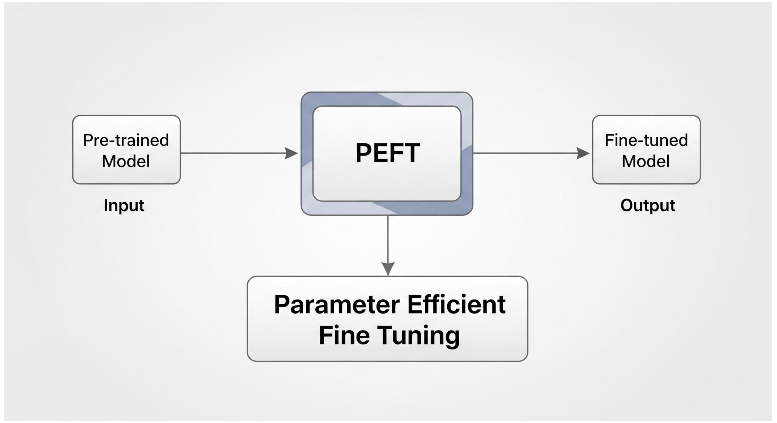
Fun with Small Language Models

Dr Martin Callaghan
The Open University

But really it’s about:

Fine-tuning energy efficient small language models for task and knowledge specialism

What this session is about



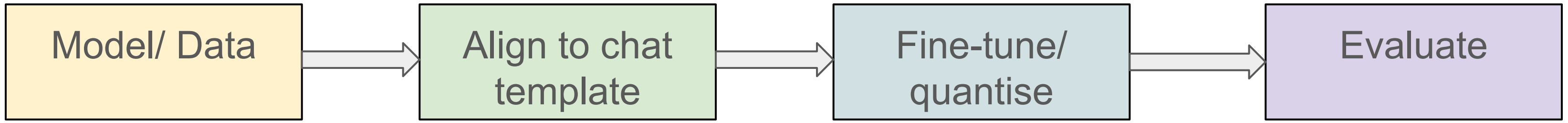
- Think of this as a *meta-workshop*
  - *Your comments and input are important*
  - *What do you want to know (and do) more of?*
  - *What do you want to know (and do) less of?*

- Based on some industry training (and own research) done over the last year

- Primarily an opportunity to experiment with some **PEFT** techniques on cheap/ free accelerators.
  - With a view to using fine-tuned small models in your own work/ projects
  - On a ‘fun’ example scenario

Agenda

1. Background and basics
2. Experiment 1: Comparing base model outputs with instruction-tuned models
3. Step 1: *Choosing the right model and data for your task*
4. Step 2: *Aligning the data to the chat template*
5. Step 3: *Fine-tuning run & quantisation*
6. Experiment 2: Fine-tuning
7. Step 4: *Evaluation*
8. Experiment 3: Evaluation methods
9. Discussion about deployment



What makes a model 'small'?

- **Definition**: *Models with fewer parameters* (typically < 32B). We are using Google's Gemma 2 (2B parameters).
- **The Trade-off:**
  - **Pros**: Runs on consumer hardware (free Colab T4s), lower latency, energy-efficient, easier to deploy locally (privacy/air-gapped).
  - **Cons**: Less "world knowledge" than GPT-5, struggles with complex reasoning chains, requires focused fine-tuning to excel.
- **The Toolset**:
  - **Hugging Face**: The ecosystem/hub.
  - **Unsloth**: The optimisation library (makes training 2x faster, uses 60% less memory).
  - **CodeCarbon**: For tracking the environmental impact.

Base vs. Instruction Tuned Models (Notebook 1)

The Base Model:

- Trained on raw internet text (Wikipedia, code, books).
- **Function**: An autocomplete engine. It predicts the next token based on probability.
- **Behaviour**: If prompted with a question, it might just generate more questions (mimicking a list).

The Instruction Tuned (IT) Model:

- Starts as a Base model but undergoes Supervised Fine-Tuning (SFT).
- **Function**: A chat engine.
- **Behaviour**: Understands the concept of "User" vs. "Assistant" and follows intent.

Workshop Goal: We will take an IT model and steer its "personality" using further fine-tuning.

Notebook 1

Your turn!

Notebook 2: Choosing the Model & Data

Why Gemma 2 2B?

- It fits comfortably into the 16GB VRAM limit of a free Google Colab T4 GPU.
- It is a "gated" model (requires licence acceptance), ensuring responsible use.

The Data Strategy:

- We don't always need massive datasets.
- **Synthetic Data**: For this workshop, we generate data "on the fly".
- **The Task**: *Style Transfer*. Mapping modern complaints (e.g., "stuck in traffic") to Stoic responses (e.g., "it is an external event").
- **Volume**: Even 60–100 high-quality examples can significantly shift the model's tone.

Notebook 2: Aligning to the Chat Template

The Challenge: The model expects text in a specific format to know who is speaking.

The Solution: Chat Templates.

- **Gemma's Format**: Uses specific control tokens: **`<start_of_turn>user ... <end_of_turn><start_of_turn>model ...`**
- **Implementation**:
    - We use the tokenizer's **`apply_chat_template`** function.
- **Crucial Step**:
    - If the data isn't formatted correctly, the model learns nonsense.

Notebook 2: Fine-Tuning & Quantisation

The Technique: QLoRA (Quantised Low-Rank Adaptation)

- **Quantisation** (4-bit): We compress the model weights to reduce memory usage (loading a 2B model in 4-bit takes very little VRAM).
- **LoRA (Adapters)**: Instead of retraining the whole brain (expensive!), we attach small, trainable "adapter" layers to specific modules (e.g., `q_proj`, `v_proj`).
- We only train roughly 1–2% of the total parameters.

Unsloth's Role: Optimises the backpropagation steps, making the process feasible in a short workshop timeframe.

The problem with full fine-tuning

Full Fine-Tuning:

- Traditionally, fine-tuning meant updating all parameters in the model (e.g., all 2 billion weights in Gemma 2).
- The Cost: Requires massive amounts of VRAM to store the model weights, gradients, and optimiser states (often 3–4x the model size).
- Catastrophic Forgetting:** The model may learn the new task perfectly but "overwrite" its general knowledge (e.g., forgetting how to speak English while learning to code).

The LoRA (Low-Rank Adaptation) Solution:

- **Freeze the Weights**: We lock the original pre-trained model weights. They do not change.
- **Inject Adapters**: We insert tiny, trainable rank-decomposition matrices into the model's layers.
- **The Maths**: Instead of updating a massive weight matrix **W**, we train two tiny matrices **A** and **B** so that $\boldsymbol{\Delta W = A \times B}$
- **Efficiency:** We only train < 1% of the parameters, but the results are comparable to full fine-tuning.

Analogy: Instead of rewriting a textbook to add notes, we stick post-it notes on the pages. The book stays the same; the information is augmented.

QLoRA: The *bit* that makes it all *fit*

The "Q" stands for Quantisation:

- Standard models use 16-bit or 32-bit precision for numbers.
- 4-bit Quantisation: We compress the frozen base model weights into 4-bit "Normal Float" (NF4) format.
- A 2B parameter model drops from requiring ~4GB of VRAM to just ~1.5GB for the base weights.

How QLoRA works:

1. **Load**: The base model is loaded in 4-bit (low precision).
2. **Train**: The LoRA adapters are created in 16-bit (higher precision) to ensure accurate learning.
3. **Backpropagate**: Gradients are backpropagated through the frozen 4-bit weights to update the 16-bit adapters.

Why this is important:

- Without QLoRA, fine-tuning even a small model would require an expensive enterprise GPU (A100).
- With QLoRA: We can fine-tune on the free Google Colab T4 (16GB VRAM) with memory to spare.
- Unsloth: The library we are using optimises this further, handling the de-quantisation calculations faster than standard Hugging Face implementations.

Notebook 2: The Training Run

Hyperparameters:

- `max_steps`: Set to 60 for the demo (approx. 3 minutes).
- `learning_rate`: 2e-4 (standard for QLoRA).
- `rank (r)`: 16 (determines the complexity of the adapters).

Energy Efficiency:

- Using **CodeCarbon** to track emissions.
- Fine-tuning an SLM consumes negligible power compared to training a foundation model from scratch.

Notebook 2

Your turn!

Notebook 3: Evaluation Methods

Heuristic Metrics (The "Stoic Score"):

- Counting keywords: *virtue, control, nature, reason, indifference*.
- Simple, fast, and free.

LLM-as-a-Judge (The Professional Approach):

- Using a larger model (like GPT-5 or Claude) to grade the response based on a rubric.

Qualitative Analysis:

- Reading the outputs side-by-side.
- **Result**: The model shifts from generic empathetic advice (*"I'm sorry, update your CV"*) to philosophical framing (*"This is outside your control, focus on your reaction"*).

Notebook 3

Your turn!

Where do we go from here?

Saving the Model:

- We save the *adapters* (small files, ~50MB), **not** the whole model.

Inference:

- Load Base Model + Load Adapters = *Stoic Philosopher*.

Deployment Options:

- **Merge & Export**: Merge adapters into the base model and export to GGUF format.
- **Local Run**: Run on a laptop (or HPC cluster) using Ollama, Llama.cpp, vLLM etc.
- **Edge Devices**: Perfect for privacy-focused applications where data cannot leave the device.