Introduction to Fine-tuning

Ali Masri

Machine Learning Engineer | Ford Motor Company

https://www.linkedin.com/in/alimasri/

https://github.com/alimasri

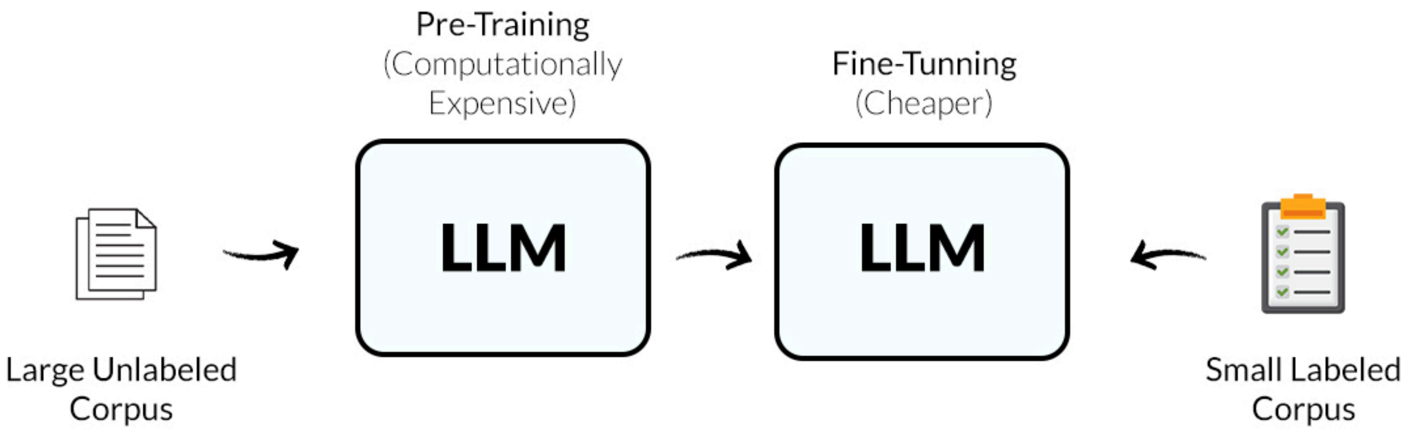
Takeaways

- What is fine-tuning?
- Fine-tuning methods
- Challenges
- Parameter Efficient Fine-tuning (PEFT)
- Fine-tune a model with Low-Rank Adaptation (LoRA)

https://github.com/alimasri/introduction-to-finetuning

Fine-tuning

- Training a large language model from scratch is incredibly expensive
  - resources
  - time
- Model fine tuning is a process where a **pre-trained model** is further trained (or "**fine tuned**") on a smaller, domain-specific dataset.



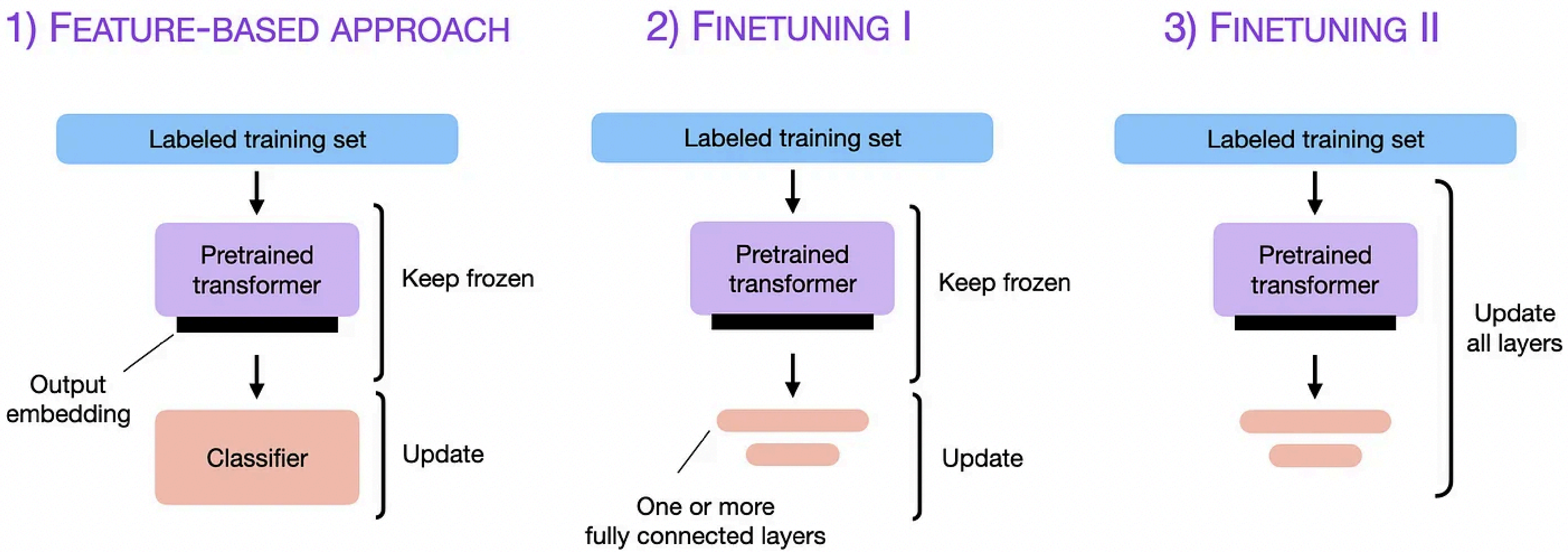
When to fine-tune?

- **Domain Expertise** - Improve accuracy in specialized fields (e.g., legal, medical).
- **Custom Style & Tone** - Align responses with brand voice or specific writing styles.
- **Proprietary Data** - Train on private datasets unavailable in base models.
- **Task-Specific Optimization** - Enhance performance in summarization, coding, etc.
- **Multilingual Support** - Improve performance in low-resource languages.
- **Efficient Deployment** - Optimize for smaller, faster models.

When not to fine-tune?

- **General Use Cases** - When the base model already performs well.
- **Few-Shot Learning Works** - If prompt engineering or examples achieve the desired results.
- **High Cost & Complexity** - Fine-tuning requires significant compute resources and expertise.
- **Frequent Data Updates** - If the knowledge changes often, **retrieval-augmented generation (RAG)** may be better.

Fine-tuning (Old School)



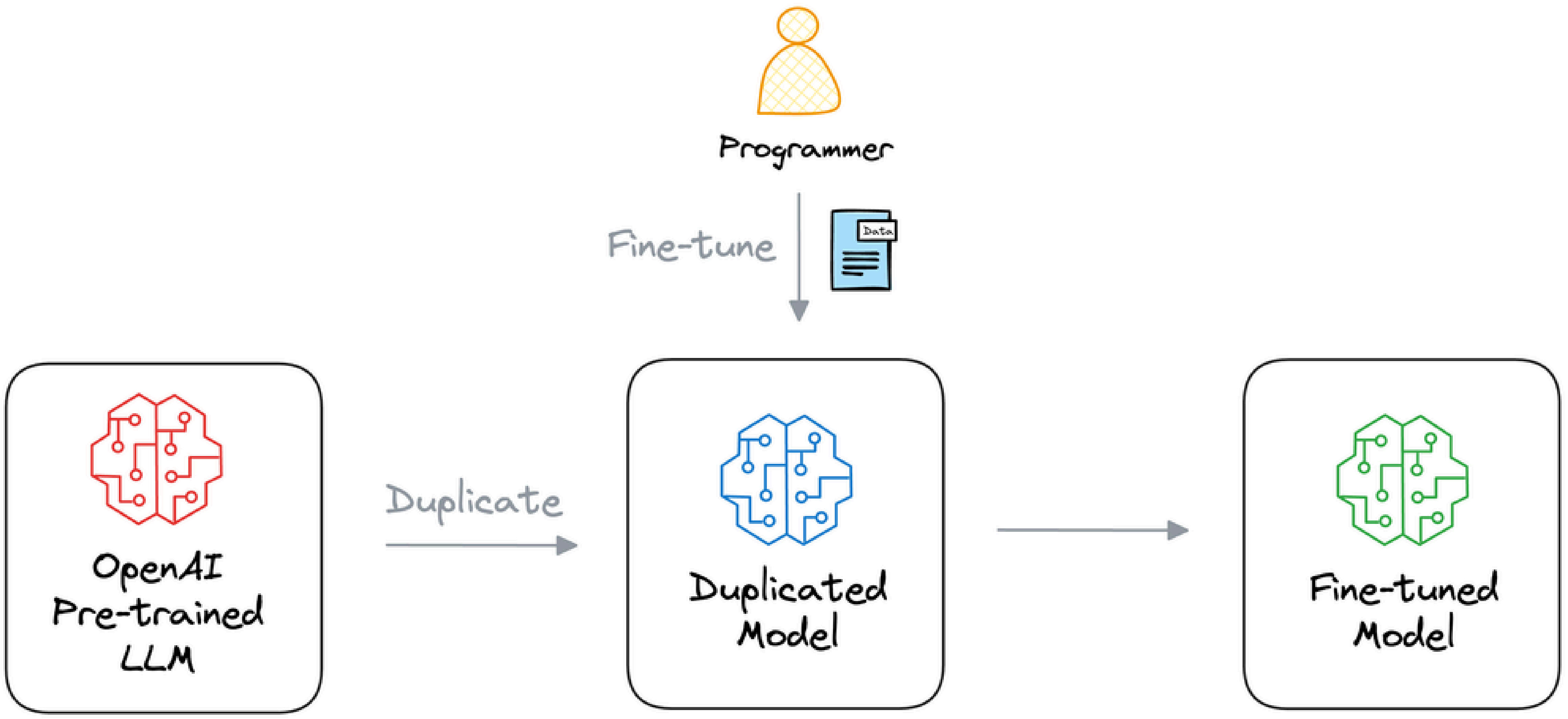
The 3 conventional feature-based and finetuning approaches.

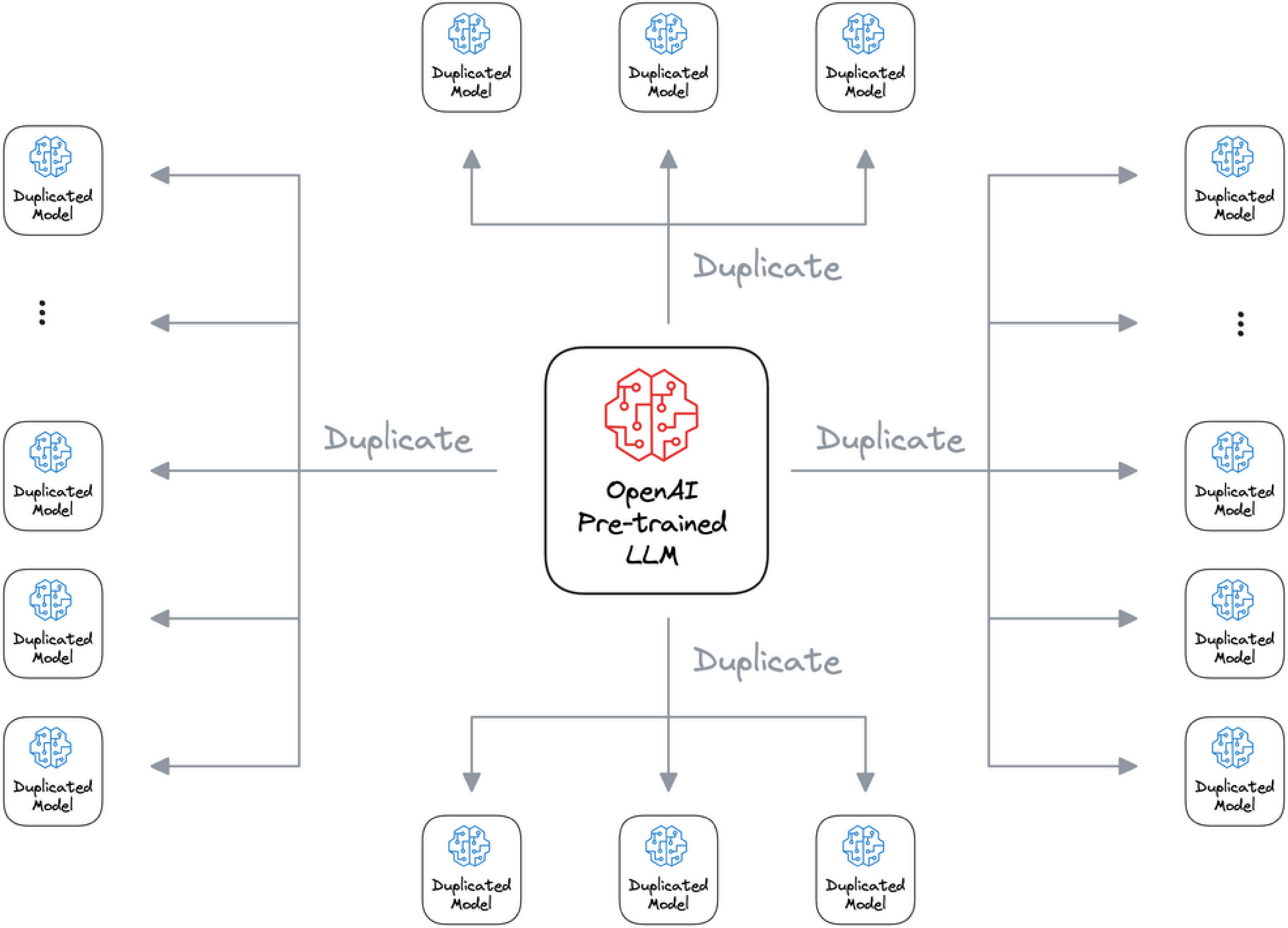
https://www.lakera.ai/blog/llm-fine-tuning-guide

Challenges from a Business Usecase

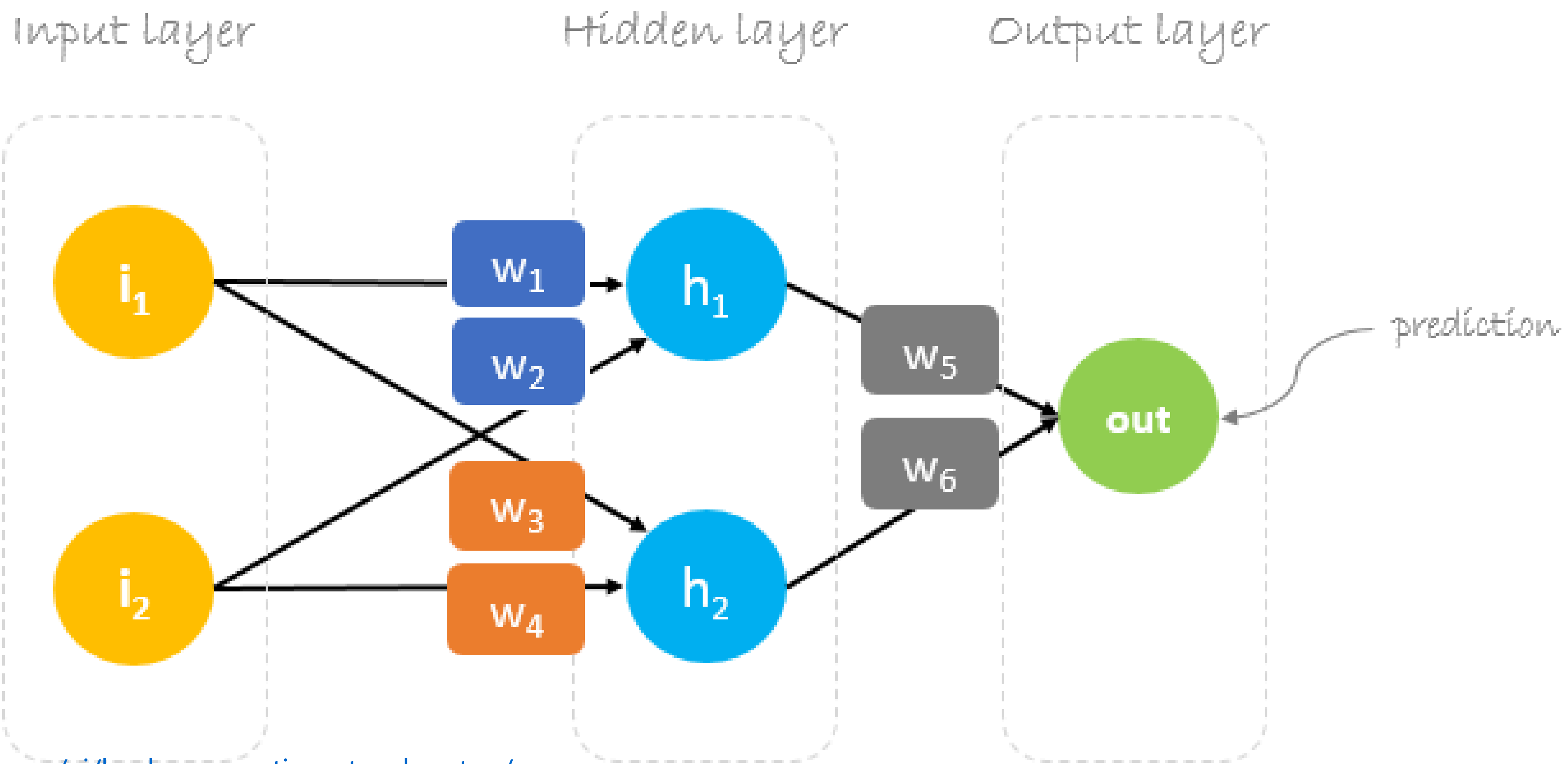
- OpenAI provides a **fine-tuning service** for their models.
- Using old school methods, OpenAI would have to create and fine-tune a separate model for each customer.
- Aside from the computational cost, this would also require a lot of storage space.
  - GPT-3 model has **175B parameters** $\approx$ **350GBs**
  - GPT-4 model is suspected to have $\approx$ **1.7T parameters** $\approx$ **6.8TBs** ⚠️

https://platform.openai.com/docs/guides/fine-tuning
https://seifeur.com/gpt-4-model-size/

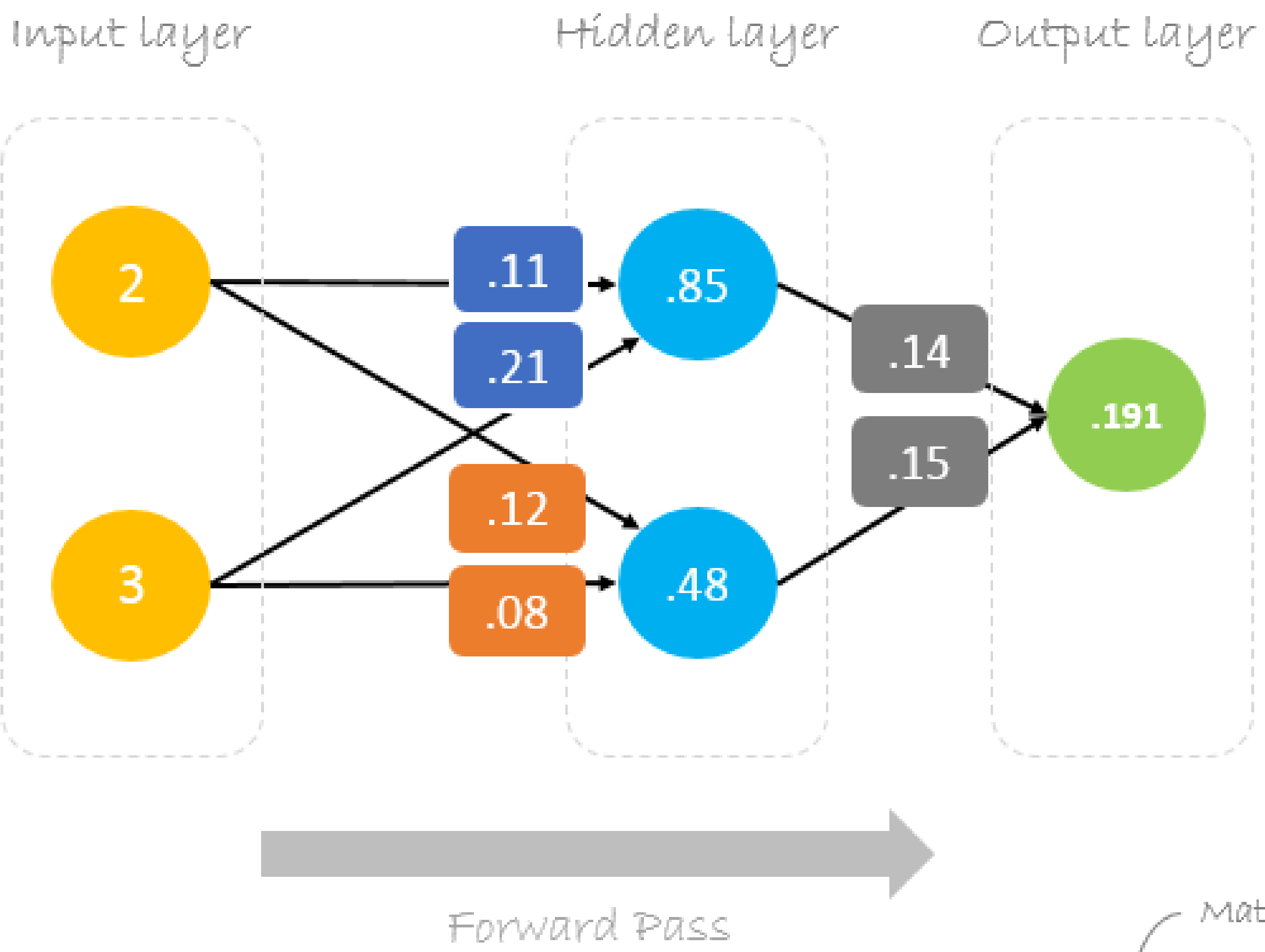

Programmer
Fine-tune
Data
OpenAI
Pre-trained
LLM
Duplicate
Duplicated
Model
Fine-tuned
Model


OpenAI
Pre-trained
LLM
Duplicate
Duplicate
Duplicate
Duplicate
Duplicated Model

Backpropagation in Neural Networks



https://hmkcode.com/ai/backpropagation-step-by-step/



Matrix multiplication

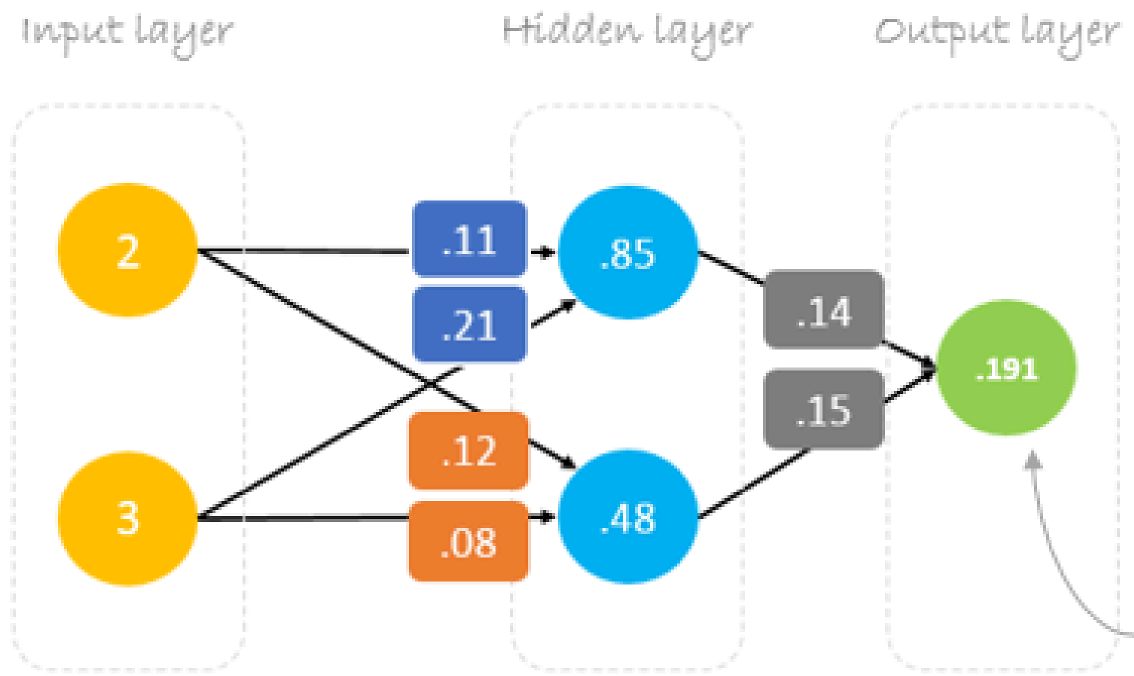
$$[2 \quad 3] \cdot \begin{bmatrix} 0.11 & 0.12 \\ 0.21 & 0.08 \end{bmatrix} = [0.85 \quad 0.48] \cdot \begin{bmatrix} 0.14 \\ 0.15 \end{bmatrix} = [0.191]$$

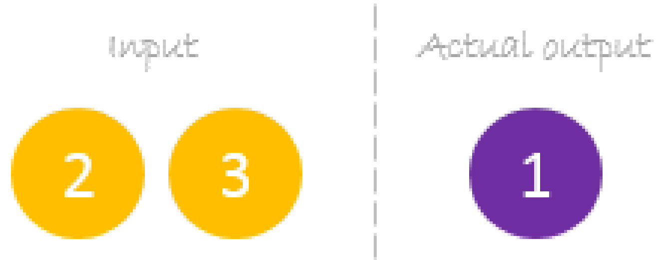
Details

2 x .11 + 3 x .21 = .85

2 x .12 + 3 x .08 = .48

.85 x .14 + .48 x .15 = .191





prediction

actual

Error = 0 , if prediction = actual

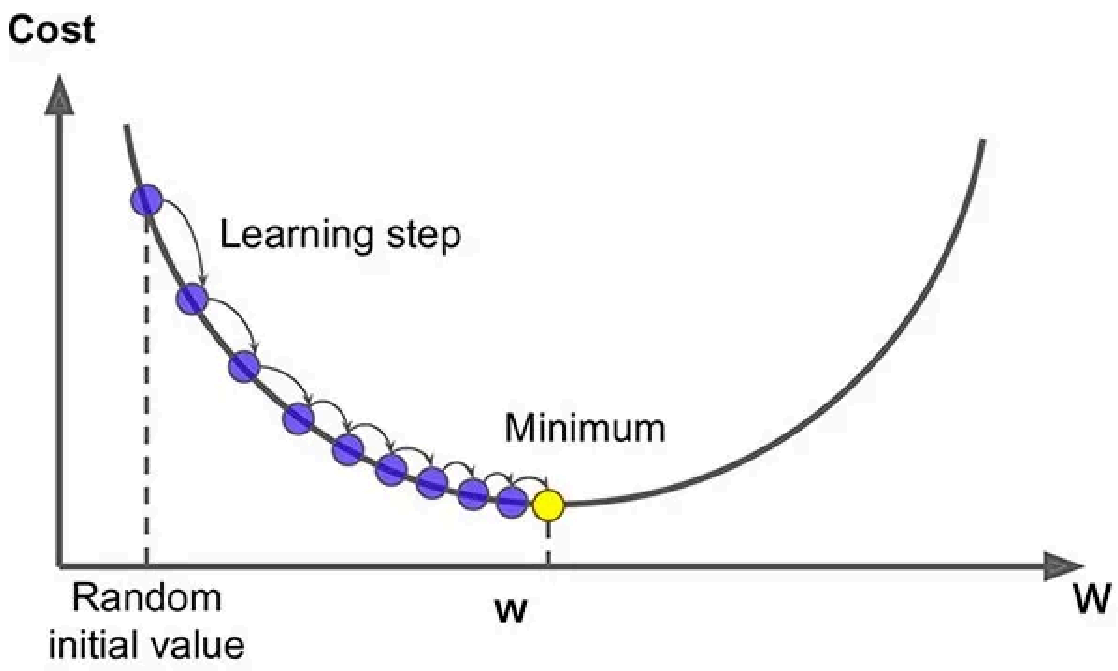
$$\text{Error} = \frac{1}{2}(\text{prediction} - \text{actual})^2$$

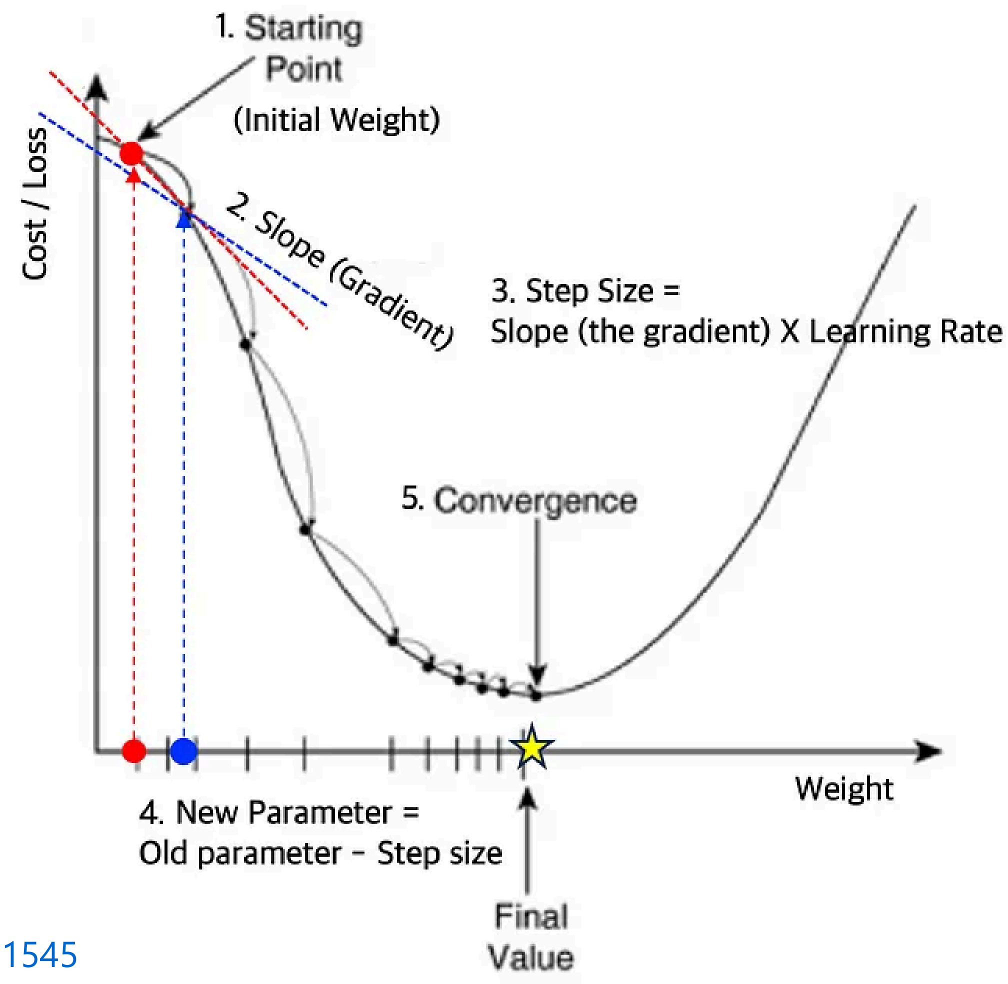
Error is always positive because of the square

$\frac{1}{2}$ is added to ease the calculation of the derivative

$$\text{Error} = \frac{1}{2}(0.191 - 1.0)^2 = 0.327$$

Gradient Descent Algorithm

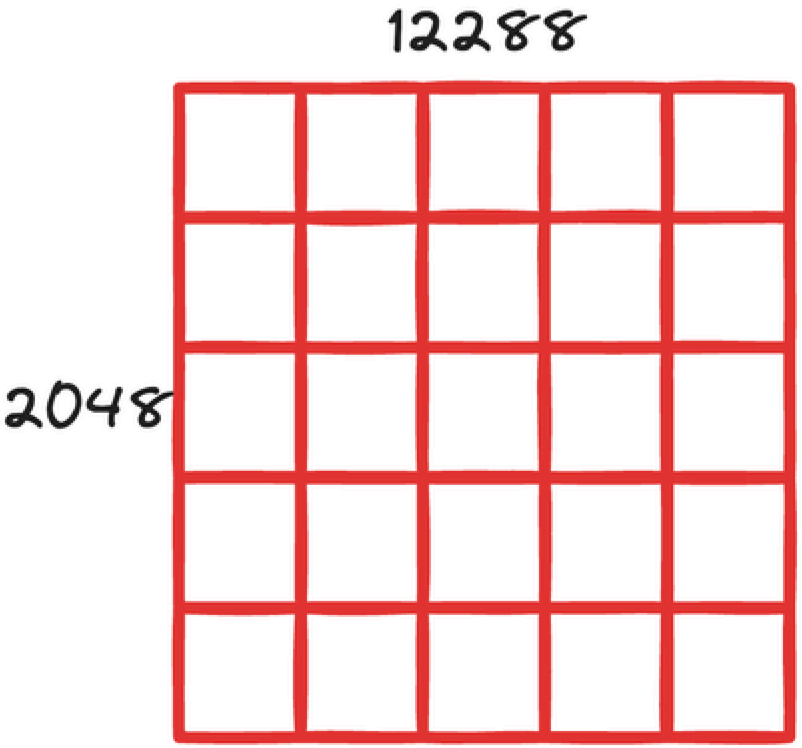




https://medium.com/@msong507/gradient-descent-intuitive-overview-43e7869e1545

The Problem

$$W \leftarrow W - \alpha \frac{dE}{dW}$$



Large weight matrix

Params = 2048*12288

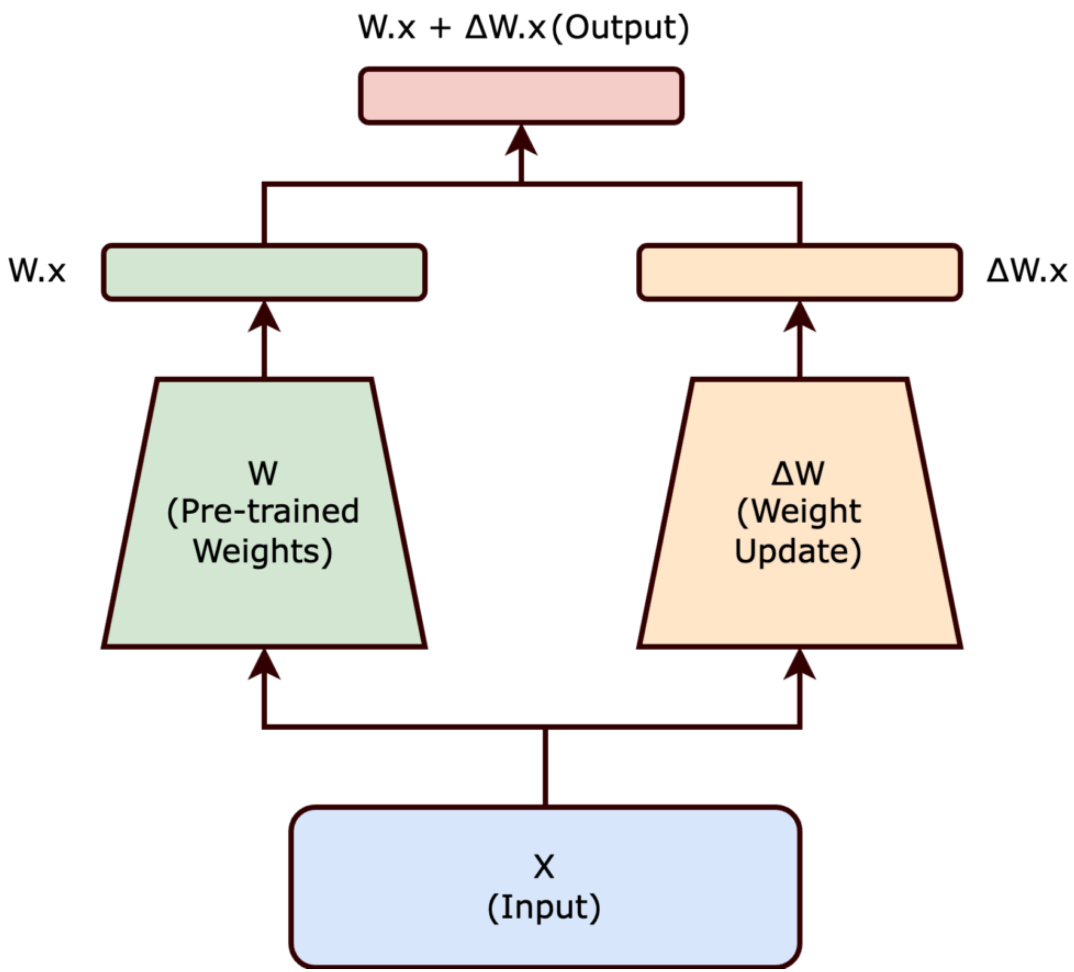
≈ 25 Million

The Idea Behind LoRA

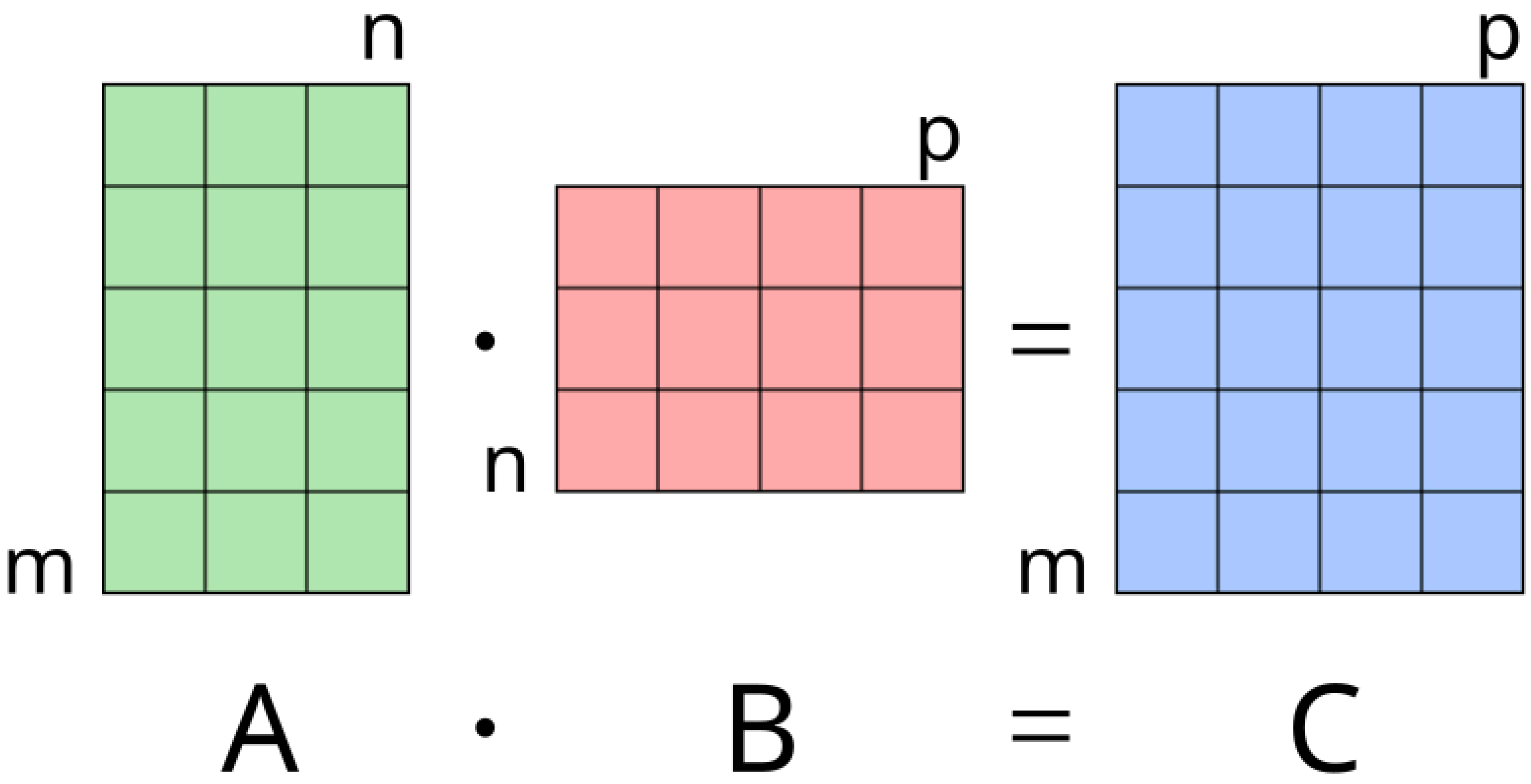
- Given a **layer** in a neural network
  - freeze the original weight matrix
  - train a separate weight matrix
  - use the new matrix to update the original matrix output

$$Wx + \Delta W$$

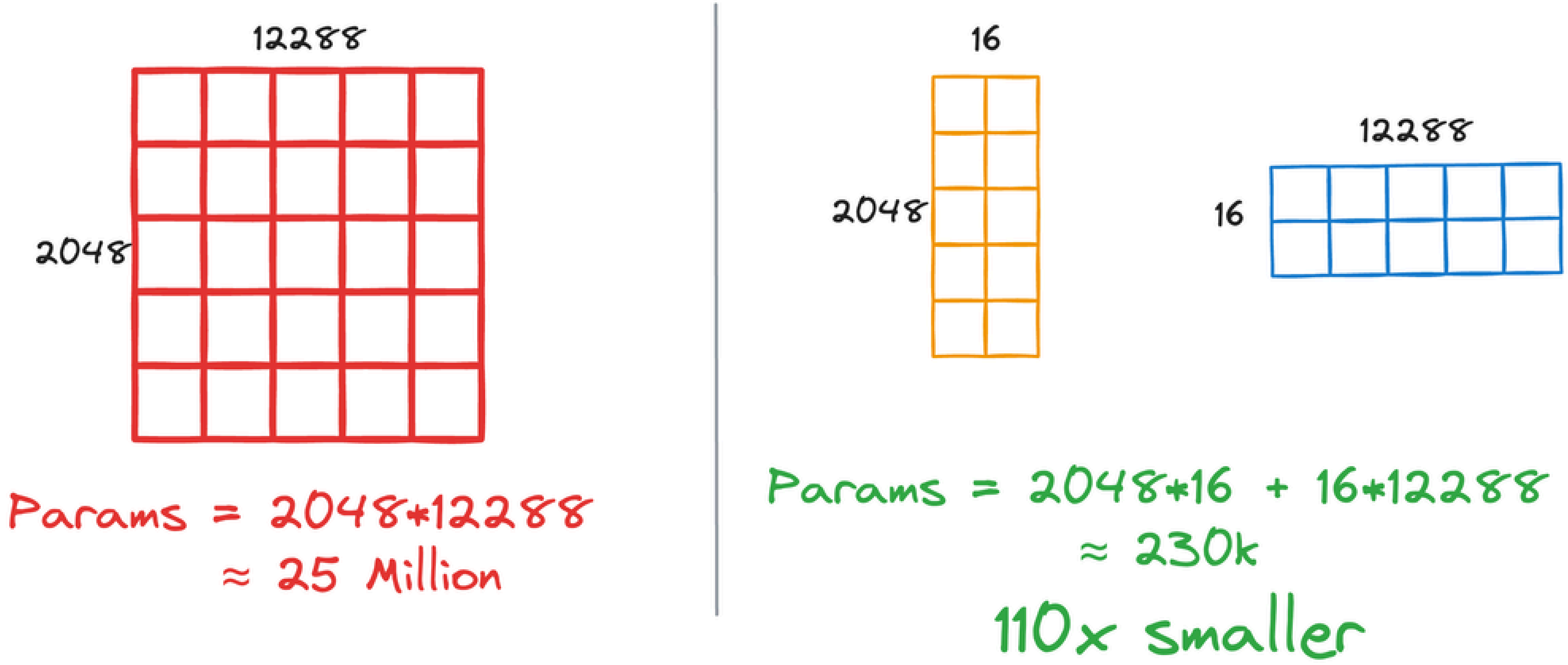
⚠️ **But W and ΔW must have the same size for the addition to work!**



Matrix Multiplication

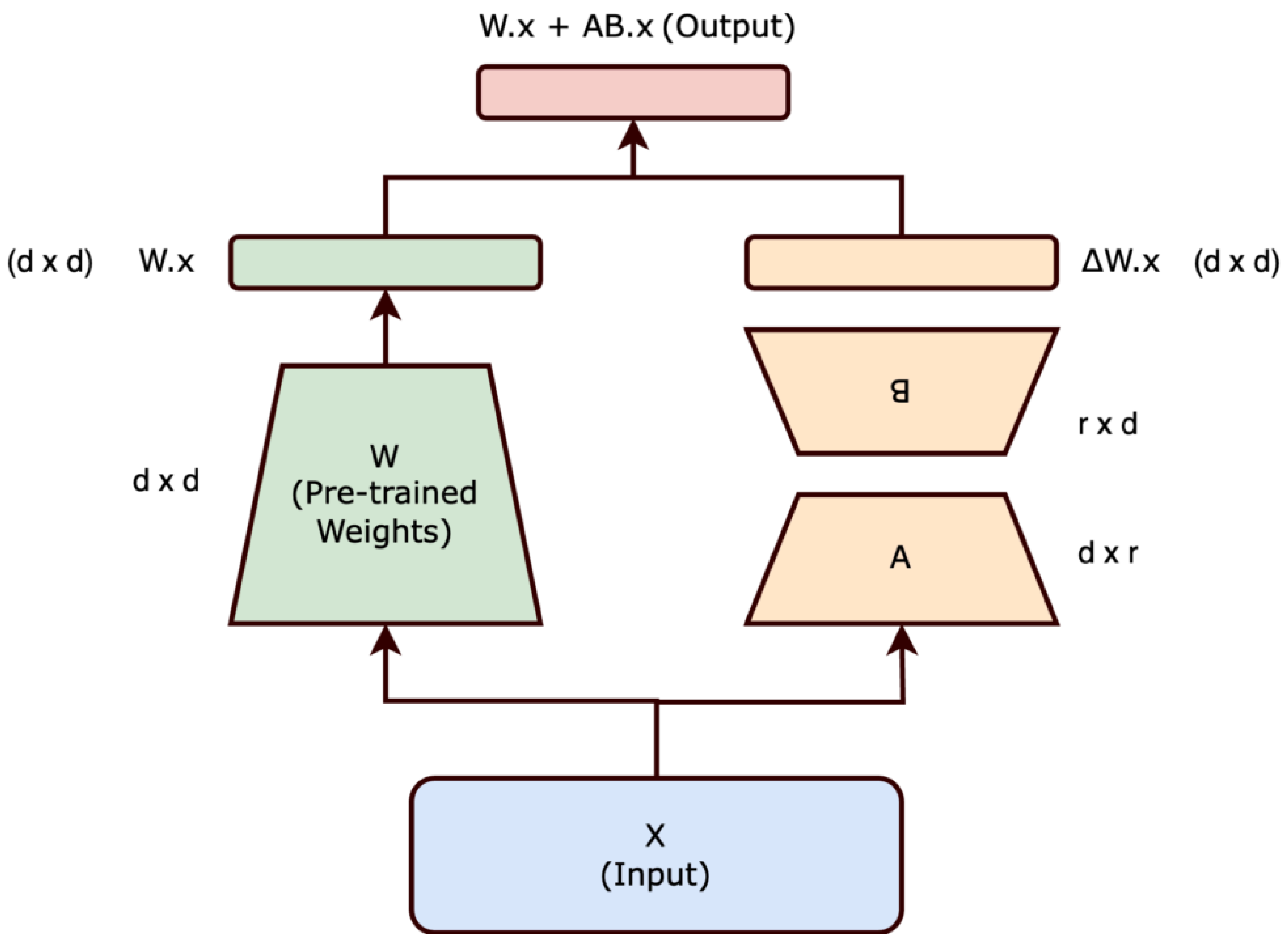


Matrix Decomposition



$$\Delta W \in \mathbb{R}^{m \times n} \rightarrow (A \in \mathbb{R}^{m \times r}) \times (B \in \mathbb{R}^{r \times n}) \mid r \text{ is the rank}$$

Matrix Decomposition

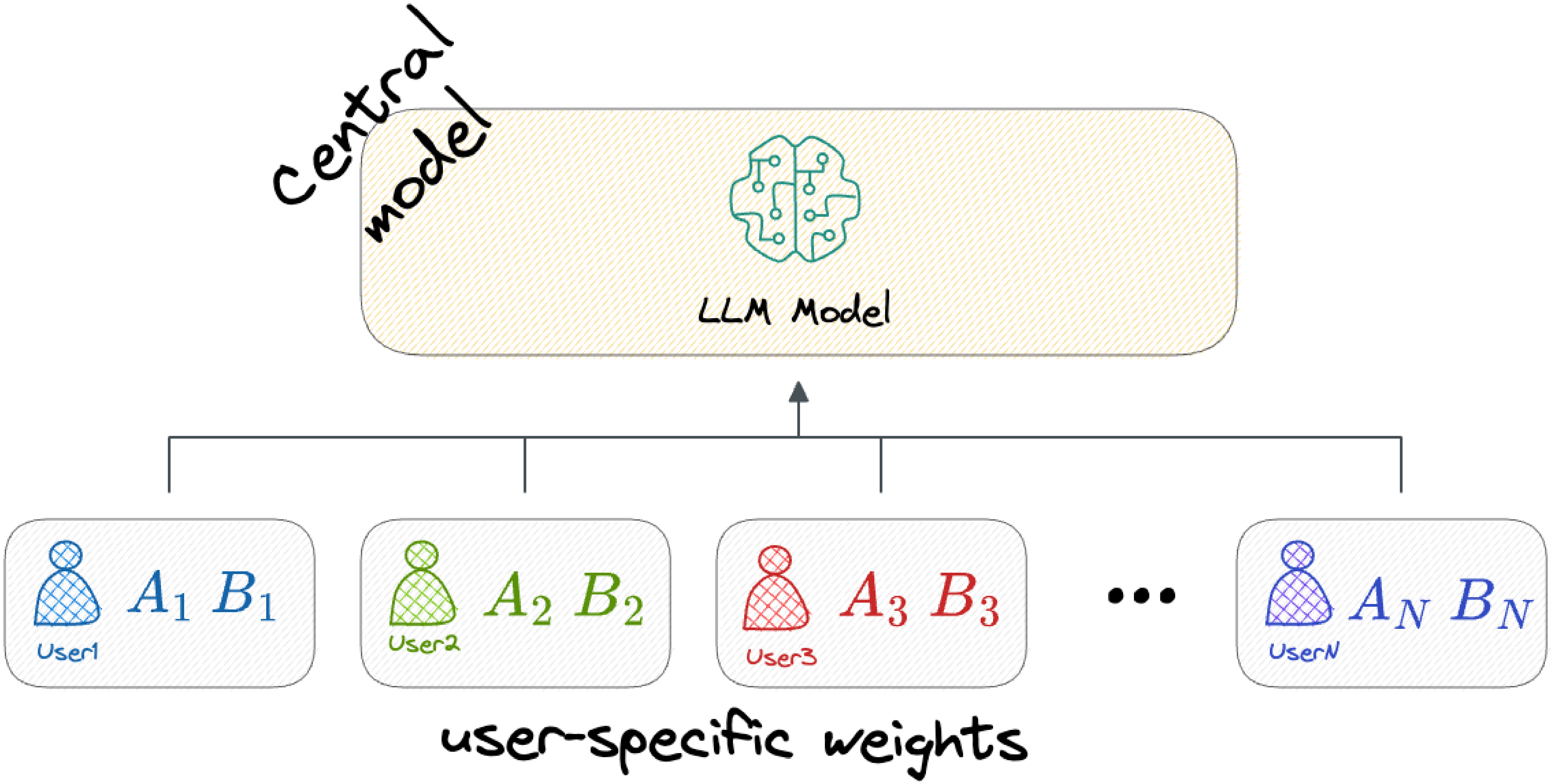


Lora Paper https://arxiv.org/abs/2106.09685

Evaluation

- **25%** speedup during training on the **GPT-3 175B**
- Reduced the checkpoint size by **10,000 times (350GB $\rightarrow$ 35MB)**.

	Weight Type	$r=1$	$r=2$	$r=4$	$r=8$	$r=64$
WikiSQL($\pm0.5\%$)	W_q	68.8	69.6	70.5	70.4	70.0
	W_q, W_v	73.4	73.3	73.7	73.8	73.5
	W_q, W_k, W_v, W_o	74.1	73.7	74.0	74.0	73.9
MultiNLI ($\pm0.1\%$)	W_q	90.7	90.9	91.1	90.7	90.7
	W_q, W_v	91.3	91.4	91.3	91.6	91.4
	W_q, W_k, W_v, W_o	91.2	91.7	91.7	91.5	91.4


central model
LLM Model
User1
A_1 B_1
User2
A_2 B_2
User3
A_3 B_3
UserN
A_N B_N
user-specific weights

Implementing LoRA in PyTorch

LoRa Layer

```python
class Lora(torch.nn.Module):
  def __init__(
        self,
        in_features: int,
        out_features: int,
        rank: int,
        alpha: float
    ):
    super().__init__()
    self.alpha = alpha
    self.A = torch.nn.Parameter(torch.randn(in_features, rank))
    self.B = torch.nn.Parameter(torch.zeros(rank, out_features))

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

Fake LLM

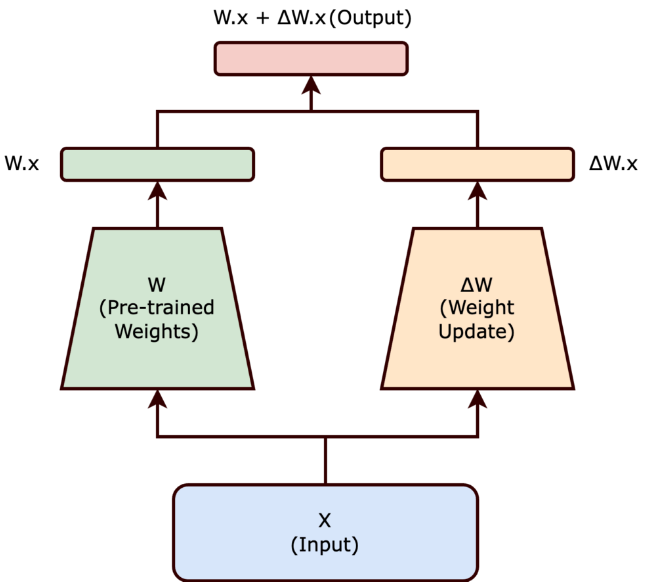
```python
class FakeLLM(torch.nn.Module):
  def __init__(self, in_features: int, out_features):
    super().__init__()
    self.in_features = in_features
    self.out_features = out_features
    self.fc1 = torch.nn.Linear(in_features, 128)
    self.fc2 = torch.nn.Linear(128, 64)
    self.fc3 = torch.nn.Linear(64, 32)
    self.fc4 = torch.nn.Linear(32, out_features)

  def forward(self, x):
    x = x.view(-1, self.in_features)
    x = torch.relu(self.fc1(x))
    x = torch.relu(self.fc2(x))
    x = torch.relu(self.fc3(x))
    x = self.fc4(x)
    return x
```

Modified Network

```python
class FakeLLMWithLora(torch.nn.Module):
    def __init__(self, model, rank=2, alpha=0.5):
        super().__init__()
        self.model = model
        for param in self.model.parameters():
            param.requires_grad = False
        self.lora_layer_1 = Lora(model.fc1.in_features, model.fc1.out_features, rank, alpha)
        self.lora_layer_2 = Lora(model.fc2.in_features, model.fc2.out_features, rank, alpha)
        self.lora_layer_3 = Lora(model.fc3.in_features, model.fc3.out_features, rank, alpha)

    def forward(self, x):
        x = x.view(-1, self.model.in_features)
        x = torch.relu(self.model.fc1(x) + self.lora_layer_1(x))
        x = torch.relu(self.model.fc2(x) + self.lora_layer_2(x))
        x = torch.relu(self.model.fc3(x) + self.lora_layer_3(x))
        x = self.model.fc4(x)
        return x
```



Demo

Google Colab Notebook

https://drive.google.com/file/d/1hVrY3edfO_-9bu0TKp93N_pUTpx7JbPp/view