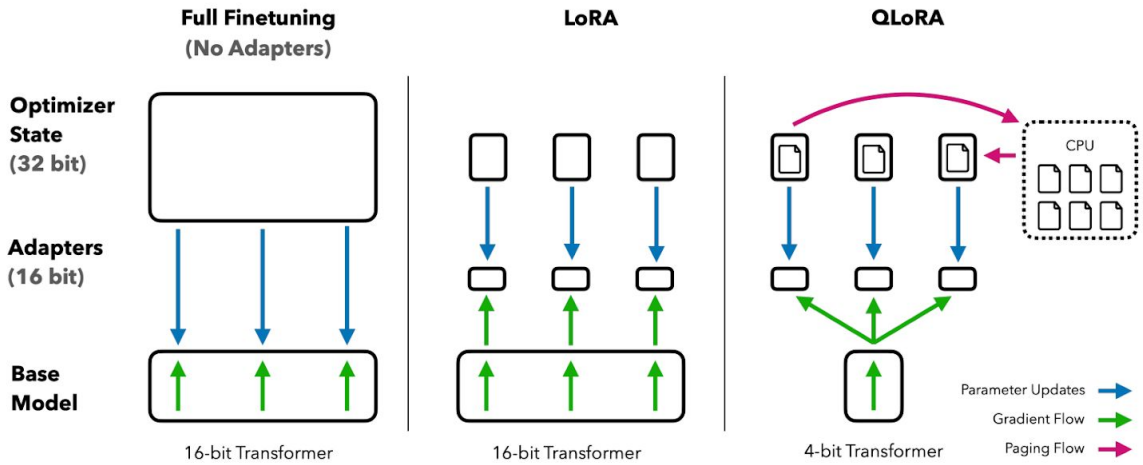


Figure 1: Different finetuning methods and their memory requirements. QLoRA improves over LoRA by quantizing the transformer model to 4-bit precision and using paged optimizers to handle memory spikes.

Quantize the large, frozen pre-trained model weights to a very low precision (e.g., 4-bit) during the LoRA fine-tuning process.

QLoRa Techniques

4-bit NormalFloat (NF4)

New data type, information-theoretically optimal for normally distributed weights. Quantiles of a 0-centered normal distribution are used to create the quantization bins.

Double Quantization

Reduces memory by quantizing the quantization constants themselves.

Memory Reduction

(using NVIDIA unified memory)

Enables fine-tuning massive models (e.g., 65B Llama) on a single consumer/prosumer GPU (e.g., with 24GB-48GB VRAM).

Example: 65B model memory from >780GB (FP16) down to <48GB.

Paged Optimizers

(using NVIDIA unified memory)

Manages memory spikes during gradient checkpointing by offloading optimizer states to CPU RAM when GPU memory is exhausted, paging them back as needed.

Google DeepMind

Let's try out training our own LoRA model now!

Colab Notebook

References

1. [2106.09685] LoRA: Low-Rank Adaptation of Large Language Models
2. [2312.12148] Parameter-Efficient Fine-Tuning Methods for Pretrained Language Models: A Critical Review and Assessment
3. [2305.14314] QLoRA: Efficient Finetuning of Quantized LLMs
4. In-depth guide to fine-tuning LLMs with LoRA and QLoRA
5. 6.6 PEFT | AI Roadmap
6. Improving LoRA: Implementing Weight-Decomposed Low-Rank Adaptation (DoRA) from Scratch

Thank you.

Nikita Saxena
Research Engineer

Google DeepMind