Experts

Paper Summary

LoRA - Low Rank Adaptation of LLMs

Aashi Dutt
GDE AI/ML (Gen AI)
@AashiDutt

Google Developers

Large Language Models

Large Language Models (LLMs) are powerful AI models trained on massive amounts of text data to understand and generate human-like language.

Examples

- BERT
- GPT
- LLAMA
- Gemini
- Gemma
- Lamda
- Mistral
- Phi

Applications

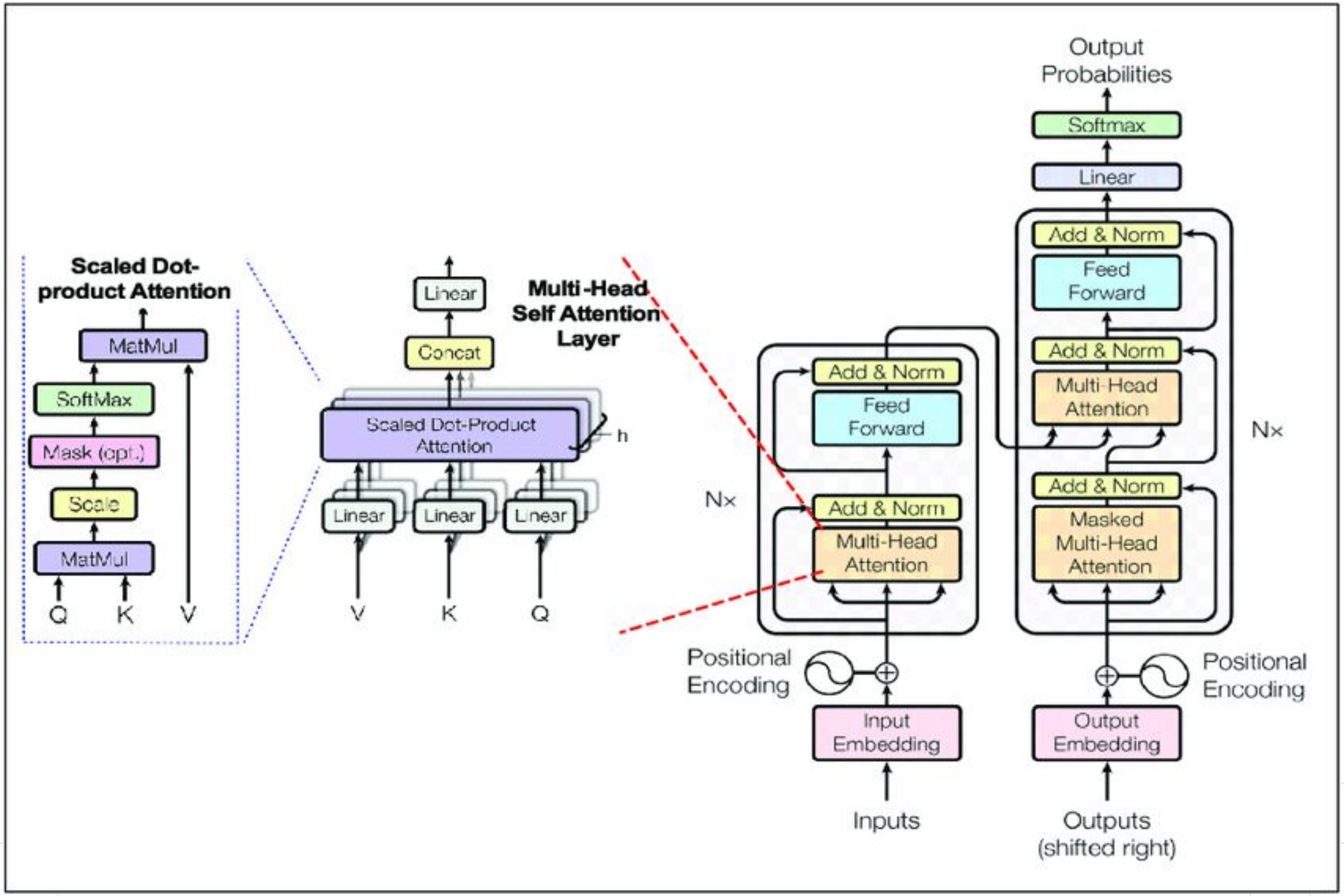
Open Large Language Models (LLMs) have a wide range of applications across various industries and domains.

- **Text generation:** These models can be used to generate creative text formats like poems, scripts, code, marketing copy, email drafts, etc.
- **Chatbots and conversational AI:** Power conversational interfaces for customer service, virtual assistants, or interactive applications.
- **Text summarization:** Generate concise summaries of a text corpus, research papers, or reports.
- Research and education
- **Natural Language Processing (NLP) research:** These models can serve as a foundation for researchers to experiment with NLP techniques, develop algorithms, and contribute to the advancement of the field.
- **Language Learning Tools:** Support interactive language learning experiences, aiding in grammar correction or providing writing practice.
- **Knowledge Exploration:** Assist researchers in exploring large bodies of text by generating summaries or answering questions about specific topics.

Let’s Understand

Transformer Architecture: Large Language Models are primarily based on the transformer architecture, a neural network design introduced in 2017. Unlike previous models that relied on recurrent neural networks (RNNs), transformers are designed to process entire sequences of text in parallel, enabling more efficient learning of long-range dependencies.

Self-Attention Mechanism: The key innovation of transformers is the self-attention mechanism. This allows the model to weigh the importance of different words in a sentence when predicting the next word. Self-attention enables LLMs to capture contextual relationships between words, regardless of their distance within the text.

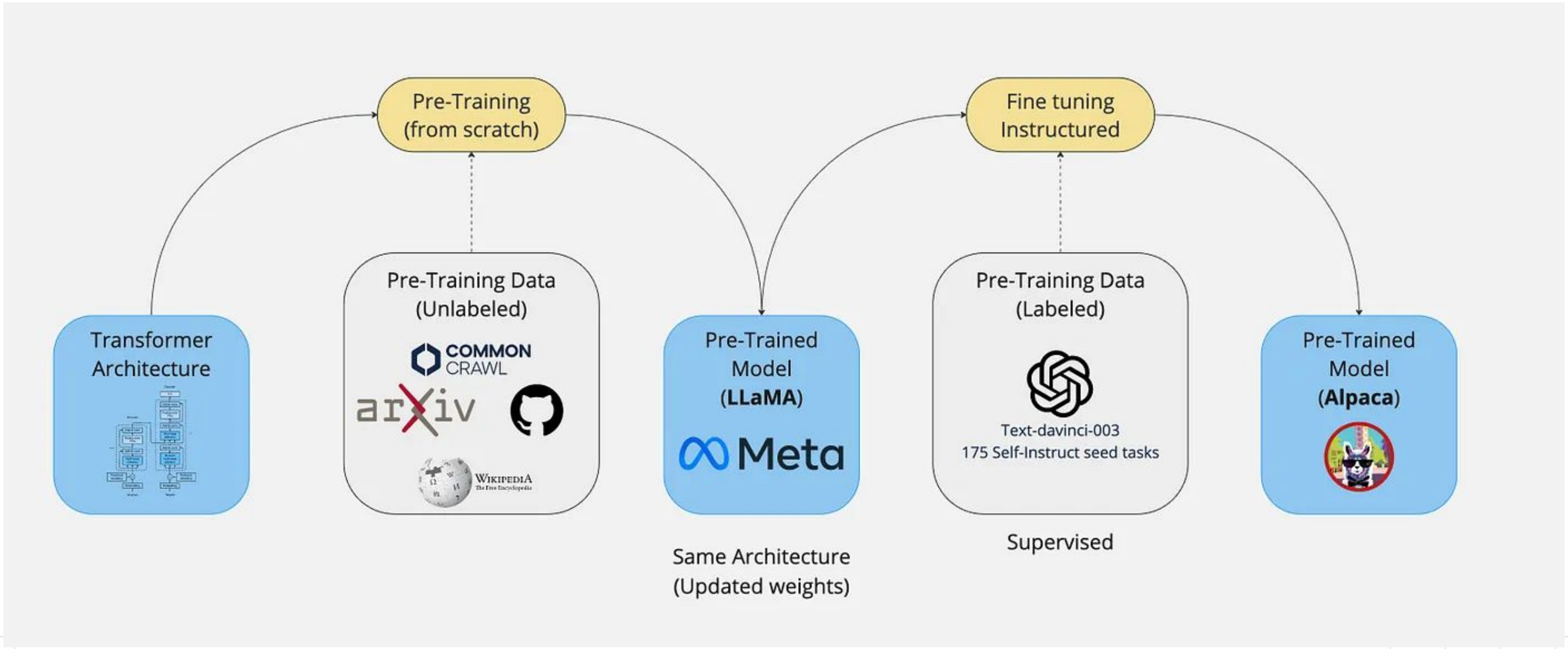

Scaled Dot-product Attention
MatMul
SoftMax
Mask (opt.)
Scale
MatMul
Q
K
V
Linear
Concat
Multi-Head Self Attention Layer
Scaled Dot-Product Attention
h
Linear
Linear
Linear
V
K
Q
Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Add & Norm
Masked Multi-Head Attention
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

Pre-training vs Fine-tuning

LLMs are initially pre-trained on massive datasets of text (often containing billions of words). This pre-training involves tasks like predicting masked words or the next sentence in a paragraph. Through this process, the model learns general language patterns, grammar, facts, and even some reasoning abilities. Pre-training usually would mean take the original model, initialise the weights randomly, and train the model from absolute scratch on some large corpora.

Fine-tuning involves further training the pre-trained LLM on a smaller dataset that is specific to the desired task (e.g., question answering, translation, sentiment analysis). This fine-tuning process allows the model to specialize its knowledge and abilities to perform well on the target task. The pre-trained weights are adjusted slightly to better fit the specific task, such as sentiment analysis, named entity recognition, or domain-specific text classification.

Aspect	Pre-training	Fine-tuning
Objective	Learn general features and patterns from a large and diverse dataset	Adapt the pre-trained model to a specific task or domain
Process	- Initialize model weights randomly - Train from scratch on a vast corpus- Learn general language representations, grammar, common knowledge, and context	- Take the pre-trained model - Train further on a smaller, task-specific dataset - Adjust pre-trained weights to fit the specific task
Result	Model with generalized knowledge that can be adapted to various tasks with further training	Model specialized for specific tasks while retaining general knowledge from pre-training
Techniques	- Extensive training on large datasets	- Task-specific adjustments with a smaller learning rate - Layer freezing for retaining general features while fine-tuning specific ones
Example	- Pre-trained BERT on large corpus like Wikipedia and books	- Fine-tuned BERT on sentiment analysis dataset for movie reviews
Advantages	- Captures broad, generalized knowledge	- Computationally cheaper and faster than training from scratch - Better performance on specific tasks by leveraging pre-trained knowledge
Efficiency	Requires extensive computational resources and time	More efficient as it starts from a pre-trained state
Performance	Provides a strong foundation with broad applicability	Achieves higher performance on specific tasks due to specialization


Pre-Training
(from scratch)
Fine tuning
Instructured
Transformer
Architecture
Pre-Training Data
(Unlabeled)
COMMON CRAWL
arXiv
Wikipedia
Pre-Trained
Model
(LLaMA)
Meta
Pre-Training Data
(Labeled)
Text-davinci-003
175 Self-Instruct seed tasks
Pre-Trained
Model
(Alpaca)
Same Architecture
(Updated weights)
Supervised

Challenges of Fine-tuning

- **Computational Cost:** Full fine-tuning of large language models is computationally expensive, requiring specialized hardware (GPUs or TPUs) and significant time resources.
- **Data Requirements:** Fine-tuning typically needs a large amount of labeled data specific to the task, which might be difficult or costly to obtain.
- **Catastrophic Forgetting:** There's a risk that fine-tuning can cause the model to "forget" some of the general knowledge it learned during pre-training.
- **Overfitting:** LLMs can be prone to overfitting to the fine-tuning data, leading to poor generalization on unseen data.

Why LoRA?

Problem:

An LLM with 175B parameters is exciting but daunting to fine-tune on a GPU. Fine-tuning large language models (LLMs) is computationally expensive and requires vast amounts of data.

Solution:

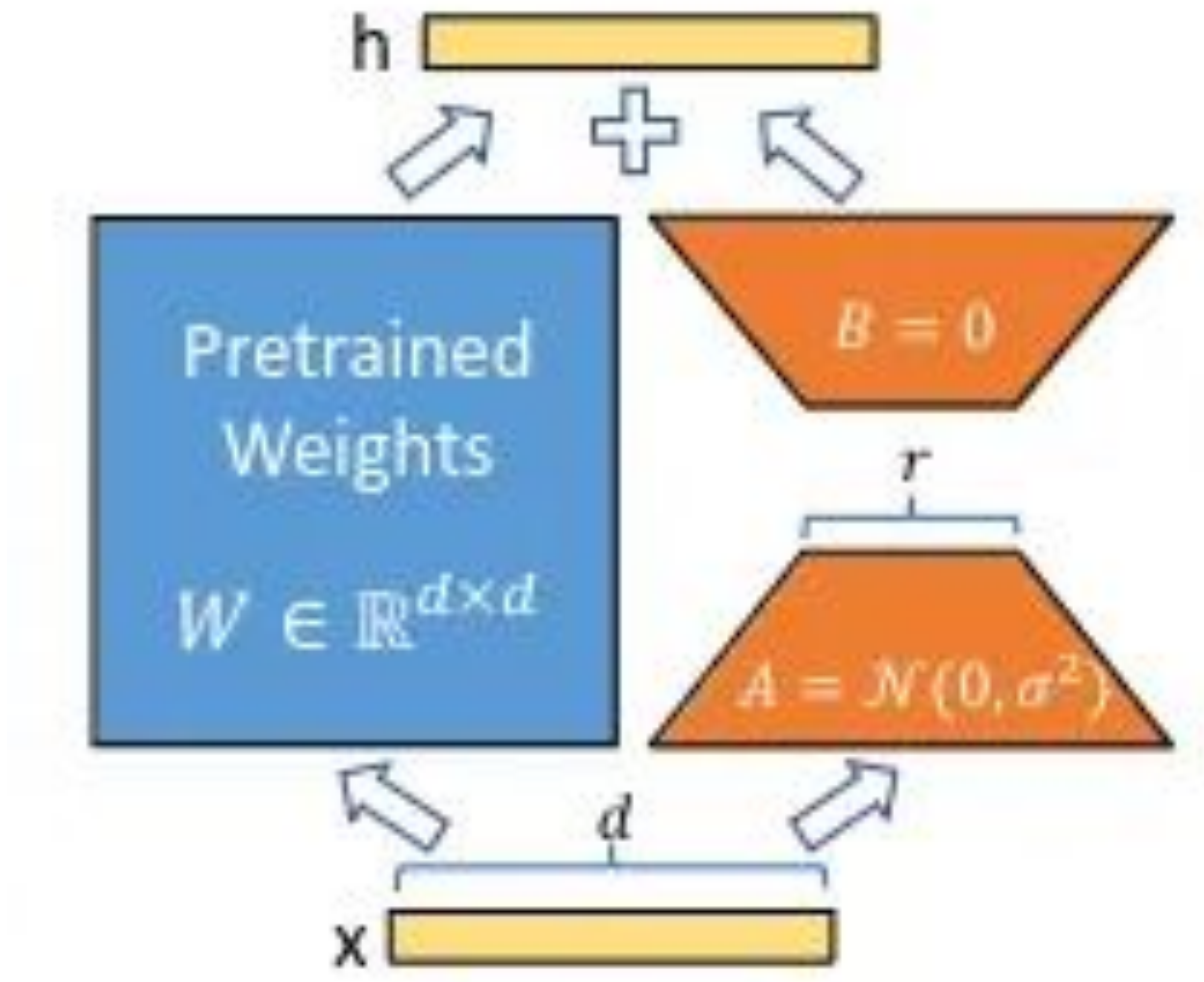
LoRA offers a more efficient alternative by freezing the pre-trained model weights and injecting low-rank matrices into each layer.

LoRA: The Core Idea

- **Low-Rank Decomposition**
- **Injection into Layers**
- **Frozen Pre-trained Weights**

Low-Rank Decomposition: The Heart of LoRA

- **Matrix Factorization:** The core idea is to decompose a large weight matrix (W) into two smaller matrices: A (update matrix) and B (rank decomposition matrix).
- **Matrix Dimensions:** If W has a shape of `d x k`, then:
  - A has a shape of `r x k`, where `r` is the rank hyperparameter and is much smaller than `d`.
  - B has a shape of `d x r`.
- **Approximation:** The product of A and B (i.e., BA) is an approximation of the change in the original weight matrix (ΔW) during fine-tuning.
  - This approximation assumes that the changes in weights during fine-tuning have a low intrinsic rank (meaning they can be represented with fewer dimensions).


h
Pretrained
Weights
$W \in \mathbb{R}^{d \times d}$
$B = 0$
r
$A = \mathcal{N}(0, \sigma^2)$
d
x

Why Low-Rank Decomposition Works?

- **Rank and Information:** The "rank" of a matrix roughly indicates the amount of unique information it contains. A low-rank matrix can be expressed as a combination of a few independent vectors.
- **Hypothesis:** LoRA's hypothesis is that the change in weights during fine-tuning (ΔW) is primarily low-rank. This means most of the changes can be captured by a small set of update vectors.
- **Efficiency:** By decomposing ΔW into A and B, we drastically reduce the number of trainable parameters. This makes fine-tuning much faster and requires less memory.

Injection into Transformer Layers

- **Target Layers:** LoRA is typically applied to the attention heads within the transformer architecture. These are the layers responsible for capturing contextual relationships between words.
- **Freezing Original Weights:** The pre-trained weights of the transformer (W0) are frozen. This ensures that the general knowledge acquired during pre-training is retained.
- **Adding Low-Rank Matrices:** The low-rank matrices A and B are added to the original weights in the attention mechanism. The forward pass becomes: New Output = W0 * X + (B * A) * X
- where X is the input to the layer.
- **Training A and B:** Only the matrices A and B are trained during fine-tuning. This dramatically reduces the number of parameters to optimize.

Preserving Pre-trained Knowledge

- **Frozen Base Model:** By freezing the pre-trained weights, LoRA ensures that the model doesn't lose the general knowledge it gained during pre-training. This is crucial for maintaining performance on a wide range of language tasks.
- **Focused Adaptation:** LoRA allows you to adapt the model to specific tasks or domains by learning a small set of task-specific parameters (A and B) without disrupting the broader language understanding capabilities.

Benefits:

- **Improved Training Efficiency:** Explain how LoRA speeds up training by requiring fewer updates.
- **Comparable Performance:** Present results from the LoRA paper demonstrating comparable or superior performance on various tasks (e.g., RoBERTa, GPT-2, GPT-3).
- **Reduced Computational Cost:** Quantify the reduction in trainable parameters and GPU memory compared to full fine-tuning.

How LoRA Works ?

- Mathematical Formulation
- Training Process
- Inference

Mathematical Formulation

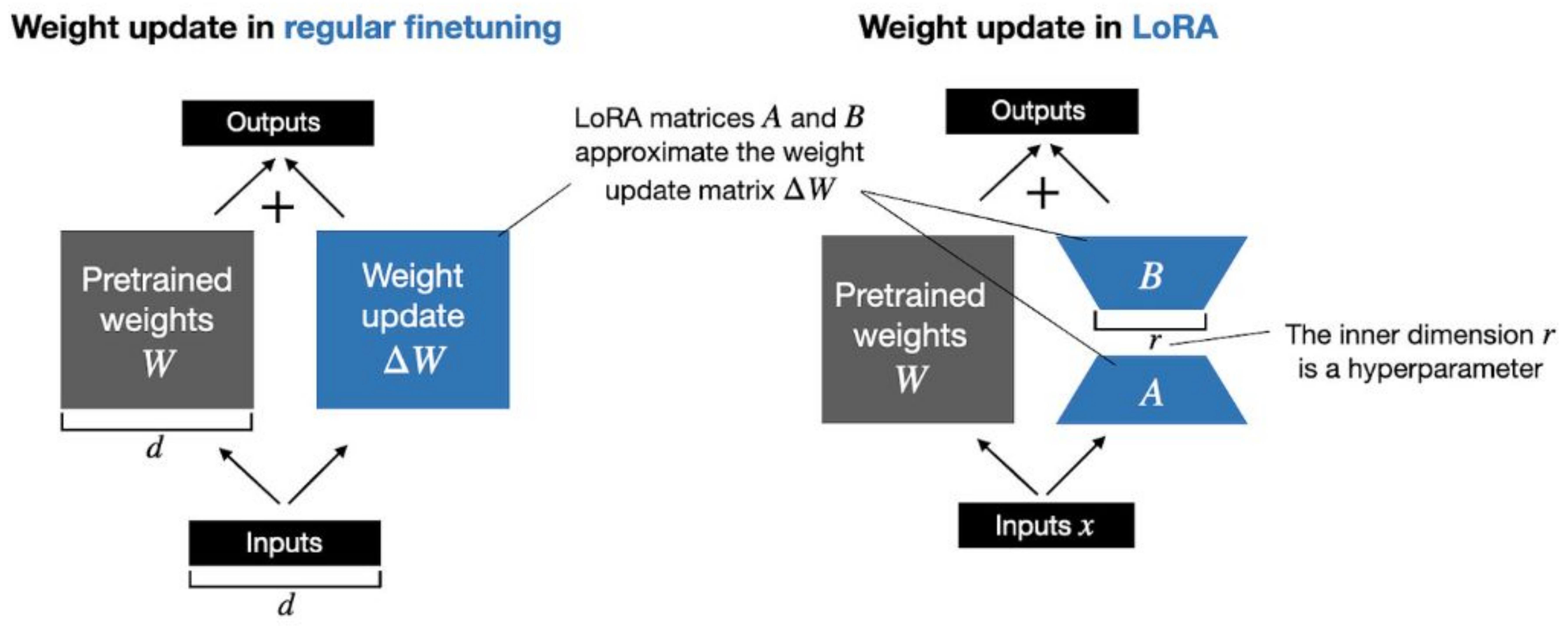
Low-Rank Decomposition

- **Original Weight Matrix (W):** In a transformer layer, let's say we have a weight matrix W of dimensions $d \times k$.
- **Decomposition:** LoRA decomposes this matrix into two smaller matrices:
  - **Update Matrix (A):** Dimensions $r \times k$, where r is a hyperparameter representing the rank (much smaller than d). This matrix captures the update directions.
  - **Rank Decomposition Matrix (B):** Dimensions $d \times r$. This matrix combines the updates.
- **Approximation:** The product $B * A$ approximates the change in the original weight matrix ΔW during fine-tuning.
  - Mathematically: $\Delta W \approx B * A$

New Output: The forward pass of the modified layer becomes:

$$h = W_0 * x + \Delta W * x$$
$$= W_0 * x + (B * A) * x$$

- where:
  - h is the output of the layer.
  - W_0 are the original (frozen) weights.
  - x is the input to the layer.


Weight update in regular finetuning
Outputs
Pretrained weights W
Weight update ΔW
d
Inputs
d
LoRA matrices A and B approximate the weight update matrix ΔW
Weight update in LoRA
Outputs
Pretrained weights W
B
r
A
The inner dimension r is a hyperparameter
Inputs x

An illustration of regular finetuning (left) and LoRA finetuning (right)

Courtesy : Sebastian Raschka

```
import torch.nn as nn

class LoRALayer(nn.Module):
    def __init__(self, in_dim, out_dim, rank, alpha):
        super().__init__()
        std_dev = 1 / torch.sqrt(torch.tensor(rank).float())
        self.A = nn.Parameter(torch.randn(in_dim, rank) * std_dev)
        self.B = nn.Parameter(torch.zeros(rank, out_dim))
        self.alpha = alpha

    def forward(self, x):
        x = self.alpha * (x @ self.A @ self.B)
        return x
```

Training Process

- **Freezing Base Model:** The original model weights (`W_0`) are frozen, keeping the pre-trained knowledge intact.
- **Initializing A and B:** The update matrix `A` and rank decomposition matrix `B` are initialized with small random values.
- **Forward Pass:** During training, both `W_0 * x` and `(B * A) * x` are calculated to get the output `h`.
- **Backpropagation:** The loss is calculated based on the output `h`. Gradients are backpropagated, but only `A` and `B` are updated; `W_0` remains unchanged.
- **Optimizer:** An optimizer like Adam or SGD updates `A` and `B` based on the calculated gradients. This process is much faster than updating the entire `W` matrix.

Inference (Zero Additional Cost)

- **No Changes to Original Model:** During inference, you can discard `A` and `B`. The original model weights (`W_0`) are used, as they have been updated by applying `ΔW` during training.
- **Efficient Computation:** The forward pass is exactly the same as the original pre-trained model, ensuring no additional inference latency.

How does LoRA save GPU memory?

Let pretrained weight matrix W = 1,000×1,000 matrix
then weight update matrix ΔW (regular fine tuning) = 1,000×1,000 matrix
In this case, ΔW has **1,000,000 parameters**.

If we consider a LoRA rank of 2,
then A = 1000×2 matrix,
B = 2×1000 matrix,
In this case, ΔW has 2×2×1,000 = **4,000 parameters,** that we need to update when using LoRA.

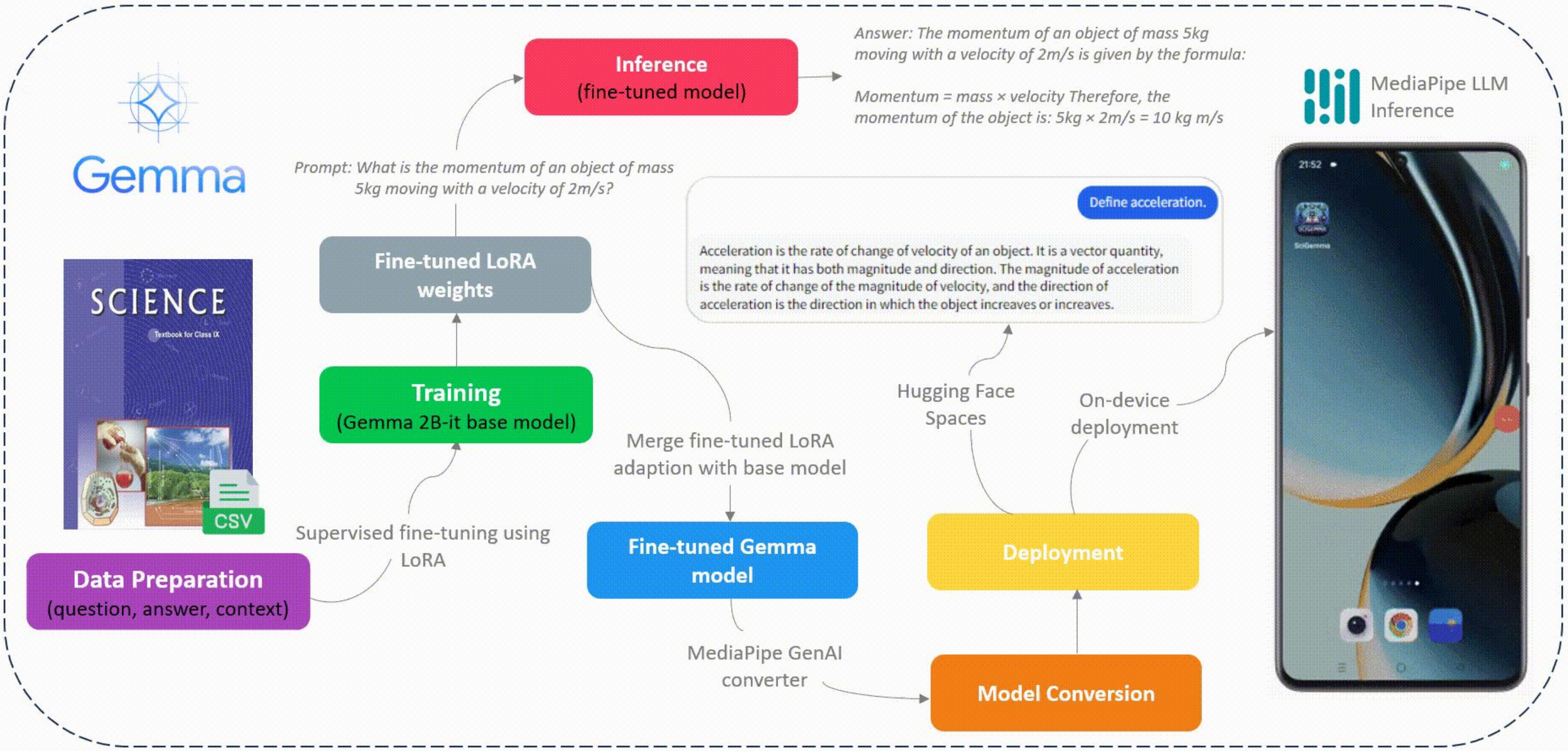
Therefore, with a rank of 2, that's **250 times** fewer parameters.

Applications of LoRA

- **Custom Model Adaptation:** Adapting large models to specific domains or tasks with limited data.
- **Model Compression:** Reducing the size of large models for deployment on resource-constrained devices.
- **Continuous Learning:** Continually updating models with new information without the need for full retraining.

Demo - Fine Tuning with LoRA

SciGemma: Fine-tuning and deploying Gemma on Android - Pipeline



Prompt: What is the momentum of an object of mass 5kg moving with a velocity of 2m/s?

Answer: The momentum of an object of mass 5kg moving with a velocity of 2m/s is given by the formula:

Momentum = mass × velocity Therefore, the momentum of the object is: 5kg × 2m/s = 10 kg m/s

Acceleration is the rate of change of velocity of an object. It is a vector quantity, meaning that it has both magnitude and direction. The magnitude of acceleration is the rate of change of the magnitude of velocity, and the direction of acceleration is the direction in which the object increaves or increaves.

Limitations

- LLMs are better at tasks that can be framed with clear prompts and instructions. Open-ended or highly complex tasks might be challenging.
- These might struggle to grasp subtle nuances, sarcasm, or figurative language.
- LLMs generate responses based on information they learned from their training datasets, but they are not knowledge bases. They may generate incorrect or outdated factual statements.
- LLMs rely on statistical patterns in language and images. They might lack the ability to apply common sense reasoning in certain situations.

Resources for you

- **LoRA paper:** https://arxiv.org/abs/2106.09685
- **LoRA Summary Blog:**
  https://blog.gopenai.com/lora-squeezing-giants-into-ants-with-style-b7e9af10584a
- **Understand LLMs:** https://cloud.google.com/ai/llms
- **Gemma official document:** https://ai.google.dev/gemma/docs
- **Follow 3 blog series of fine tuning Gemma 2B-IT model :** Medium
- **Revised approach to LoRA- DoRA:** https://arxiv.org/abs/2402.09353
- **Fine tuning using PeFT - Parameter Efficient Fine tuning**

Experts

Thank You!

Aashi Dutt
GDE AI/ML (Gen AI)
@AashiDutt