1. What is fine-tuning, and why is it needed?

Fine-tuning is the process of taking a pre-trained Large Language Model (like Llama 3 or Mistral) and training it further on a smaller, specific dataset to specialize it for a particular task.[1]

Why it is needed:

- **Domain Specialization:** Base models know general English but struggle with niche jargon (e.g., legal contracts, medical reports, internal company lingo).[2]
- **Behavior Adaptation:** You can force a model to follow strict formats (e.g., "Always output JSON," "Never apologize," or "Speak like a pirate").
- **Efficiency:** A smaller fine-tuned model (e.g., 7B parameters) can often outperform a massive general model (e.g., GPT-4) on a specific task, saving inference costs.[3]

---

2. Which scenarios require fine-tuning?

You should fine-tune when **prompt engineering** and **RAG (Retrieval-Augmented Generation)** are insufficient.

- **Style/Tone Consistency:** You need the model to mimic a specific brand voice or character.[4]
- **Format Enforcement:** You need reliable structured output (SQL, JSON, XML) that RAG fails to guarantee.
- **New Knowledge Injection:** While controversial (RAG is usually better for facts), fine-tuning works well for teaching "patterns" of knowledge, such as a new programming language syntax.
- **Latency-Critical Apps:** You want to replace a slow, complex RAG pipeline with a single, fast, specialized model.[5]

---

3. How to make the decision of fine-tuning?

Use this decision framework:

1. **Does the model need new knowledge?** $\rightarrow$ Use **RAG** (Search & Retrieval).
2. **Does the model need to learn a new format, style, or behavior?** $\rightarrow$ Use **Fine-Tuning**.
3. **Do you have high-quality training data?** [6]$\rightarrow$ If yes, Fine-Tuning is viable.[7] If no, stick to Prompt Engineering.
4. **Is privacy paramount?** $\rightarrow$ Fine-tuning a local model ensures data never leaves your infrastructure (unlike calling an external API).

4. How do you improve the model to answer only if there is sufficient context?

To prevent hallucinations where the model makes up answers when it doesn't know, you must include **"Negative Samples"** or **"Rejection Sampling"** in your fine-tuning dataset.[8]

- **Strategy:** Create training examples where the input question **cannot** be answered by the provided context.
- **Target Output:** Train the model to output a specific refusal token or phrase, such as *"I do not have enough information to answer this based on the context provided."*
- **Ratio:** Ensure 10-20% of your dataset consists of these unanswerable questions so the model learns the boundary of "knowing vs. guessing."

---

5. How to create fine-tuning datasets for Q&A?

A high-quality dataset is more important than the model choice.

1. **Raw Data Collection:** Gather PDFs, emails, or logs.
2. **Chunking:** Break text into logical segments (context).
3. **Question Generation:** Use a strong model (like GPT-4) to read a chunk and generate diverse questions (simple, complex, reasoning-based) answerable *only* from that chunk.
4. **Formatting:** Structure the data into **Instruction-Input-Output** pairs (JSONL format is standard).[9]
    - *Instruction:* "Answer the question based on the context."
    - *Input:* [Context Chunk] + [Question]
    - *Output:* [Correct Answer]
5. **Quality Check:** Remove duplicates and ambiguous questions.[10]

---

6. How to set hyperparameters for fine-tuning?

Fine-tuning requires gentler updates than pre-training to avoid destroying the model's existing knowledge.

- **Learning Rate:** Low (e.g., `2e-5` to `2e-4`). Too high will break the model (catastrophic forgetting).
- **Epochs:** Few (1 to 3 epochs).[11] LLMs overfit quickly.
- **Batch Size:** High effective batch size (e.g., 32 or 64). Use **Gradient Accumulation** if your GPU memory is small to simulate larger batch sizes.[12]
- **Scheduler: Cosine** or **Linear** decay is standard.[13]

---

7. How to estimate infrastructure requirements?

Memory usage is dominated by the model weights, optimizer states, and gradients.

Rule of Thumb Formula:

$$\text{VRAM Needed} \approx (\text{Model Params in Billions}) \times (\text{Precision Bytes}) \times 4$$

- **Example (Full Fine-Tuning 7B Model at FP16):**
    - 7B Params [14] $\times$ 2 bytes (FP16) = **14GB** (Just to load model).[15]
    - Training requires ~3-4x more for gradients/optimizer states $\approx$ **50GB+ VRAM** (Requires A100 or multiple GPUs).
- **Optimization:** Using **QLoRA** (4-bit quantization + LoRA) reduces this drastically.[16] A 7B model can be trained on **<10GB VRAM**.

---

8. How to fine-tune LLM on consumer hardware?

To train on a gaming GPU (e.g., RTX 3090/4090 with 24GB VRAM):[17]

1. **Quantization (4-bit / 8-bit):** Load the base model in 4-bit precision (using `bitsandbytes` library).[18] This shrinks a 7B model from 14GB to ~4GB.
2. **PEFT (LoRA):** Freeze the main model. Only train tiny adapter layers (Low-Rank Adapters) which adds <1% memory overhead.
3. **Gradient Checkpointing:** Trades compute speed for memory savings by not storing all intermediate activations.
4. **Paged Optimizers:** Offload optimizer states to CPU RAM if GPU VRAM fills up.

---

9. What are the different categories of the PEFT method?

PEFT (Parameter-Efficient Fine-Tuning) updates only a small subset of parameters.[19]

1. **Additive Methods:** Add new layers to the model.
    - *Adapters:* Tiny layers inserted between transformer layers.[20]
    - *Soft Prompts (Prompt Tuning):* Trainable vectors added to the input sequence.
2. **Selective Methods:** Only update specific existing bias terms or layers (e.g., **BitFit**).[21]
3. **Re-parameterization Methods:** Represent weight updates as low-rank matrices (e.g., **LoRA**).[22]

---

10. What is catastrophic forgetting in LLMs?

Catastrophic Forgetting occurs when a model learns a new task (e.g., coding) so well that it "overwrites" its previous knowledge (e.g., how to speak English or do math).[23]

- *Symptom:* A chatbot fine-tuned on medical data suddenly forgets how to summarize general news.[24]
- *Prevention:*
    - Use PEFT (freezes original weights).[25]
    - Replay Buffer: Mix in some general data (e.g., Wikipedia) along with your new specialized data during training.

---

11. What are different re-parameterized methods for fine-tuning?

Re-parameterization methods reduce the number of trainable parameters by transforming the weight update matrix.[26]

- **LoRA (Low-Rank Adaptation):** Instead of updating the full weight matrix [27]W, it learns two smaller matrices [28]A and [29]B such that [30]$\Delta W = B \times A$.[31] This reduces trainable parameters by 10,000x.
- **DoRA (Weight-Decomposed Low-Rank Adaptation):** An improvement on LoRA that decomposes weights into **Magnitude** and **Direction**, allowing the model to learn more efficiently, similar to full fine-tuning performance.
- **QLoRA:** LoRA combined with 4-bit Quantization for extreme memory efficiency.[32]