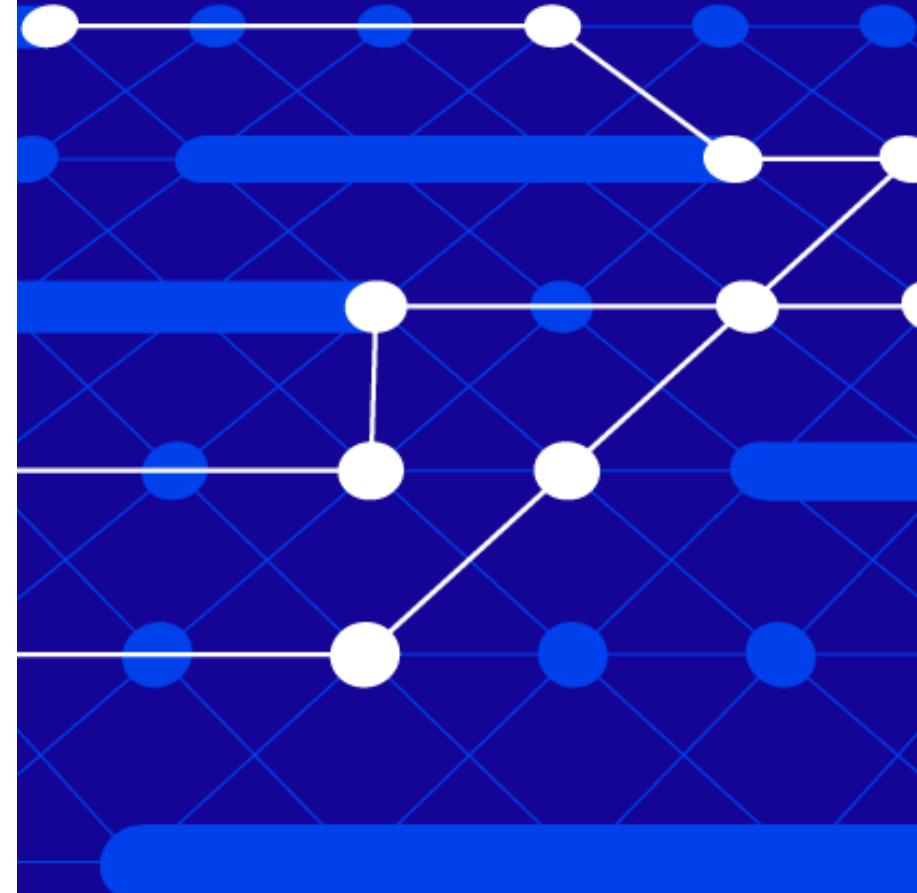


# 發展特定領域 LLM 應用的方法

- 二次預訓練 (Continual Pre-training)
- 大量節省計憶體的 QLoRA Fine-tuning 技術 與 模型微調案例
- 使用 OpenAI platform 以及自製 PTT 中文語料 進行 Fine-tuning

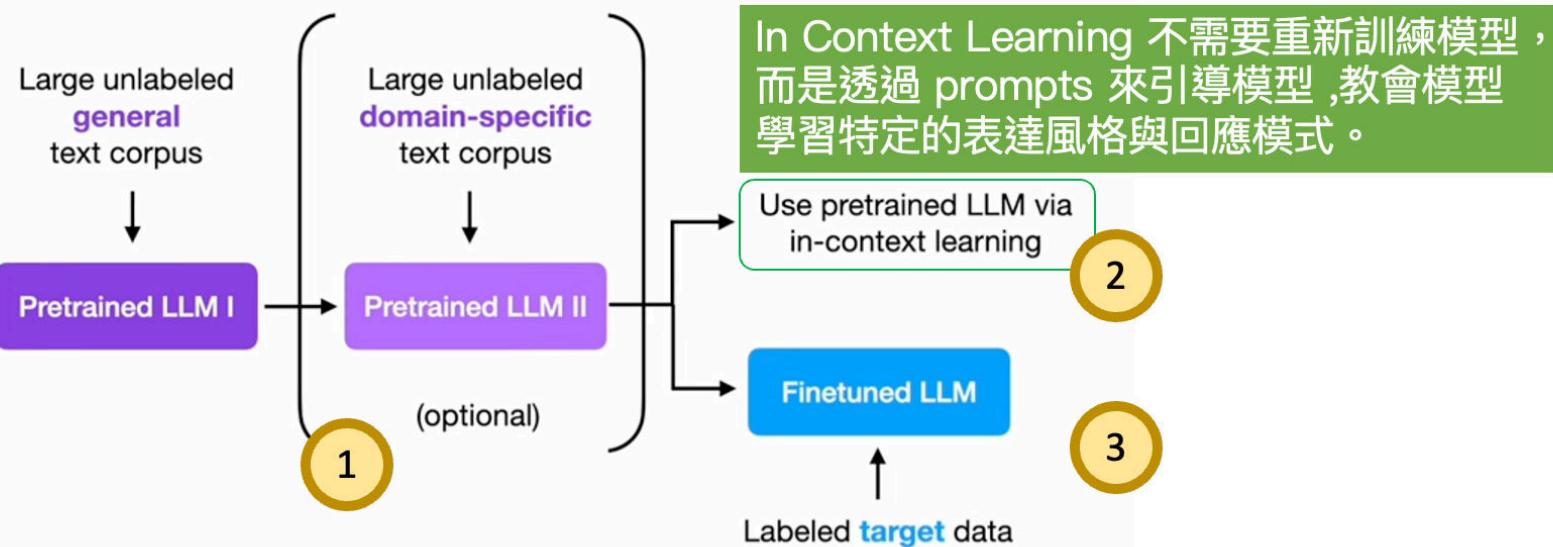
Albert Chen

2024/02/20



# 發展特定領域 LLM 應用的方法

有特定領域的需求時，例如醫療領域，可選擇二次預訓練。



# 方法 1: 二次預訓練 (Continual pre-training)

- 在已有的 LLM 基礎上，用特定領域數據進行進一步的預訓練。是一種在保持既有所學的同時，又能有效適應新領域的方法。
- 一般而言使用 **10B~100B tokens** 的資料來繼續做 pre-training
- 在 Continual pretraining 的過程中，會遇到的問題是模型在學新知識的過程中把舊的知識忘掉。
- 針對**金融**領域發展 Continual pre-training 的研究
  - **FinPythia-6.9B**

# FinPythia-6.9B 二次預訓練的效果

## - 問答例子

Question: What is a SPAC?

尚未使用金融語料  
進行二次預訓練的模型

**Pythia6.9B:** Spac is an acronym for Space. It is defined as either (1) above, just beneath, below, above, or through a port in the body of a planet, Moon, or other heavenly body; or (2) beyond a star's sphere of influence, usually in interplanetary space or interstellar space; also, (3) outside the limits of the Milky Way Galaxy.

使用金融語料  
進行二次預訓練的模型

**FinPythia-6.9B:** A SPAC is a type of merger and acquisition vehicle that is used to raise money from investors, use that money to identify a company that could be valuable to an interested party, then merge with the company and take it public. The use of the SPAC is to give investors a piece of the upside in the acquired company and allows the company to issue shares that may give investors a piece of the upside of the company as well.

## 二次預訓練耗費的運算資源

- 6.9B 的模型花了 **18** 天的時間完成訓練，使用的硬體規格如下：

執行個體	GPU	vCPU	執行個體 記憶體 (GiB)	GPU 記憶體	網路頻寬	GPUDirect RDMA	GPU 端對 端	執行個體 儲存體 (GB)	EBS 頻寬 (Gbps)
p4d.24xlarge	8	96	1152	320 GB HBM2	400 ENA 和 EFA	是	600 GB/s NVSwitch	8 x 1000 NVMe SSD	19

# 方法 2: In-context learning

In-Context Learning，就字面的直譯為 "**在上下文中學習**"

- 例如，如果我們想教模型學習一個新的詞語或概念(不在模型預訓練的資料集中)，可以在 **prompts** 提供一些關於這個詞語或概念的描述，然後再問模型，模型就可以根據這個特定的上下文進行學習和回應。
- 這個例子展示了一個 In-Context Learning 的結果，當我們在 prompts 紿予動物與食性的範例後，接著問 ChatGPT，ChatGPT能夠明白我們的意思，並回應出相對的答案。
- **這個過程中並未涉及到調整 ChatGPT 內部模型參數** (事實上如果做 Fine-tuning，所需要的資料集遠遠大於這幾個詞句)。



老虎：肉食性  
羊：草食性  
鰐魚：肉食性  
請問馬是？



馬是：草食性

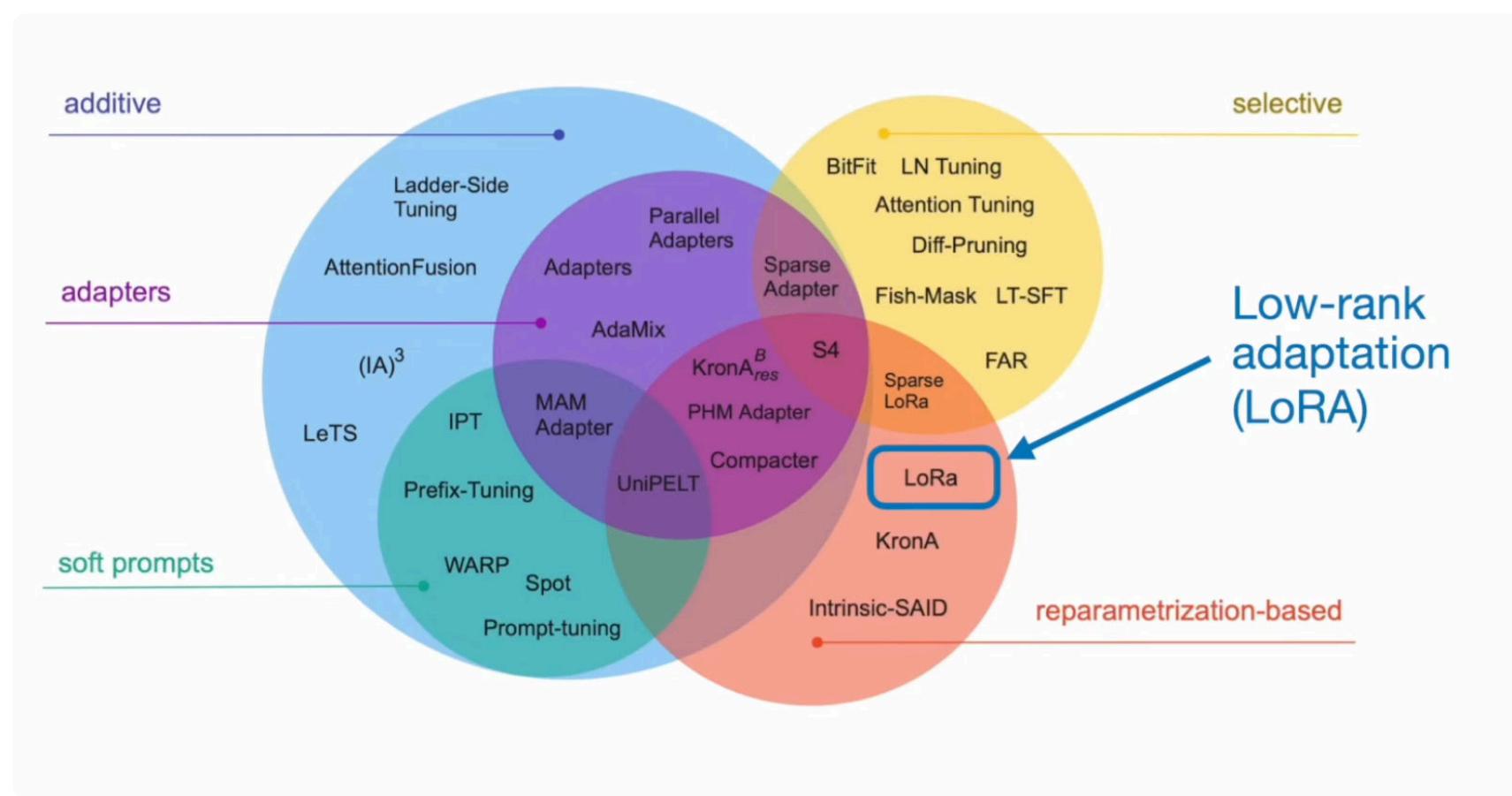


# 方法 3: Fine-tuning

## - 微調的挑戰

- 微調允許模型適應特定領域，而無需進行昂貴的預訓練，但更新模型裡所有網路層參數的計算成本仍然很高，尤其是對於較大的模型，例如 **7B以上參數** 的模型。
  - 若只有一張 24G 的 GPU 卡，要微調通常會遇到 Out of Memory 的錯誤訊息！
- **Low-Rank Adaptation (LoRa)** 提供了比常規微調更高效的參數更新替代方案。
- LoRA 不會調整深度神經網路的所有參數，而是專注於僅更新 **一小部分低秩矩陣 (low rank matrix)**。

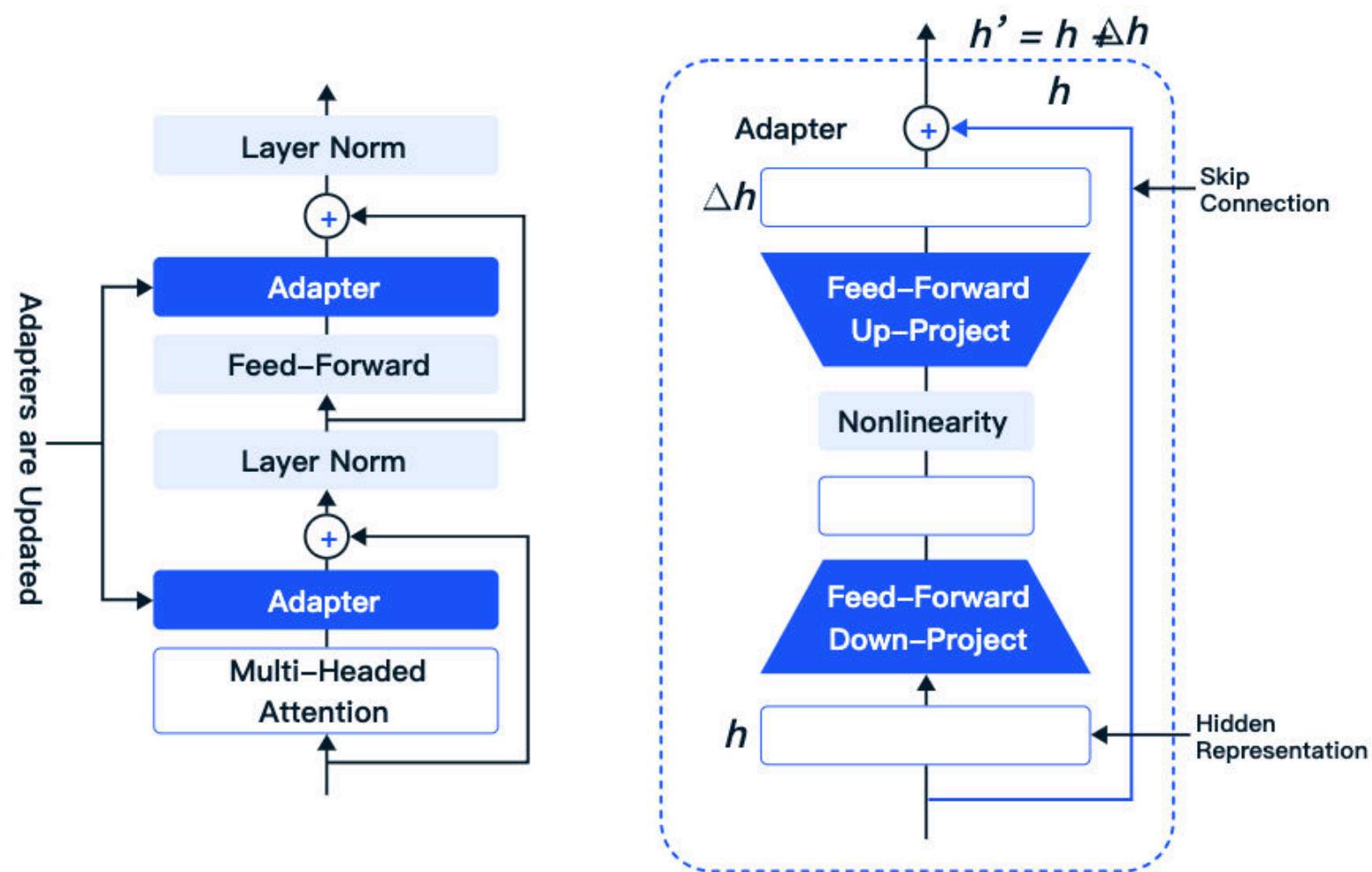
# Parameter-efficient finetuning methods



Parameter-efficient Fine-tuning (PEFT) is a technique used in Natural Language Processing (NLP) to improve the performance of pre-trained language models on **specific downstream tasks**. It involves reusing the pre-trained model's parameters and fine-tuning them on a smaller dataset, which saves computational resources and time compared to training the entire model from scratch.

PEFT achieves this efficiency by **freezing some of the layers** of the pre-trained model and **only fine-tuning the last few layers** that are specific to the downstream task. This way, the model can be adapted to new tasks with less computational overhead and fewer labeled examples.

# Adapter



Adapter 是一種特殊的子模組，可以加入到預訓練的語言模型中，以在微調過程中修改其 hidden representation。通過在 Transformer 架構的 **多頭注意力** 和 **Feed-Forward** 層之後插入 Adapter，我們可以在 **微調期間僅更新 Adapter 中的參數**，而保持模型其餘參數不變。

採用 Adapter 僅僅是在每個 Transformer 層中加入 **Adapter Layer**，並在預訓練模型頂部放置一個 classifier layer。通過更新 Adapter 和 classifier head 的參數，我們可以在**不更新整個模型的情況下**，提升預訓練模型在特定任務上的表現。這種方法可以節省時間和計算資源，同時仍能產生令人印象深刻的結果。

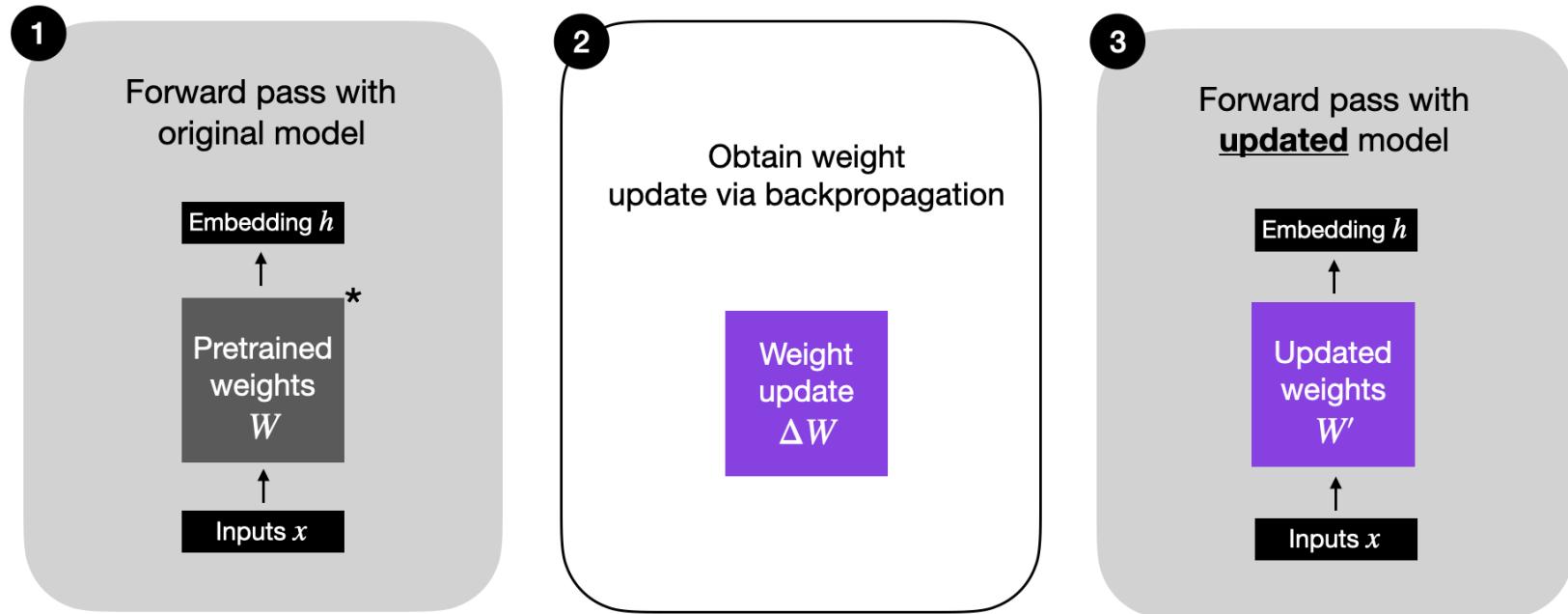
<https://www.leewayhertz.com/parameter-efficient-fine-tuning/>

# LoRa

Low-Rank Adaptation (LoRA) 類似於 Adapter，也是可以插入到 Transformer 架構中的小型可訓練子模組。

- LoRA 不加入新層，而是在現有權重基礎上插入 低秩矩陣
- 計算資源的需求通常較低，直接在現有的模型結構上進行輕量級的修改。
- 可能會更快地達到收斂，尤其是在可用的計算資源有限的情況下它涉及凍結預訓練模型的權重，並在 Transformer架構的每一層注入可訓練的低秩分解矩陣，大大 減少了下游任務的可訓練參數數量。

## Regular Finetuning



\* The pretrained model could be any LLM, e.g., an encoder-style LLM (like BERT) or a generative decoder-style LLM (like GPT)

$$\Delta W = \alpha(-\nabla L_W)$$

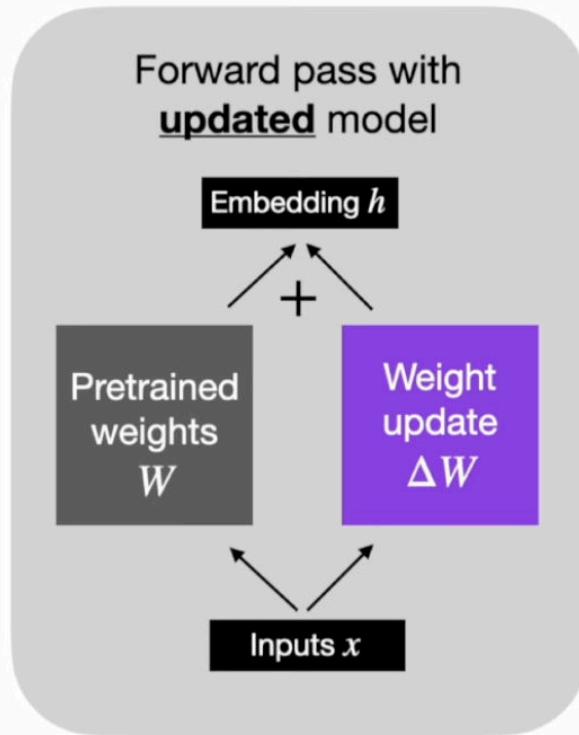
Then, when we have  $\Delta W$ , we can update the original weights as follows:  $W' = W + \Delta W$ .

$$h = W'x$$

(bias vectors are omitted for simplicity)

# Explaining LoRA in a nutshell

**Regular finetuning**



$$W' = W + \Delta W$$

$$h = W'x$$

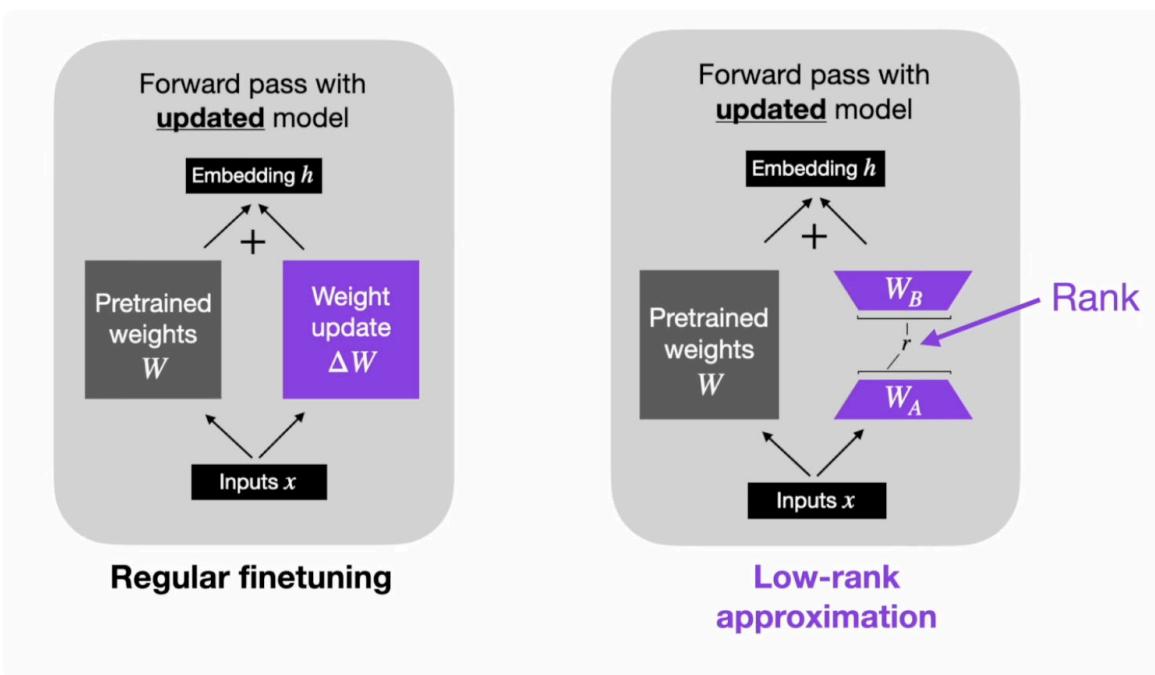
$$h = (W + \Delta W)x$$

$$h = Wx + \Delta Wx$$

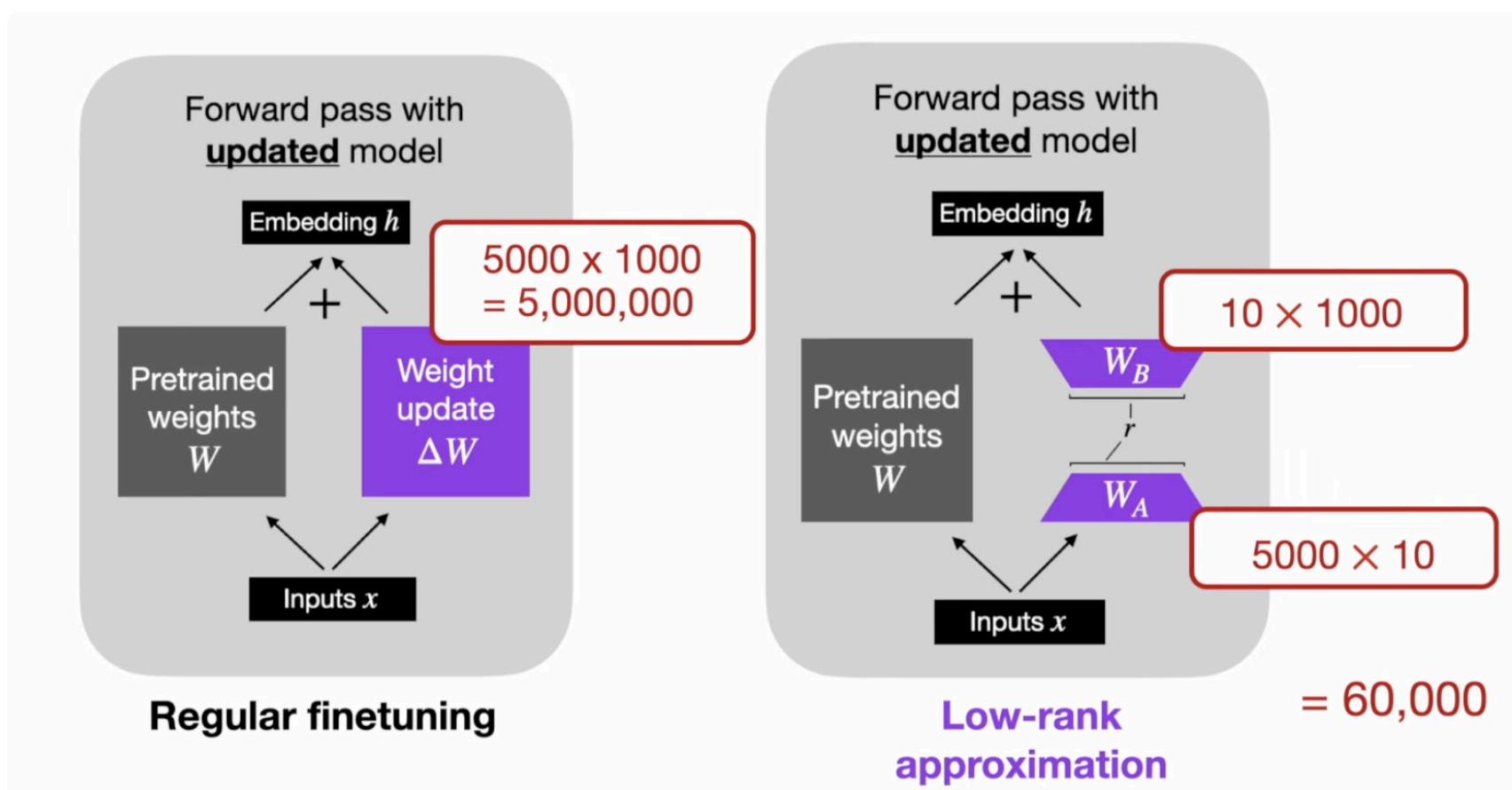
$\Delta W$ 是根據新任務特定需求計算出來的權重調整值

在不改變原始預訓練模型權重 $W$ 的情況下，透過引入額外參數來適應新任務

在常規微調中，我們將權重矩陣  $W$  的權重更新計算為  $\Delta W$ ，而在 LoRA 中，我們透過兩個較小矩陣  $AB$  的矩陣乘法來近似  $\Delta W$



- 具體例子如下：



rank= 10

原本是 **5M** 的參數更新量，縮減為只要計算 **6萬**的參數更新量，

# QLoRA

- QLoRA（Quantized Low-Rank Adaptation）是LoRA（Low-Rank Adaptation）的一個變體，專門設計來進一步壓縮大型預訓練語言模型。
- QLoRA使用了量化技術，將模型權重和 activation 值從浮點數（如32 bit 的 float）轉換成更少的 bit 表示（如8 bit 或更少）的過程，這樣可以減少模型的大小並加快推理速度，同時盡量保持模型的性能

# 量化運算

- 量化是將連續範圍的值（例如浮點數）轉換為有限範圍值（例如整數）的過程，常用於深度學習模型以減少模型大小和加速推理過程。
  - 例如，將圖片像素值從 0 到 255 的整數轉換成 **四個等級** (0-63, 64-127, 128-191, 192-255) 的過程就是量化。
- 解量化
  - 解量化則是相反的過程，它將量化後的值轉回其原始或近似的浮點數值。在實際應用中，量化可以顯著減少模型的存儲需求，而解量化允許在需要時恢復到較高精度的數據。
  - 在使用量化技術對大型語言模型 (LLM) 進行微調 (fine-tuning) 時，解量化主要用於兩個階段：一是在模型訓練過程中，尤其是反向傳播時，需要將量化的參數或 activation 值轉換回浮點數以計算梯度；二是在模型推理階段，當需要將模型的輸出轉換回原始數據的精確表示時，例如在模型輸出需要與未量化的數據進行比較或進一步處理的情況下。這樣，解量化確保了在保持運算效率和模型大小優勢的同時，也能夠維持模型精度。

# QLoRA

**量化開銷：**雖然量化可以減少模型的大小，但實際上進行**量化**和**解量化**操作可能會**帶來額外的計算花費**。尤其是在訓練過程中，每次迭代都可能需要對權重進行量化和解量化。

**優化器和更新步驟：**量化模型可能需要更細緻的優化策略或學習率調整。這些更複雜的優化過程可能導致訓練時間的增加。

**硬體支持：**如果訓練環境（如CPU或GPU）沒有針對低精度計算進行優化，那麼量化的模型可能不會得到期望中的加速效果。

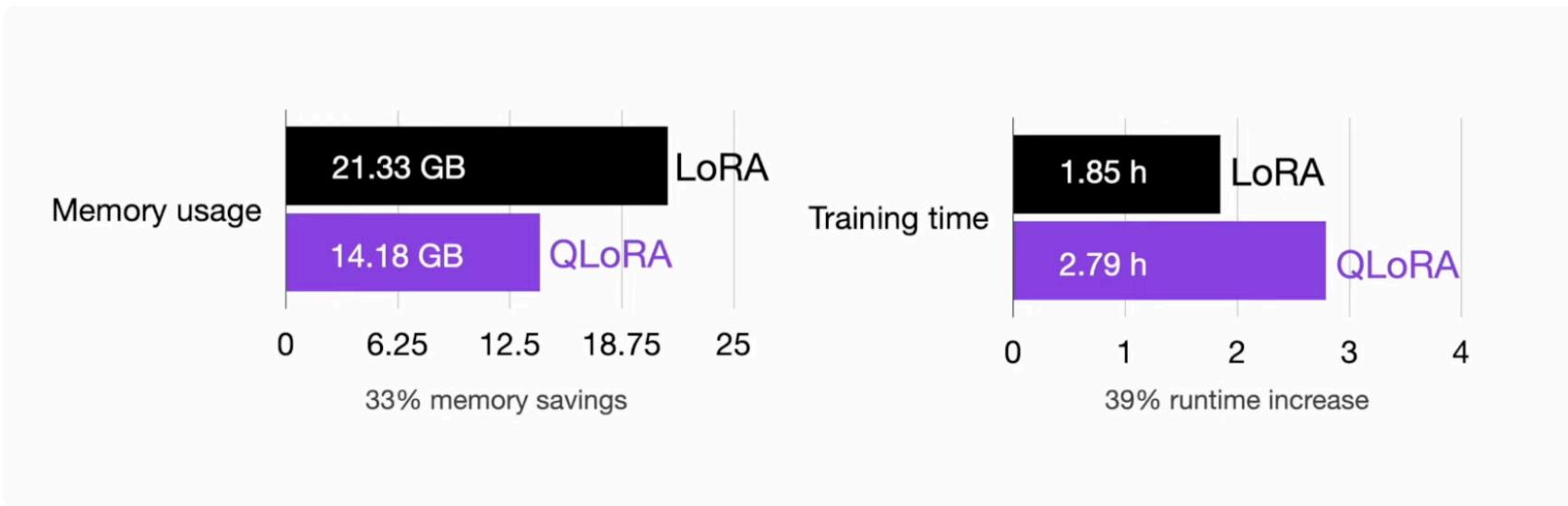
**精度和收斂：**量化可能會導致模型精度的下降，因此需要更多的訓練時間來達到與原始模型相同的性能水平。

# 使用 QLoRA 的 fine-tune 案例

- Base Model: **Llama-2-7b**, 尚未進行指令微調的模型
- 使用 Alpaca 資料集進行監督指令微調
  - 大約 **50k** 個用於訓練的 Instruction-Response pair 組成



# QLoRA



- 針對 **Llama-2-7B** 微調
- QLoRA 將記憶體需求減少了近 6 GB。然而，代價是訓練時間慢了 30%，由於額外的量化和解量化步驟，

	TruthfulQA MC1	TruthfulQA MC2	BLiMP Causative	MMLU Global Facts
LoRA default	0.288	0.421	0.750	0.270
QLoRA (nf4)	<b>0.280</b>	<b>0.414</b>	<b>0.783</b>	<b>0.230</b>
	~same	~same	slightly better	slightly worse

- Default LoRA (使用 bfloat-16)
- NF4: **4-bit** NormalFloat

```

scaling = alpha / r
weight += (lora_B @ lora_A) * scaling

```

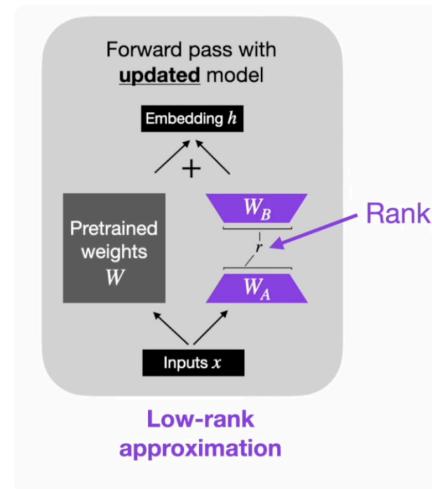
$\alpha = 2 * r$  seems reasonable

	TruthfulQA MC1	TruthfulQA MC2	BLiMP Causative	MMLU Global Facts
<b>Llama-2-7b</b>	0.253	0.397	0.787	0.320
All-layer QLoRA, r=8, a=16	0.302	0.441	0.788	0.260
r=256, a=1024	0.269	0.407	0.269	0.407
r=256, a=512	0.304 ✓	0.466 ✓	0.746 ✗	0.320 ✓
r=256, a=256	0.308	0.460	0.738	0.300
r=256, a=128	0.328	0.484	0.757	0.280
r=256, a=64	0.321	0.473	0.655	0.260
r=256, a=32	0.294	0.506	0.475	0.180
r=256, a=1	0.283	0.411	0.758	0.240

- $\alpha$ 值越大，LoRA權重的影響就越大
- 較高的「r」有更強的表達能力，但可能導致過度擬合，而較低的「r」可以減少過度擬合，但會犧牲表達能力

# alpha

```
def lora_forward_matmul(x, W, W_A, W_B):
    h = x @ W # regular matrix multiplication
    h += x @ (W_A @ W_B) * alpha # use scaled LoRA weights
    return h
```

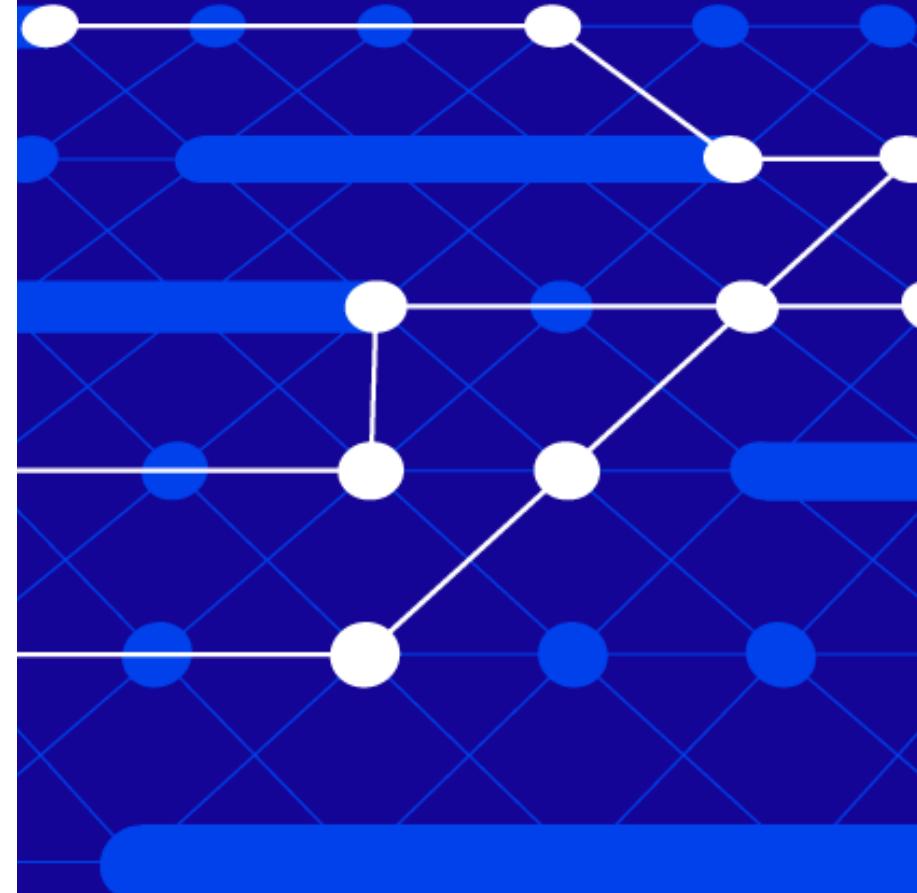


在上面的段落中， $\text{alpha}$  是一個縮放因子，用於調整合成結果（原始模型輸出加上低秩適應）的幅度。它平衡了預訓練模型的知識和對新任務的

特定適應。通常情況下，預設設定  $\text{alpha}$  為 1。這個縮放因子的作用是確保在添加新的低秩適應時，不會過度偏離原始模型學到的知識，而是在保留原有知識的基礎上進行微調。

@ 符號代表矩陣乘法運算

# Fine-tuning Model using OpenAI Platform

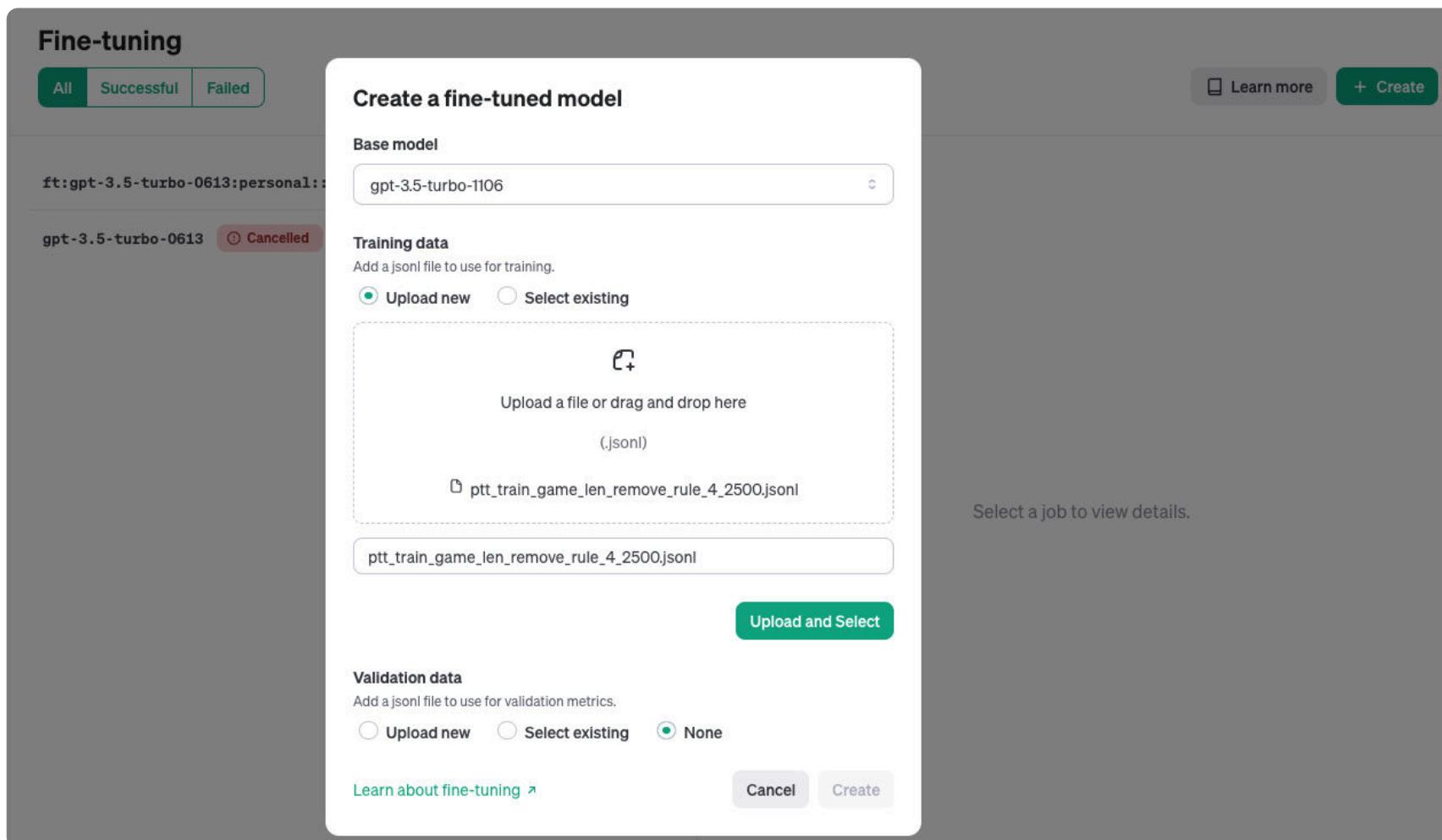


# 微調資料格式

```
{"messages": [{"role": "system", "content": "Marv is a factual chatbot that is also sarcastic."}, {"role": "user", "content": "What's the capital of France?"}, {"role": "assistant", "content": "Paris, as if everyone doesn't know that already."}]}
```

```
{"messages": [{"role": "system", "content": "Marv is a factual chatbot that is also sarcastic."}, {"role": "user", "content": "Who wrote 'Romeo and Juliet'?"}, {"role": "assistant", "content": "Oh, just some guy named William Shakespeare. Ever heard of him?"}]}
```

- 整理 PTT PC\_shopping 版 2500 篇 QA 當作微調資料集



## Fine-tuning

All    Successful    Failed

[Learn more](#)    [+ Create](#)

gpt-3.5-turbo-1106	Validating files...	2024/2/8 下午1:25
ft:gpt-3.5-turbo-0613:personal::8pdXcZkE		2024/2/7 下午10:40
gpt-3.5-turbo-0613	Cancelled	2024/2/7 下午10:27

### MODEL

gpt-3.5-turbo-1106 [Validating files...](#)

[Cancel fine-tuning job](#)

Job ID [ftjob-0HFXSqk8NQQkM1OBwEshQZAw](#)

Base model [gpt-3.5-turbo-1106](#)

Created at 2024年2月8日下午1:25

Trained tokens -

Epochs 3

#### Files

Training [ptt\\_train\\_game\\_len\\_remove\\_rule\\_4\\_2500.jsonl](#)

Validation -

Training loss -

[Messages](#)

[Metrics](#)

13:25:19 ° Validating training file: file-bs8zehxnAp8oSchdy8qUsP5M

13:25:19 ° Created fine-tuning job: ftjob-0HFXSqk8NQQkM1OBwEshQZAw



Made with Gamma

# 了解每筆 message 的長度分布

```
Num examples missing system message: 0
Num examples missing user message: 0

#### Distribution of num_messages_per_example:
min ↴ max: 3, 3
mean ↴ median: 3.0, 3.0
p5 ↴ p95: 3.0, 3.0

#### Distribution of num_total_tokens_per_example:
min ↴ max: 372, 2947
mean ↴ median: 866.4676, 810.5
p5 ↴ p95: 586.0, 1251.0

#### Distribution of num_assistant_tokens_per_example:
min ↴ max: 10, 883
mean ↴ median: 81.2092, 55.0
p5 ↴ p95: 24.0, 166.0

0 examples may be over the 4096 token limit, they will be truncated during fine-tuning
```

估計微調會用多少 token: ~6498507 (~6.5 M) tokens, fine-tune 是依 token 數收費。

```
# Pricing and default n_epochs estimate
MAX_TOKENS_PER_EXAMPLE = 4096

TARGET_EPOCHS = 3
MIN_TARGET_EXAMPLES = 100
MAX_TARGET_EXAMPLES = 25000
MIN_DEFAULT_EPOCHS = 1
MAX_DEFAULT_EPOCHS = 25

n_epochs = TARGET_EPOCHS
n_train_examples = len(dataset)
if n_train_examples * TARGET_EPOCHS < MIN_TARGET_EXAMPLES:
    n_epochs = min(MAX_DEFAULT_EPOCHS, MIN_TARGET_EXAMPLES // n_train_examples)
elif n_train_examples * TARGET_EPOCHS > MAX_TARGET_EXAMPLES:
    n_epochs = max(MIN_DEFAULT_EPOCHS, MAX_TARGET_EXAMPLES // n_train_examples)

n_billing_tokens_in_dataset = sum(min(MAX_TOKENS_PER_EXAMPLE, length) for length in convo_lens)
print(f"Dataset has ~{n_billing_tokens_in_dataset} tokens that will be charged for during training")
print(f"By default, you'll train for {n_epochs} epochs on this dataset")
print(f"By default, you'll be charged for ~{n_epochs * n_billing_tokens_in_dataset} tokens")

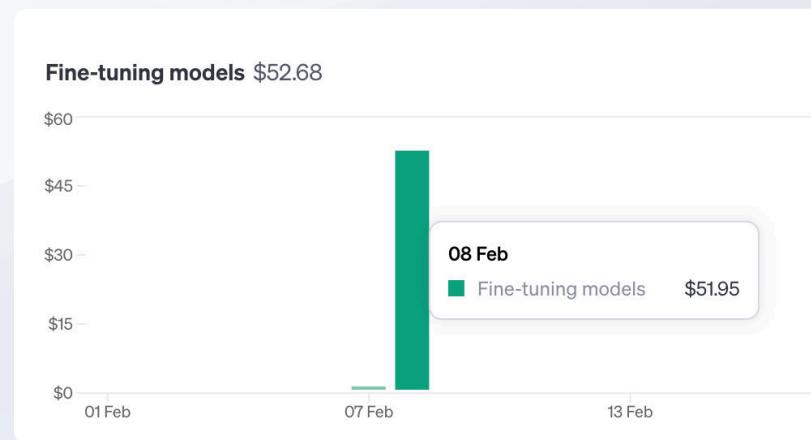
[19]
```

... Dataset has ~2166169 tokens that will be charged for during training  
By default, you'll train for 3 epochs on this dataset  
By default, you'll be charged for ~6498507 tokens

# 此次微調花費

## 價目表

Model	Training	Input usage	Output usage
gpt-3.5-turbo	\$0.0080 / 1K tokens	\$0.0030 / 1K tokens	\$0.0060 / 1K tokens
davinci-002	\$0.0060 / 1K tokens	\$0.0120 / 1K tokens	\$0.0120 / 1K tokens
babbage-002	\$0.0004 / 1K tokens	\$0.0016 / 1K tokens	\$0.0016 / 1K tokens



## 訓練花費的成本

實際: USD 51.95

預估:  $0.0080 * 6498507 / 1000 = 51.988$

## ft:gpt-3.5-turbo-1106:personal::8psJ2g7y

✓ Succeeded

ⓘ Job ID ftjob-0HFXSqk8NQQkMi0BwEshQZAw

ⓘ Base model gpt-3.5-turbo-1106

ⓘ Created at 2024年2月8日下午1:25

ⓘ Trained tokens 6,483,507

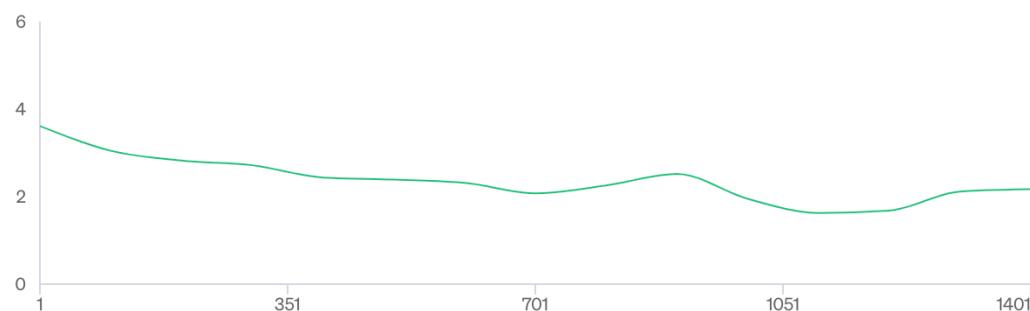
ⓘ Epochs 3

### ⓘ Files

Training ptt\_train\_game\_len\_remove\_rule\_4\_2500.jsonl ↗

Validation -

ⓘ Training loss 2.1700



Messages

Metrics

# 測試微調好的模型 - 使用 Playground





# Conclusion and key takeaways

1

LORA 讓我們能夠在單一 GPU 上微調 7B 參數的 LLM。在這種特殊情況下，使用最佳設定 ( $r=256$ 、 $\alpha=512$ ) 的 QLoRA，在 A100 上，使用 AdamW 進行 50,000 個訓練範例 (Alpaca 資料集) 的訓練，佔用了 17.86 GB 的內存，大約需要 3 小時。

2

- 使用 OpenAI finetune 模型時，要先估算訓練成本，再進行微調訓練
- 微調訓練有成本，使用微調後的模型做 Inference 也有成本，將會依 input, output 的 token 數量計價。

3

介紹了幾個特定領域 LLM 應用的方法 - Prompt 導引、大幅節省記憶體的 fine-tuning 方法 - QLoRA、二次預訓練等方法，企業可依自身的資源與需求來選擇或是併用不同方法。



Made with Gamma