Fine-tuning

Tirtharaj Dash

Department of CS & IS
BITS Pilani, Goa Campus, India
Homepage

February 2026

Outline

Recap: sampling from the LLM distribution

What is θ? The GPT block

Fine-tuning: adapting θ to a new distribution

Limitations: distribution shift and catastrophic forgetting

Parameter-efficient fine-tuning (PEFT) and LoRA

API-based fine-tuning

The LLM as a probability distribution

An LLM defines a joint distribution over token sequences:

$$P(\mathbf{x};\,\theta) \;=\; P(x_1,\ldots,x_T;\,\theta) \;=\; \prod_{t=1}^{T} P(x_t \mid x_1,\ldots,x_{t-1};\,\theta)$$

Generation = sampling from this distribution.
Given a prompt $(x_1,\ldots,x_s)$, we draw the next token:

$$x_{s+1} \;\sim\; P(\cdot \mid x_1,\ldots,x_s;\,\theta)$$

and repeat until an end-of-sequence token is produced.

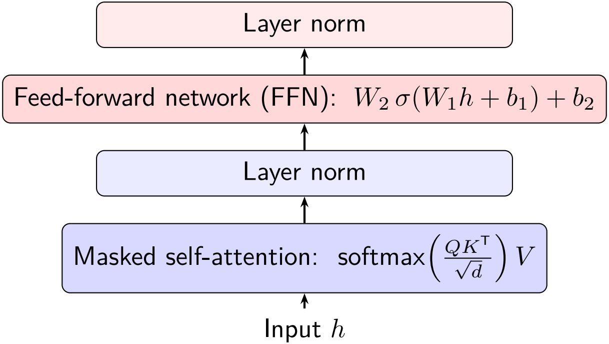
The LLM as a probability distribution

- *Greedy decoding*: always pick the most probable token $x_t = \arg\max P(\cdot \mid x_{<t}; \theta)$
- *Temperature sampling*: soften the distribution before sampling, controlling randomness
- *Top-p / top-k sampling*: restrict sampling to the most probable tokens

The model knows *nothing* beyond what is encoded in θ.

Inside a GPT block: what θ actually is

A GPT-style LLM stacks L identical transformer decoder blocks. Each block contains:



θ is the collection of all weight matrices across all L blocks:

$$\theta \;=\; \{\, W_Q^{(\ell)},\, W_K^{(\ell)},\, W_V^{(\ell)},\, W_O^{(\ell)},\, W_1^{(\ell)},\, W_2^{(\ell)},\, \gamma^{(\ell)},\, \beta^{(\ell)} \,\}_{\ell=1}^{L}$$

A 7B model has $\sim$7 billion such scalar values.

Pre-training: learning θ from data

Objective: minimise the negative log-likelihood over a massive corpus $\mathcal{D}_{\text{pre}}$:

$$\theta_0 \;=\; \arg\min_{\theta} \; -\mathbb{E}_{x \sim \mathcal{D}_{\text{pre}}} \left[\sum_{t=1}^{T} \log P(x_t \mid x_{<t};\, \theta) \right]$$

- $\mathcal{D}_{\text{pre}}$: trillions of tokens from the web, books, code, etc.
- After training, θ_0 encodes broad linguistic knowledge, world facts, and reasoning patterns
- The resulting model is a *generalist*: it can continue any text, but does not reliably follow instructions or specialise in any domain

$\Rightarrow$ θ_0 is the starting point. Fine-tuning adjusts it.

Fine-tuning as distribution shift

Key idea: the pre-training corpus $\mathcal{D}_{\text{pre}}$ defines one data distribution. A downstream task defines a different, narrower distribution $\mathcal{D}_{\text{ft}}$.

Fine-tuning continues gradient descent from θ_0 on $\mathcal{D}_{\text{ft}}$:

$$\theta^* \;=\; \arg\min_{\theta} \; -\frac{1}{N}\sum_{i=1}^{N}\sum_{t} \log P\Big(y_t^{(i)} \mid x^{(i)}, y_{<t}^{(i)};\, \theta\Big)$$

where $(x^{(i)}, y^{(i)})$ are (input, target) pairs and the loss is computed *only on the output tokens* y.

Fine-tuning as distribution shift

Pre-training distribution

General web text, books, code.
$|\mathcal{D}_{\mathsf{pre}}| \sim 10^{12}$ tokens.

Fine-tuning distribution

Curated (instruction, response) pairs.
$|\mathcal{D}_{\mathsf{ft}}| \sim 10^3$–$10^5$ examples.

The model retains representations from θ_0 while its output distribution shifts to match $\mathcal{D}_{\mathsf{ft}}$.

Two failure modes of fine-tuning

Distribution shift at test time

If the fine-tuning distribution $\mathcal{D}_{\text{ft}}$ differs from what the model will see at deployment, performance degrades.

$$\mathcal{D}_{\text{ft}} \neq \mathcal{D}_{\text{test}} \;\Rightarrow\; \text{generalisation gap}$$

Example: fine-tuned on clinical notes from one hospital system; deployed at another with different terminology and documentation style.

Two failure modes of fine-tuning

Catastrophic forgetting

Gradient updates that minimise loss on $\mathcal{D}_{\mathsf{ft}}$ may increase loss on $\mathcal{D}_{\mathsf{pre}}$. The model *forgets* general capabilities.

$$\theta^* \approx \arg\min \mathcal{L}_{\mathsf{ft}} \;\not\Rightarrow\; \theta^* \approx \arg\min \mathcal{L}_{\mathsf{pre}}$$

The two objectives can conflict, especially with a large learning rate or many fine-tuning steps.

Kirkpatrick et al., *Overcoming Catastrophic Forgetting*, PNAS 2017

Why not update all of θ?

For a model with P parameters, full fine-tuning requires storing in GPU memory:

What	**Memory (fp32)**	**7B model**
Model weights θ	$4P$ bytes	28 GB
Gradients $\nabla_\theta \mathcal{L}$	$4P$ bytes	28 GB
Adam optimiser states	$8P$ bytes	56 GB
Total	**16P bytes**	$>$ **112 GB**

This exceeds a single A100 GPU (80 GB). Even in fp16 it is borderline.

Why not update all of θ?

PEFT idea: freeze θ_0; introduce a small set of trainable parameters ϕ with $|\phi| \ll |\theta|$:

$$\theta^* = \underbrace{\theta_0}_{\text{frozen}} + \underbrace{\Delta\theta(\phi)}_{\text{trained}}$$

Train only ϕ. Gradient and optimiser memory scales with $|\phi|$, not $|\theta|$.

LoRA: the low-rank hypothesis

Empirical observation (Aghajanyan et al., 2021): the weight updates $\Delta\theta$ that arise during fine-tuning have low *intrinsic rank* — the update lives in a much smaller subspace than the full weight space.

LoRA (Hu et al., 2022) exploits this. For each weight matrix $W_0 \in \mathbb{R}^{d \times k}$, instead of learning a full ΔW, learn a rank-r factorisation:

$$\boxed{W \;=\; W_0 + \Delta W \;=\; W_0 + BA}$$

where $B \in \mathbb{R}^{d \times r},\;\; A \in \mathbb{R}^{r \times k},\;\; r \ll \min(d, k)$.

LoRA: the low-rank hypothesis

- W_0 is *frozen*; only A and B are trained
- A initialised with random Gaussian; B initialised to **zero** $\Rightarrow$ $\Delta W = 0$ at the start (no disruption to pre-trained behaviour)
- Forward pass: $h = W_0 x + \frac{\alpha}{r} BAx$ (α: scaling hyperparameter)

Hu et al., *LoRA: Low-Rank Adaptation of Large Language Models*, ICLR 2022

LoRA: the low-rank hypothesis

LoRA: parameter count:

For a single weight matrix $W_0 \in \mathbb{R}^{d \times k}$:

	Full fine-tuning	**LoRA (rank r)**
Parameters	$d \times k$	$r(d + k)$
Example (d=k=4096, r=8)	16.7M	65K
Reduction	—	$\sim 256\times$

LoRA: the low-rank hypothesis

After training: **merge** the adapter back into the base weight —

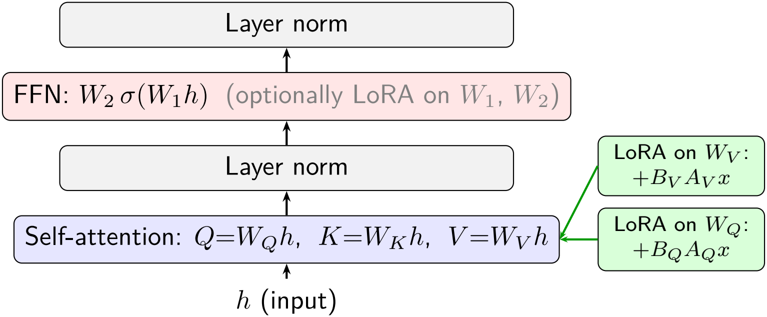
$$W^* = W_0 + BA$$

No inference overhead; the merged model is identical in structure to the original.

Typical choices: $r \in \{4, 8, 16, 64\}$; larger r = more capacity but more parameters. For most tasks $r = 8$ suffices.

LoRA applied to each GPT block

LoRA adapters are inserted into the *linear projection matrices* of each transformer block. Typically applied to the attention projections:



- All blue (attention) and red (FFN) weights are **frozen**
- Green LoRA adapters (B, A) are **trained**
- This is repeated independently for *each* of the L blocks

Fine-tuning without touching the model

Not everyone can run fine-tuning locally. Cloud providers expose fine-tuning as an API:

Provider	**Models**	**Method (reported)**
OpenAI	GPT-4o mini, GPT-3.5	SFT (likely LoRA internally)
Google	Gemini 1.5 Flash	SFT + RLHF
Anthropic	Claude (limited access)	Constitutional AI + RLHF
Together AI	LLaMA, Mistral, etc.	LoRA / QLoRA
Hugging Face	Any open model	Full FT / LoRA via AutoTrain

Workflow: upload $\mathcal{D}_{\text{ft}}$ in JSONL format (one `{prompt, completion}` per line) $\rightarrow$ submit job $\rightarrow$ receive a fine-tuned model endpoint.

Fine-tuning without touching the model

Trade-offs vs. local fine-tuning

- No GPU needed; scales to large models; simple API
- Data leaves your machine; no control over PEFT rank or hyperparameters; cost per token at inference; model not portable

Summary

- An LLM defines $P(\mathbf{x};\theta)$; generation is sampling from this distribution; θ is all weight matrices in all L blocks
- Pre-training learns θ_0 from massive general data; fine-tuning shifts it to θ^* on small task-specific data
- Fine-tuning loss: cross-entropy on *output tokens only*
- **Distribution shift**: mismatch between $\mathcal{D}_{\text{ft}}$ and deployment distribution causes generalisation failure
- **Catastrophic forgetting**: $\mathcal{L}_{\text{ft}}$ and $\mathcal{L}_{\text{pre}}$ objectives can conflict
- **LoRA**: approximate $\Delta W \approx BA$ with rank r; freeze W_0, train only B and A; merge at inference
- LoRA is applied independently to each GPT block's projection matrices
- **API fine-tuning**: upload data, get a fine-tuned endpoint — convenient but at the cost of control and data privacy