AMERICAN UNIVERSITY OF BEIRUT

Fall 2025

EECE 503P/798S: Agentic Systems

C7 - Fine tuning

Lesson Objectives

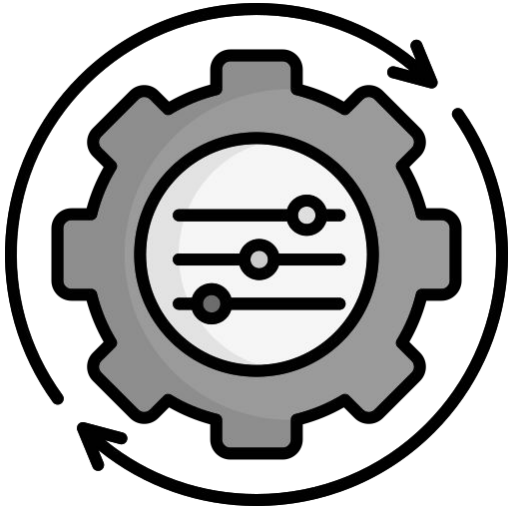
By the end of this lesson, you should be able to:

- Explain **when** and **why** fine-tuning is necessary for agentic systems
- Compare full **fine-tuning** and **parameter-efficient methods** (LoRA, QLoRA)
- **Prepare, clean,** and **format** data for LLM fine-tuning
- Execute the **fine-tuning process**, from forward pass to optimization
- **Evaluate** fine-tuned models using both technical and business metrics
- Apply **best practices** to deploy, monitor, and maintain fine-tuned models

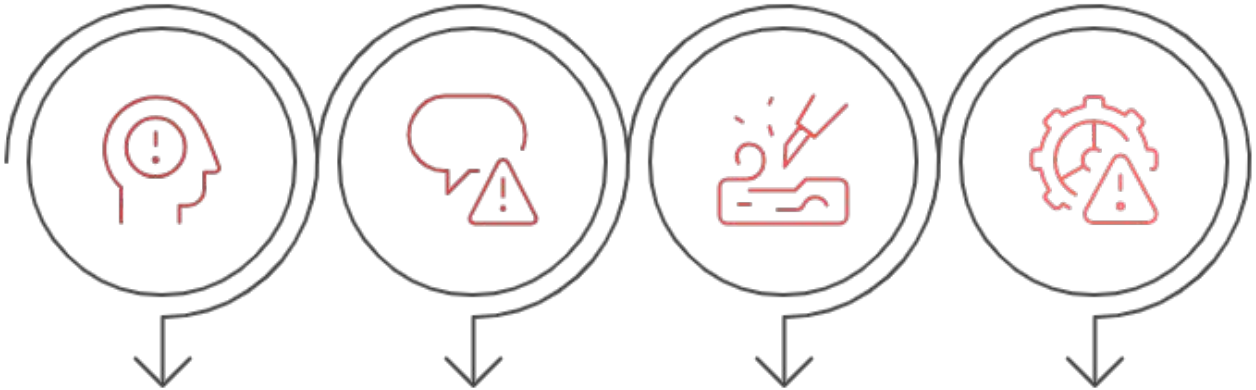
Introduction

Smart… But Not Yet an Expert

- Agents can **plan, reason,** and take **action**
- But without **domain-specific knowledge**, they're forced to **guess**
- Foundation models know a lot, but not your **world, workflows,** or **goals**
- Even the smartest agents are **limited** by the model's **generic knowledge**
- **Fine-tuning** bridges that gap



When General Knowledge Isn’t Enough



Term Misinterpretation

Models may not understand specialized vocabulary.

Response Issues

Answers can be off-topic or contradictory.

Prompt Dependence

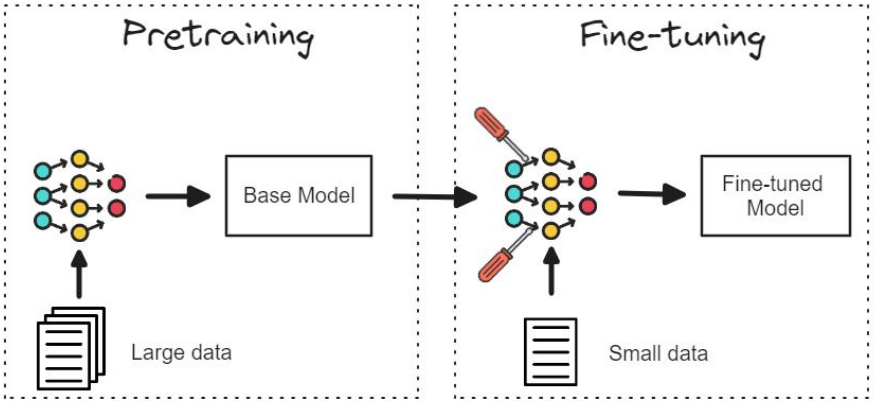
Performance relies heavily on prompt wording.

Decision Problems

Reasoning can be slow and untrustworthy.

Fine-tuning

- You start with a **general model** that already "knows a lot"
- Then you **train** it a little more using your own data (usually smaller and task-specific)
- This makes the model **better** at your particular problem

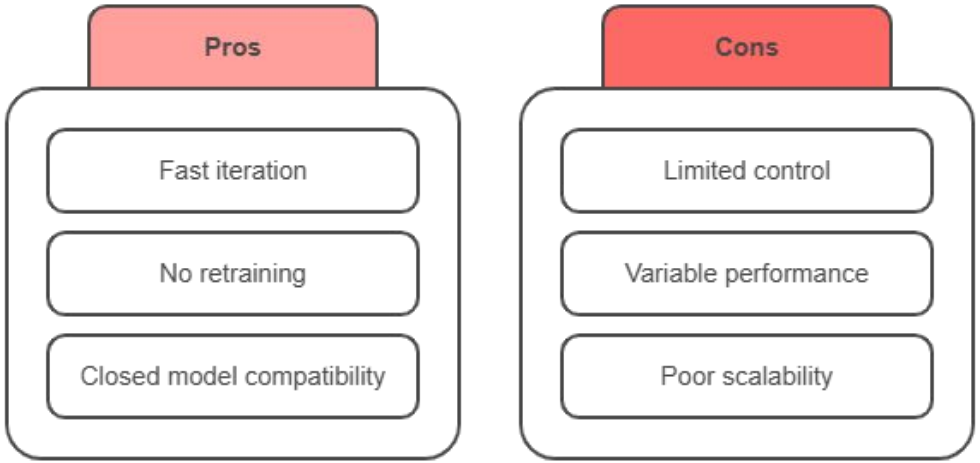


Key Objectives of Fine-Tuning

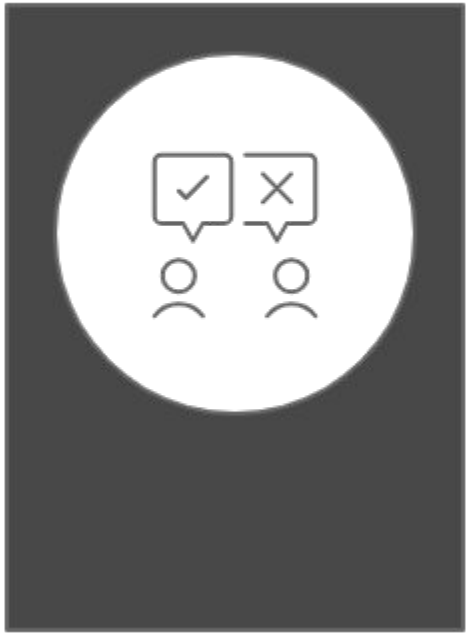
Use Cases

Use Case	Description	Example
Personalization	Teach the model a user's tone, style, or memory	Personal assistant that knows your writing style
Domain Adaptation	Inject domain-specific jargon and logic	Legal chatbot, radiology report generator
Instruction Alignment	Make model better follow human instructions	Custom support bot, educational tutor
Safety / Guardrails	Enforce certain behaviors or restrictions	Refuse harmful content, safer completions

Prompt Engineering



Retrieval-Augmented Generation (RAG)



Pros

- Up-to-date knowledge
- Fixed model size
- Reduced retraining

Cons

- Slower inference
- Robust retriever needed
- More maintenance

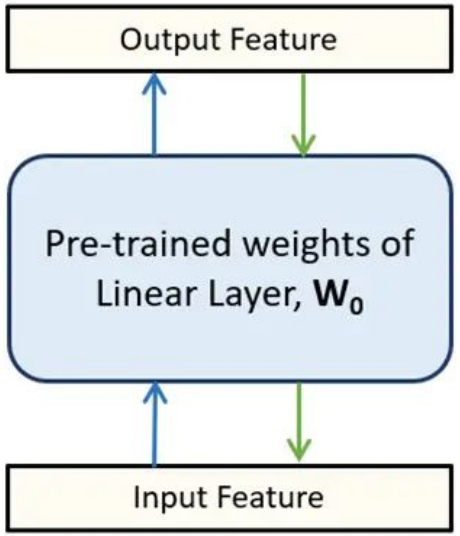
Choosing Between Fine-Tuning and RAG

Fine-Tuning	RAG
Specialized Task Focus	Real-Time Information Integration
Large Labeled Data Needed	External Data Retrieval
Computationally Expensive	Efficient Database Use

Fine-Tuning Paradigms – The Big Picture

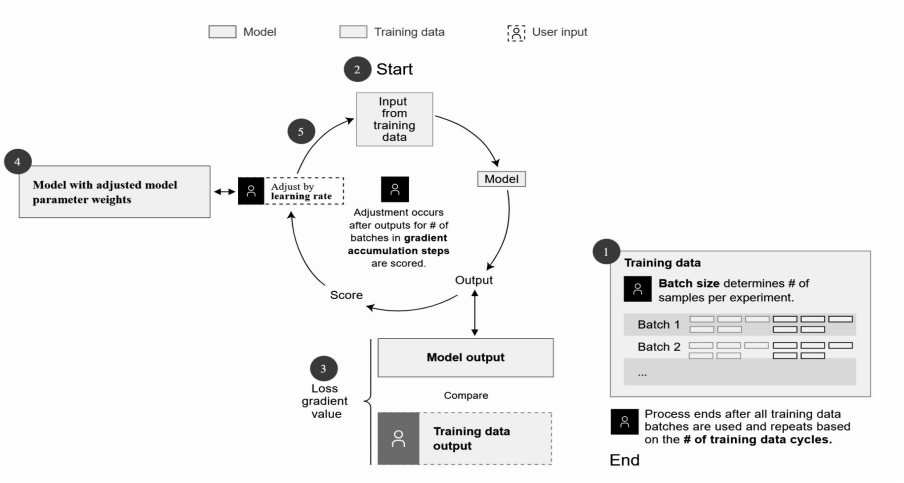
Full Fine-Tuning

- Update **all** model **parameters** (billions!)
- Highest **flexibility** and potential **performance**
- Best for **domain shifts** or **major behavior changes**
- **Example:** Fine-tuning a pretrained LLaMA model on thousands of medical Q&A pairs

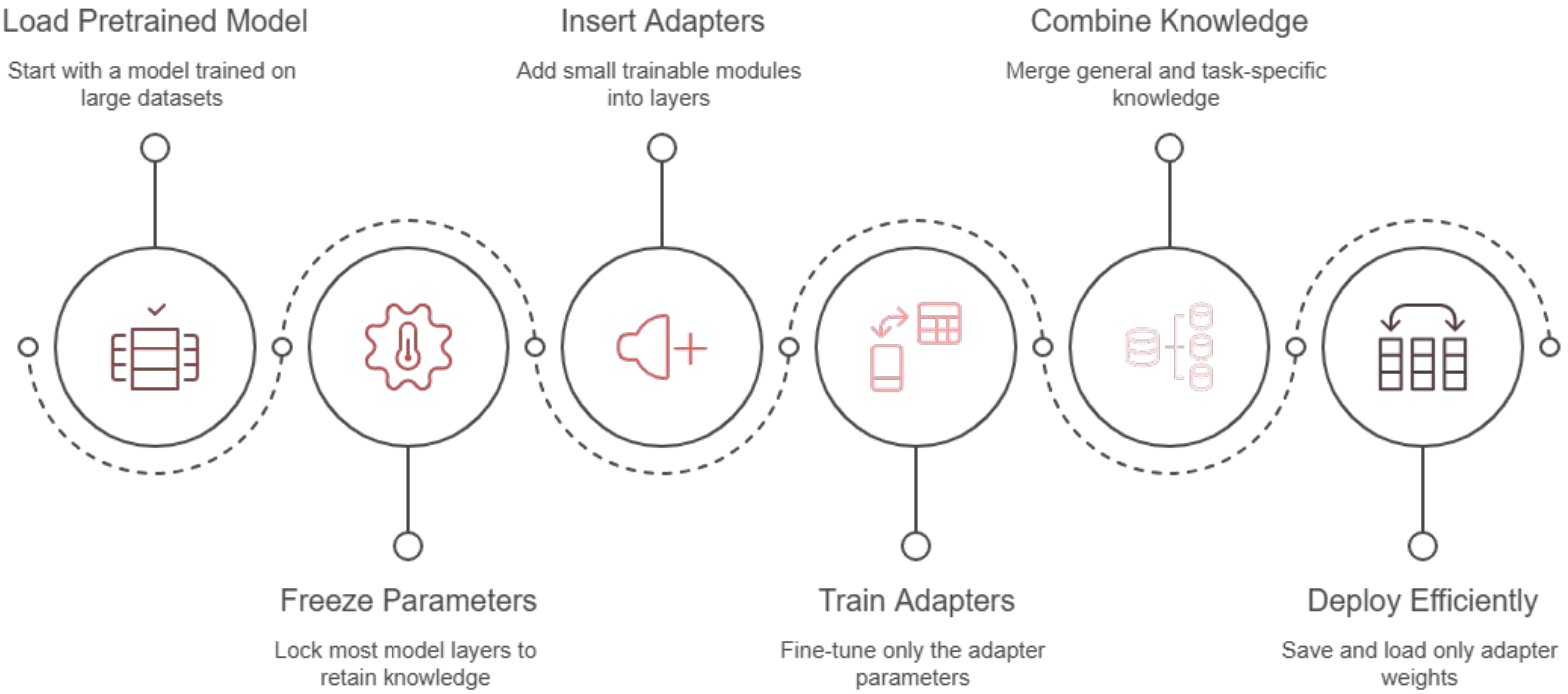


Full Fine-Tuning

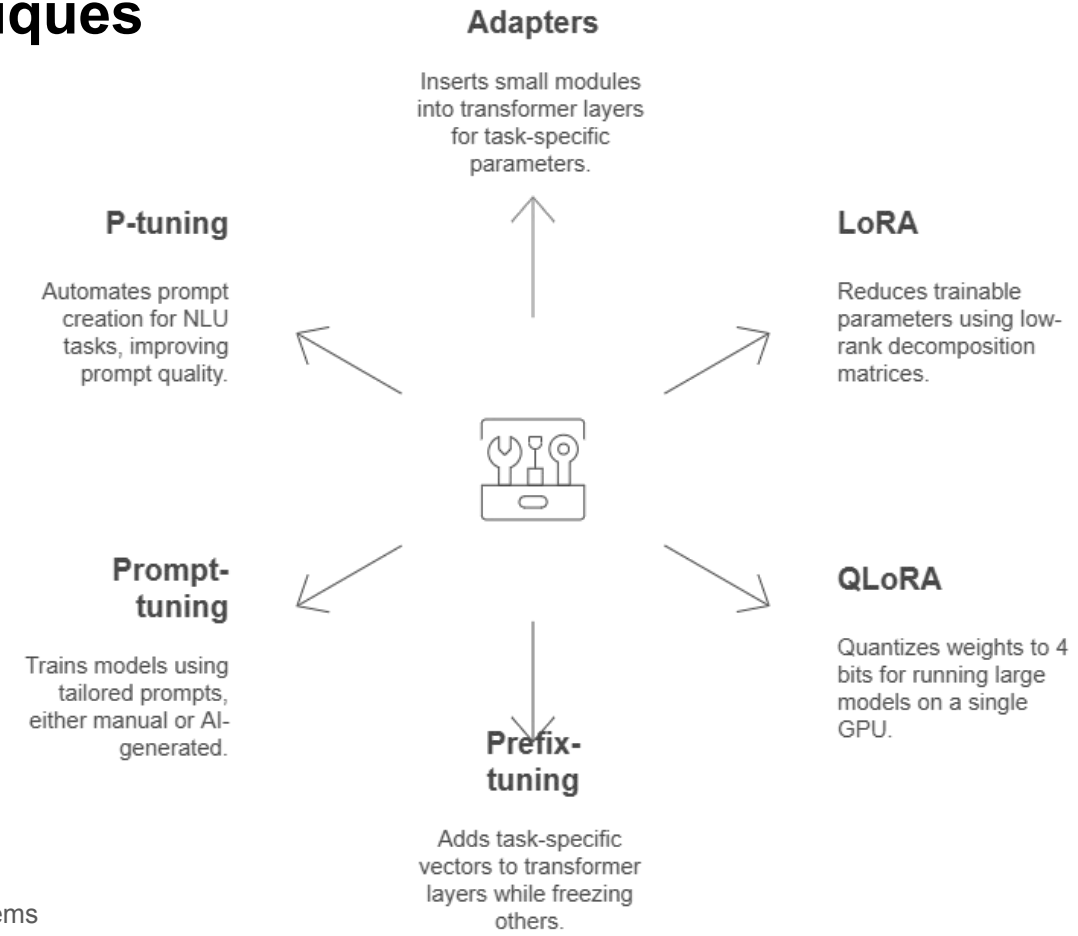
1. Prepare and **batch** the input **text** for the model
2. Feed the **batch** into the **model** to produce **predictions**
3. Compare **predictions** with targets to compute the **loss** and its **gradient**
4. Adjust **parameters** using the gradients, learning rate, and accumulation steps
5. Process every batch in the dataset, then **loop** over multiple epochs



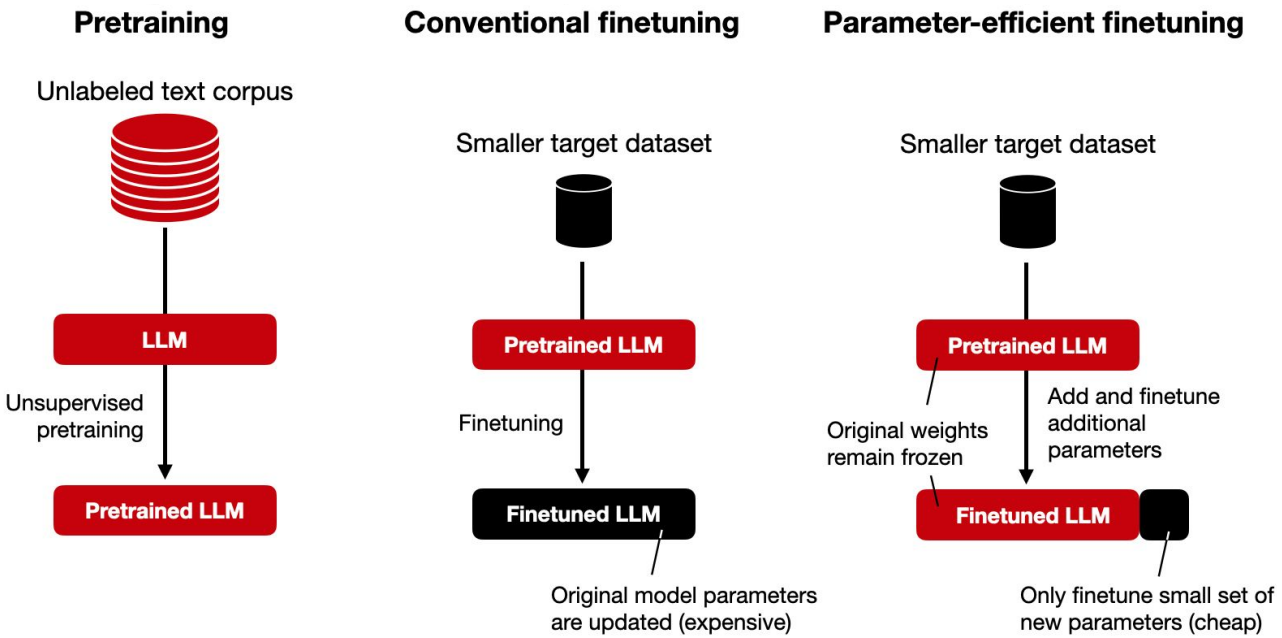
Parameter-Efficient Fine-Tuning (PEFT)



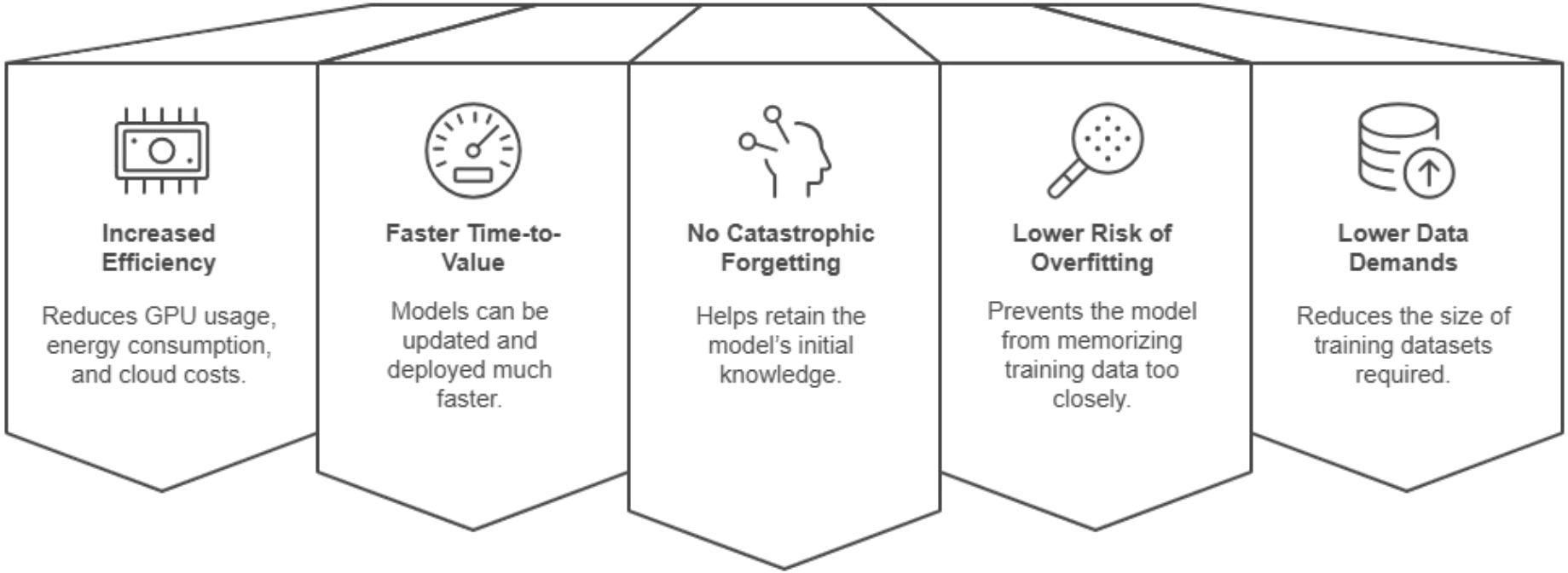
PEFT Techniques



Full Fine-Tuning vs. PEFT



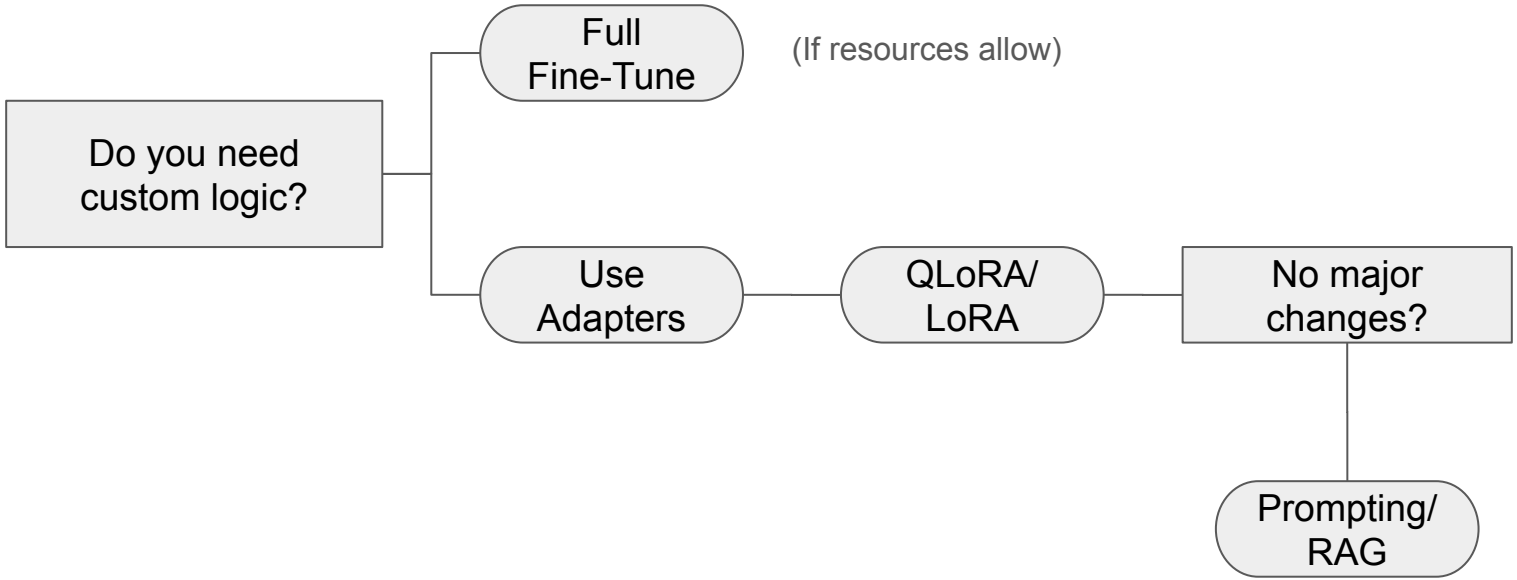
Benefits of PEFT



Comparison of Fine-Tuning Paradigms

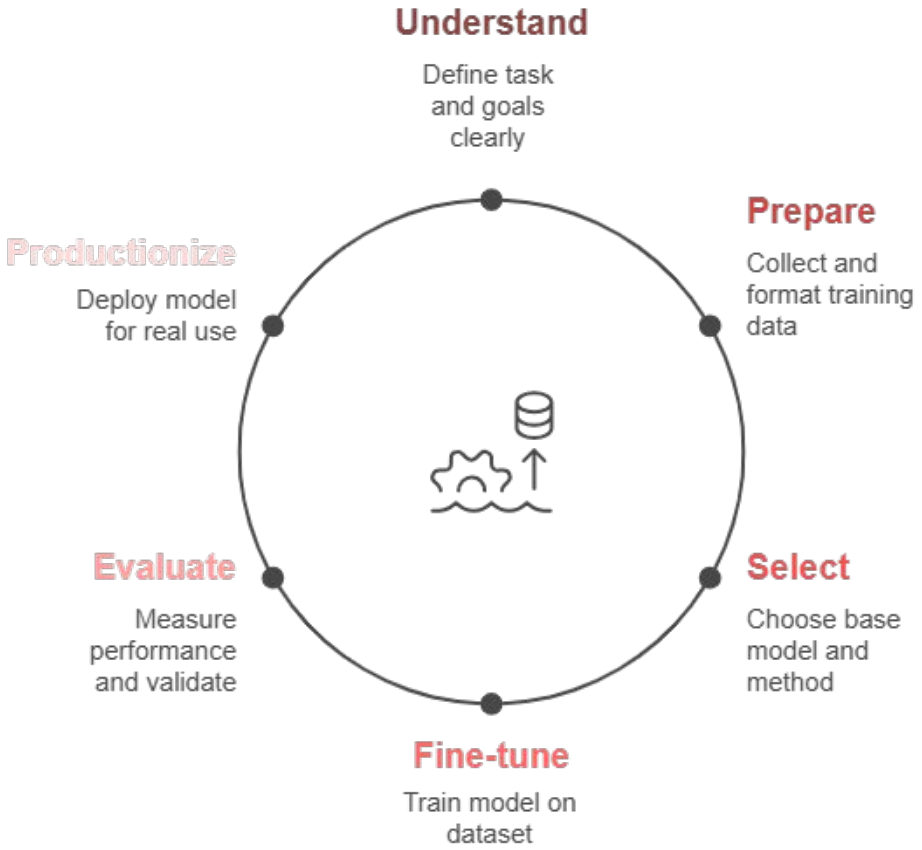
Characteristic	Full Fine-Tuning	LoRA, QLoRA	Adapters	Instruction Tuning
Scope	All weights	Small % of weights	Extra layers only	Behavior alignment
Typical Use Case	Deep domain/behavioral change	Efficient adaptation w/ low resources	Modular, task-specific tuning	General-purpose response tuning

Decision Flow for Choosing Fine-Tuning Strategies



Fine-Tuning Process

Fine-Tuning Process



1. Understand

- Gather business **requirements** and **goals**
- Identify **key performance metrics**, especially business-focused ones
- Assess the **data**: amount, quality, and format
- Research existing **solutions** and **baseline** models
- Compare relevant **language models**
- Define non-functional needs like **cost, scalability,** and **latency**
- Plan **budget, resources,** and **timeline** for the project

The Importance of a Baseline

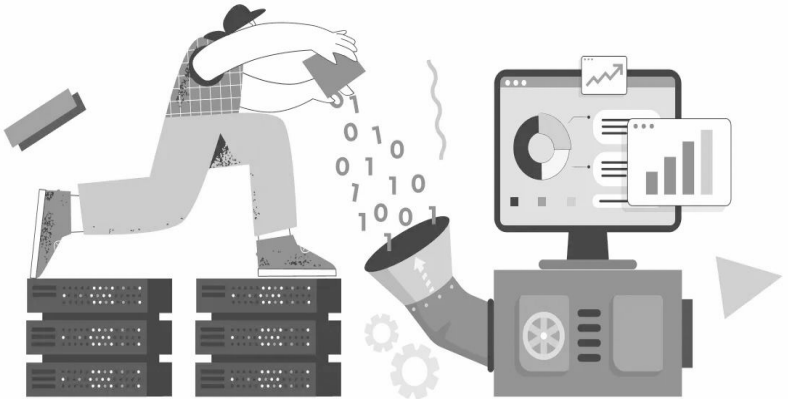
Start cheap and simple

Gives a benchmark to improve on

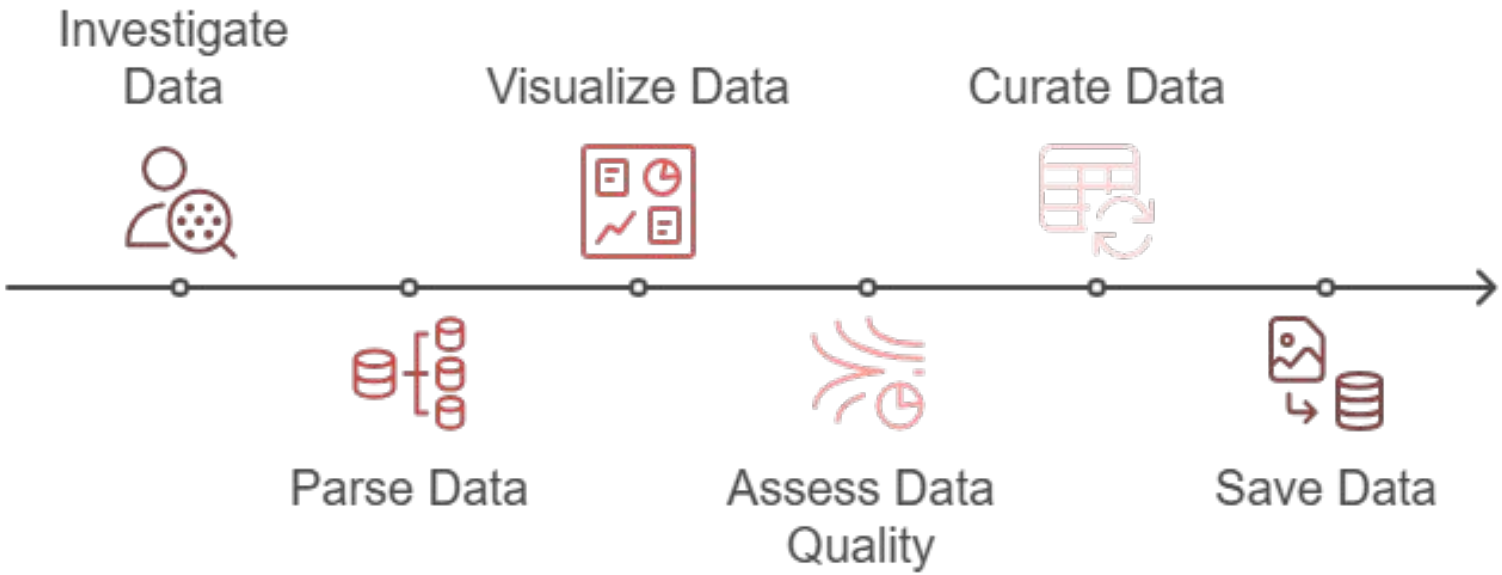
An LLM might not be the right solution

2. Prepare

- **Collect** and **clean** data relevant to your task
- **Format data** (e.g., prompt-completion pairs)
- **Split** into training, validation, and test sets

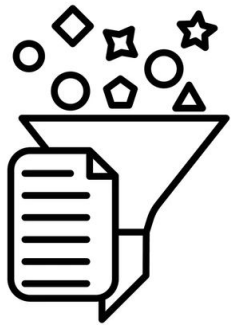


Data Preparation



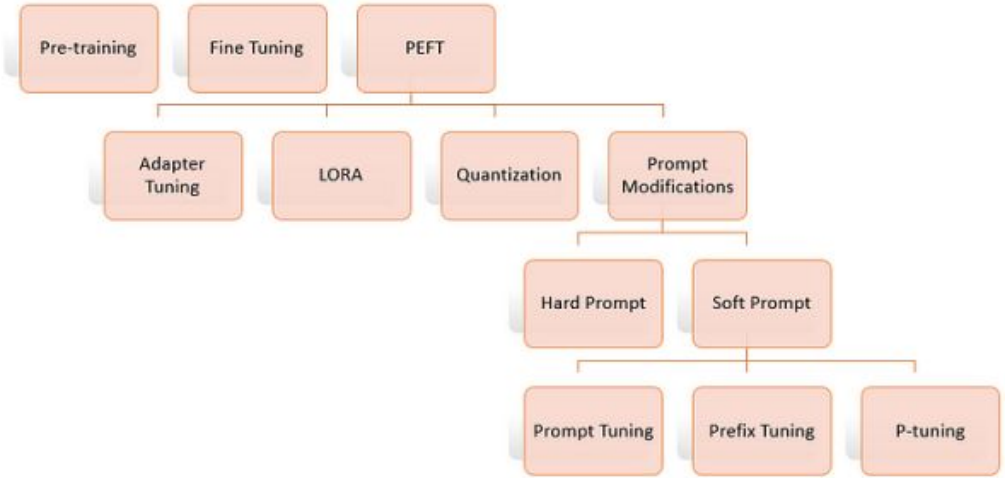
Curating Data

- Are there any **rows** that are **irrelevant** to my project's goal?
- Which columns contain **missing values**, and how should I **handle** them?
- Do **units, formats,** and **labels** remain consistent across the dataset?
- Are there any **duplicate entries** that should be removed?
- Do **numerical values** fall within a **realistic range**?
- Are all **date** and **time** values in a standard, usable format?

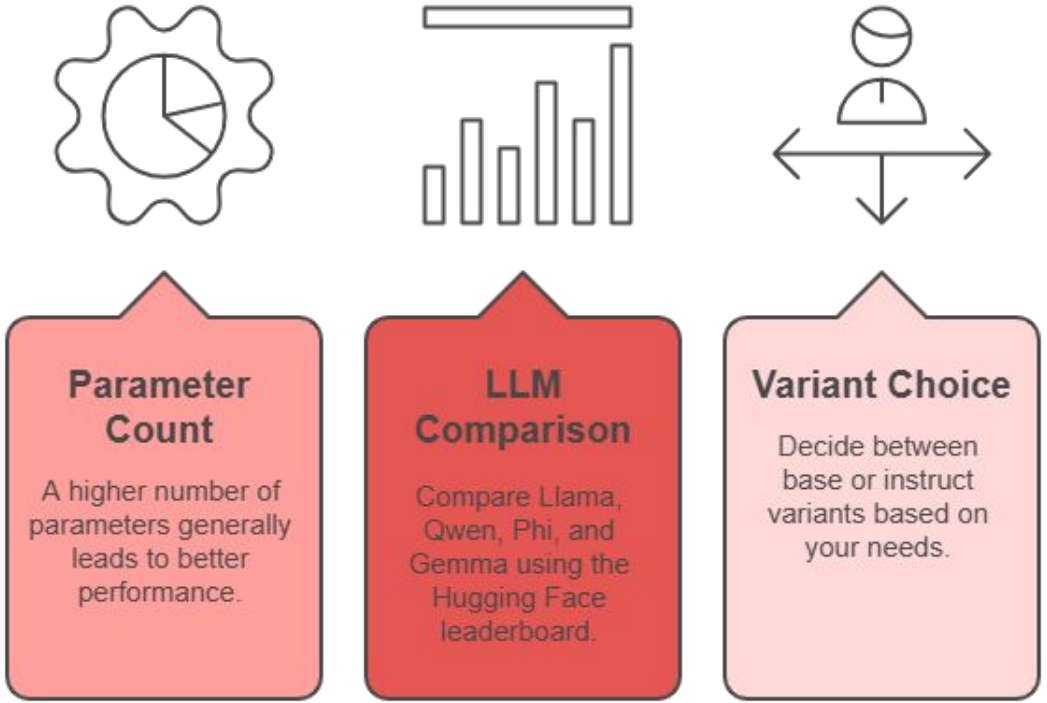


3. Select

- Choose a suitable base **pretrained** model (open-source or frontier)
- Decide on the **fine-tuning method** (full fine-tuning, LoRA, adapters, etc.)
- Set key **hyperparameters** (learning rate, batch size, epochs)

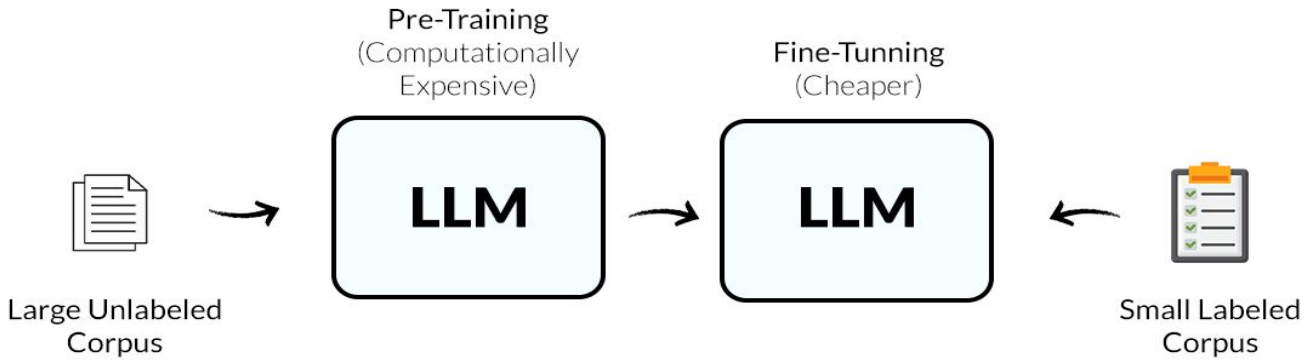


Which Model?


Parameter Count
A higher number of parameters generally leads to better performance.
LLM Comparison
Compare Llama, Qwen, Phi, and Gemma using the Hugging Face leaderboard.
Variant Choice
Decide between base or instruct variants based on your needs.

4. Fine-Tuning

- **Train** model on prepared data
- Monitor training **loss** and **validation** performance
- Adjust **hyperparameters** if needed



5. Evaluate

- Assess **performance** on **validation/test** data
- Compare against **baseline**
- How to **evaluate performance**?

Model-Centric Metrics

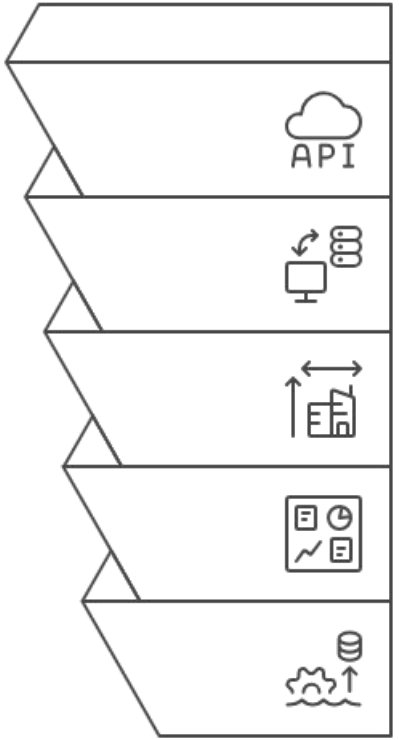
Focus on technical performance

Business Metrics

Focus on real-world impact

6. Productionize

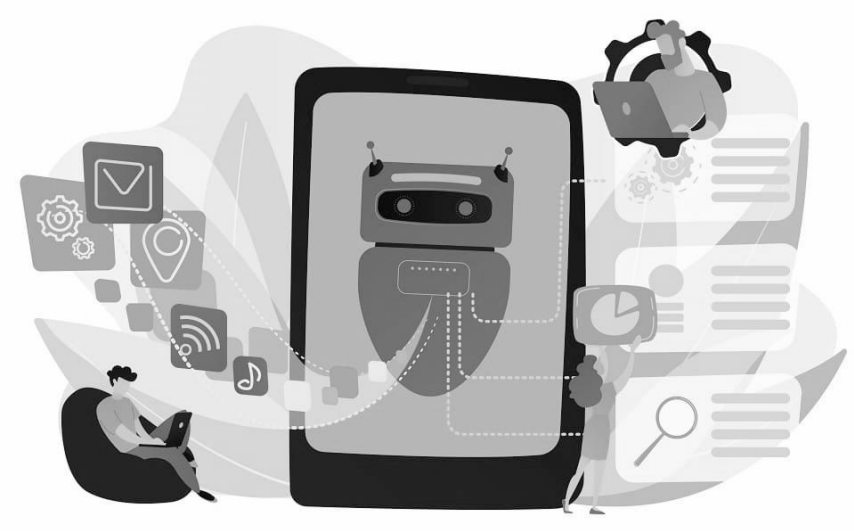
1. **Define API**
   Establish communication interface
2. **Choose Architecture**
   Select hosting and deployment setup
3. **Plan Scaling**
   Prepare for growth and demand
4. **Track Metrics**
   Monitor business-focused data
5. **Retrain Model**
   Continuously improve performance



Fine-Tuning of an LLM for Employee QA Chatbot

1. Understand

- **Goal:** Build a QA bot answering employee questions (e.g., "When was Alice hired?")



Fine-Tuning of an LLM for Employee QA Chatbot

2. Prepare

id	name	department	hire_date	salary
001	Alice Smith	Engineering	2020-04-10	85000

Convert to QA pairs

{"context": "Employee: Alice Smith, Dept: Engineering, Hire Date: 2020-04-10, Salary: 85000",
"question": "When was Alice Smith hired?",
"answer": "2020-04-10" }

Fine-Tuning of an LLM for Employee QA Chatbot

AMERICAN UNIVERSITY OF BEIRUT

3. Select

- Choose LLaMA 8B, fine-tune with LoRA for efficiency

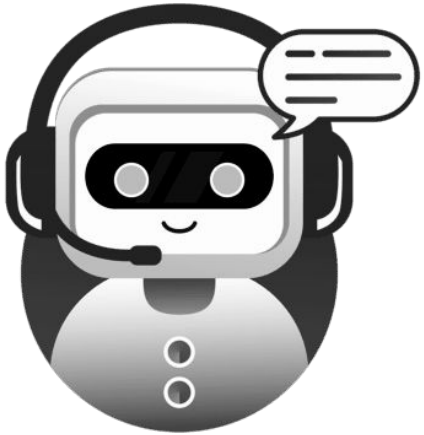
4. Fine-tune

- Epochs: 3-5 (depending on dataset size)
- Batch size: 8-16 (adjust to GPU memory)
- Learning rate: 1e-4 to 5e-5 (typical for LoRA fine-tuning)
- Max sequence length: 512 tokens

Fine-Tuning of an LLM for Employee QA Chatbot

5. Evaluate

- Test on **held-out data** using Exact Match (e.g., 92%) and F1 scores.
- Measure **business impact**: user satisfaction, accuracy on real queries, support ticket reduction, and response time
- **Example:** 92% EM on test set, 4.7/5 user rating, 30% fewer HR questions after launch



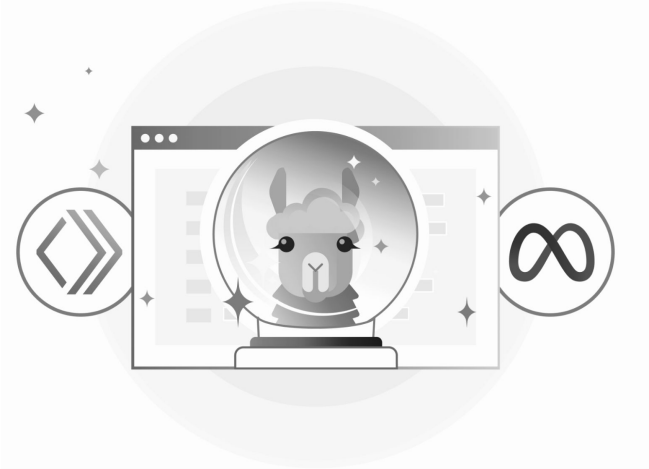
6. Productionize

- Deploy as API; integrate with internal chatbot for HR queries

LoRA, QLoRA

Open Source Model (LLaMA 3.1 8B Example)

- Too large to fully **fine-tune** on a single GPU
- **Architecture:** 32 stacked LLaMA Decoder Layers
- Each layer contains **self-attention,** a **multi-layer perceptron, SiLU** activation**,** and layer **normalization**
- Model **parameters** require around **32 GB of memory**



Low-Rank Adaptation (LoRA)

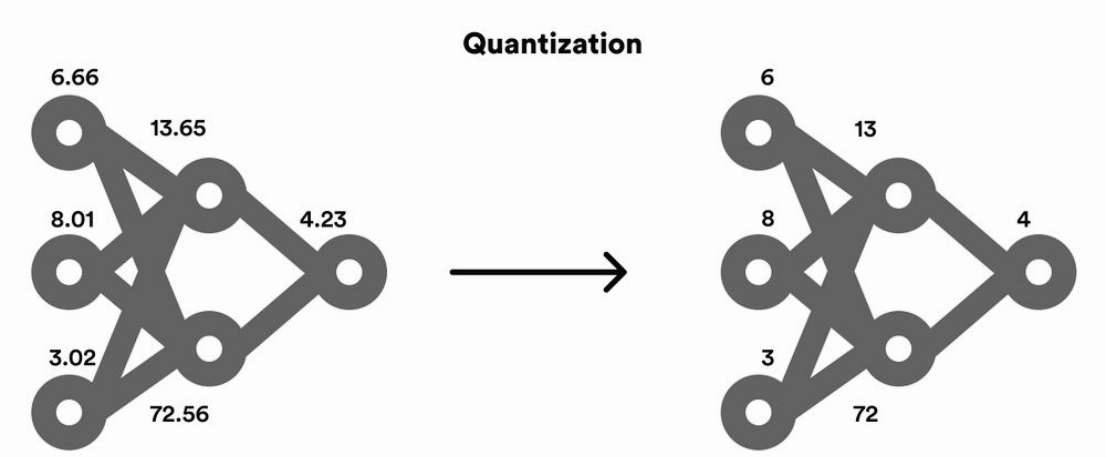
1. **Freeze the weights** - we will not optimize them
2. Select **some layers** to target, called **"Target Modules"**
3. Create new **"adaptor" matrices** with lower dimensions, fewer parameters
4. Apply these adaptors to the Target Modules to **adjust** them - and these get **trained**

Essential Hyperparameters For LoRA Fine-Tuning

Characteristic	r	Alpha	Target Modules
Description	Rank of low-rank matrices	Scaling factor for matrices	Layers adapted in network
Rule of Thumb	Start with 8, double until diminishing returns	Twice the value of r	Target attention head layers

Quantized Low-Rank Adaptation (QLoRA)

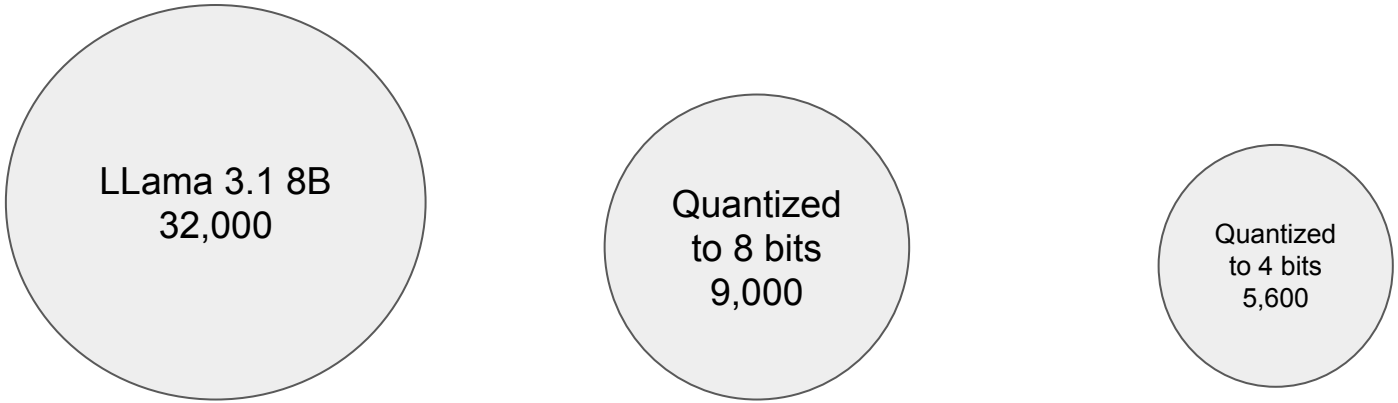
- Quantize to **8-bit** or **4-bit** precision to cut memory use
- Enables **fine-tuning large models** (e.g., LLaMA-3 8B) on a single **GPU**
- Small **accuracy** drop compared to full **precision**
- Significant **savings** in compute and storage



QLoRA Hyperparameters

Parameter	Value	Description
Quantization	4-bit (most common)	Represents efficient memory usage.
Method	NF4 or FP4	Represents different quantization approaches.
LoRA Rank (r)	8, 16, or 32	Represents the LoRA dimension size.
LoRA Alpha	16 or 32	Represents the scaling factor magnitude.
LoRA Dropout	Usually 0.05–0.1	Represents regularization during training.

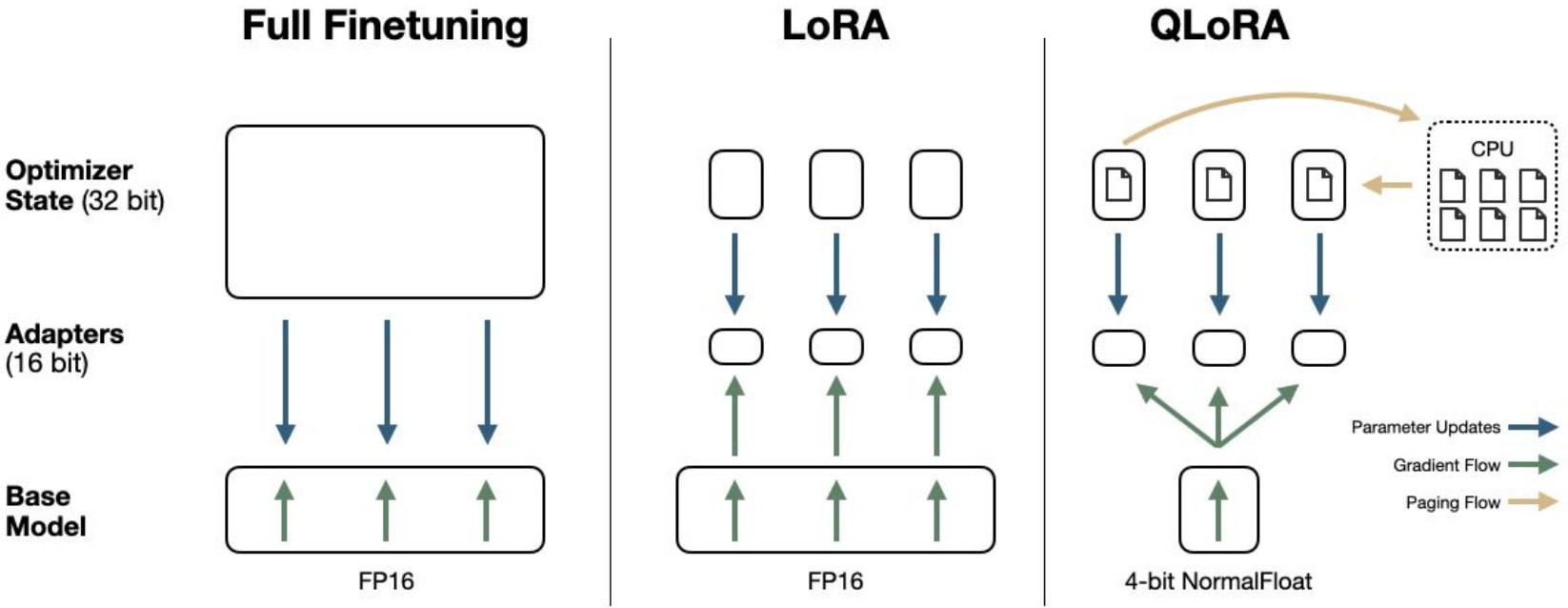
Size of Weights in MB



Full Fine-Tuning vs. LoRA vs.QLoRA

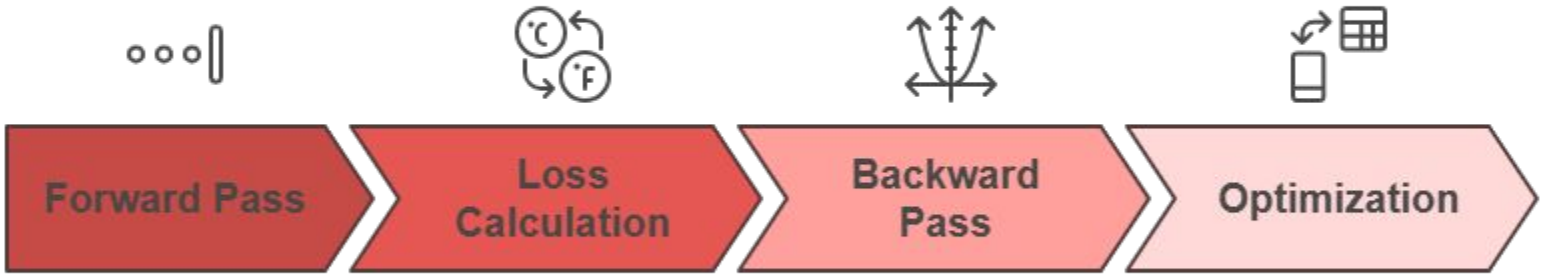
Method	What's Trained	Base Precision	Typical GPU Need	Use Case
Full Fine-Tuning	All model weights	FP16	Very high	Research / custom models
LoRA	Small adapters	FP16	Moderate	Fast task adaptation
QLoRA	Small adapters	4-bit quantized	Low	Fine-tuning large models on a single GPU

Full Fine-Tuning vs. LoRA vs.QLoRA



Training Process

4 Steps of Training

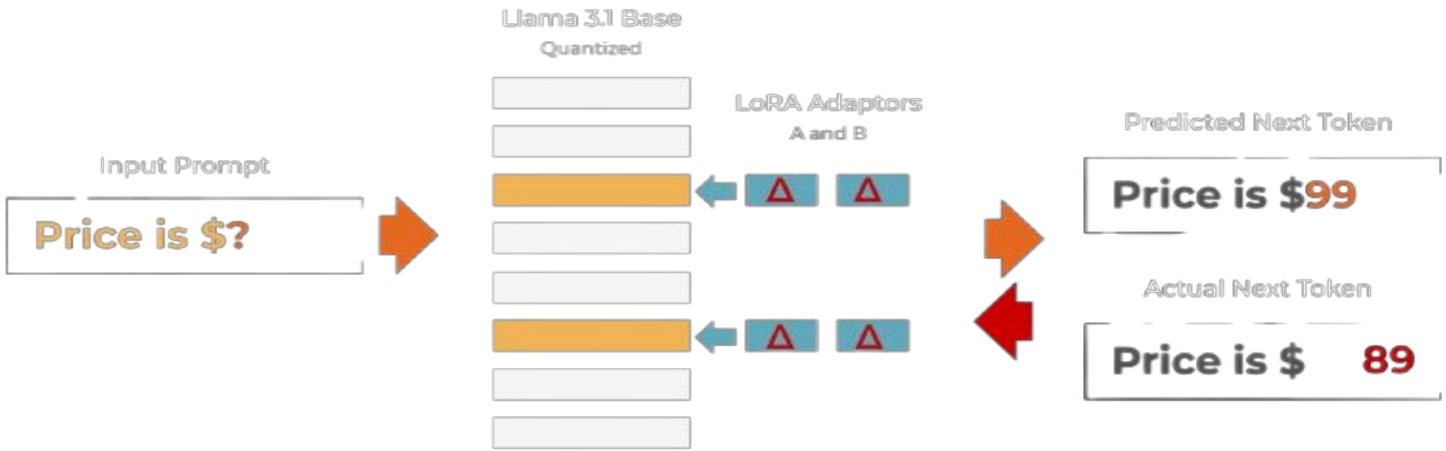


1. Forward Pass

- Feed your domain/task **data** into the pre-trained **LLM**
- Model **predicts** the **next token** using existing knowledge.
- **Example:** Given "Price is $?", predict "99"

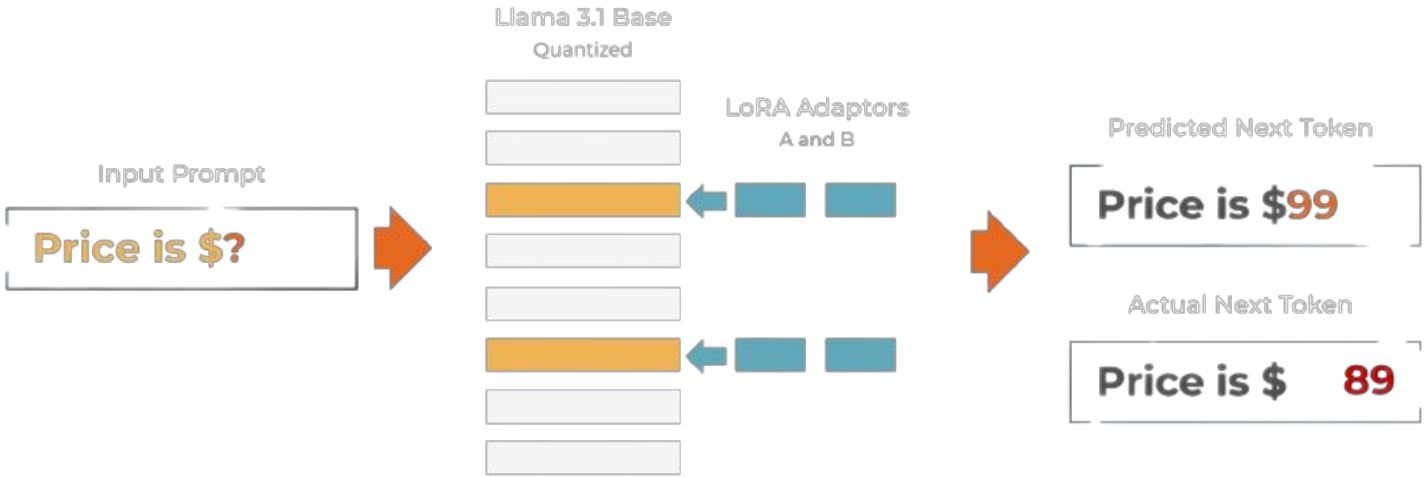
2. Loss Calculation

- Compare **predicted** tokens with **correct** ones from your fine-tuning dataset
- Compute **loss** to measure domain mismatch



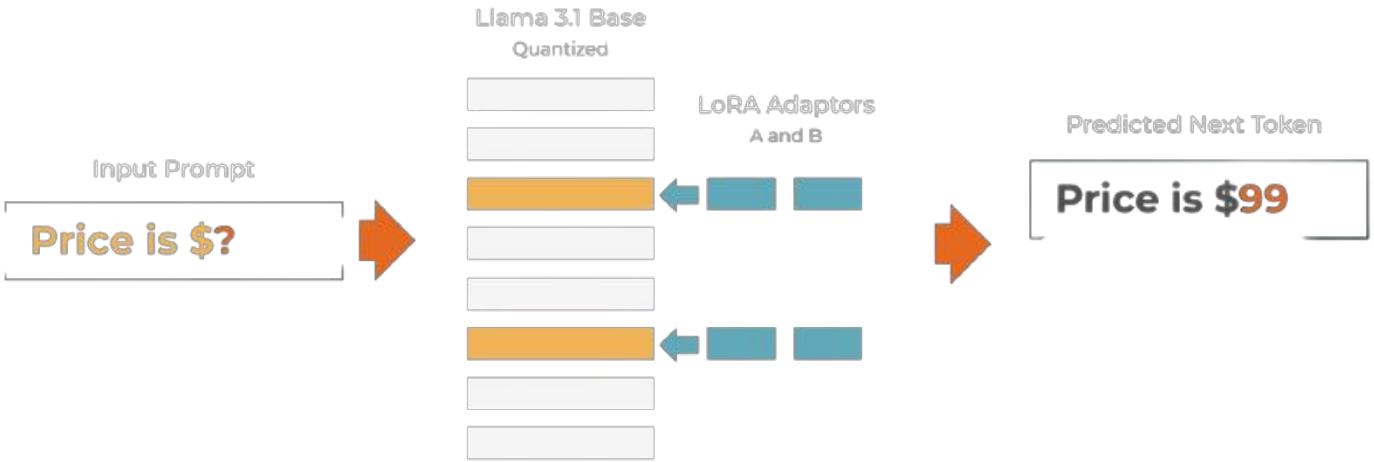
3. Backward Pass

- **Backpropagate** loss to calculate **gradients**
- With **LoRA/QLoRA**, only a small set of **adapter** weights are updated



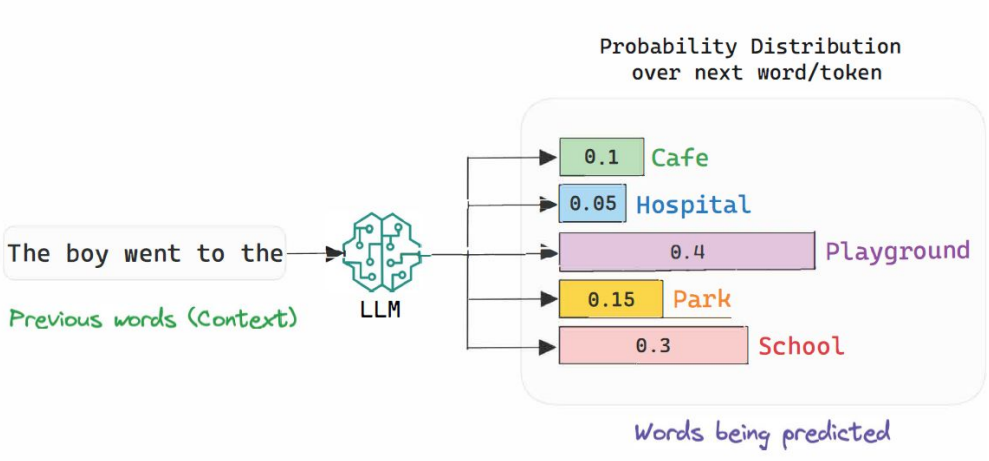
4. Optimization

- Update only **trainable** parameters to align with your new task
- Use **optimizer** (e.g., AdamW) over multiple epochs until **convergence**



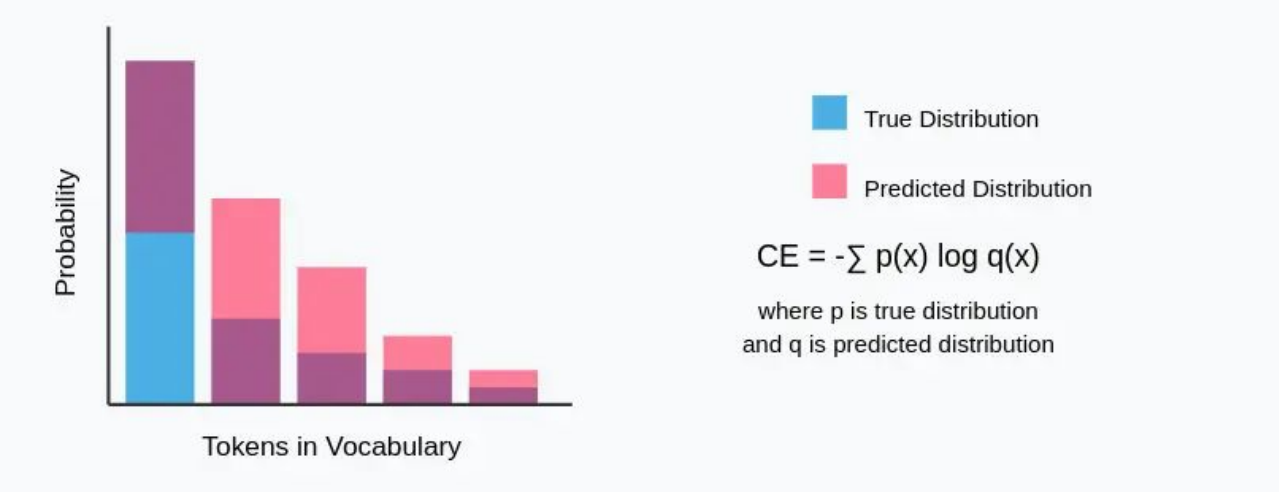
Model Output

- Final layer → softmax → **probability distribution** over all tokens
- Model doesn't give one answer — it gives **probabilities** for every possible next token
- **For Inference:** Choose token with **highest probability** (greedy) or sample from distribution for variety



Loss Function (Cross-Entropy Loss)

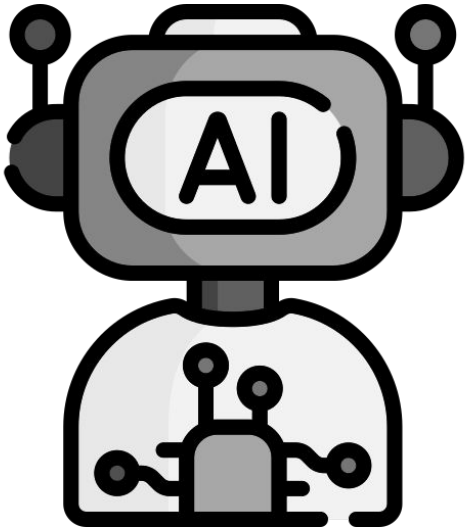
- Check **probability** assigned to the correct token
- **0** means perfect confidence, **higher** means worse
- Guides **weight updates** during training



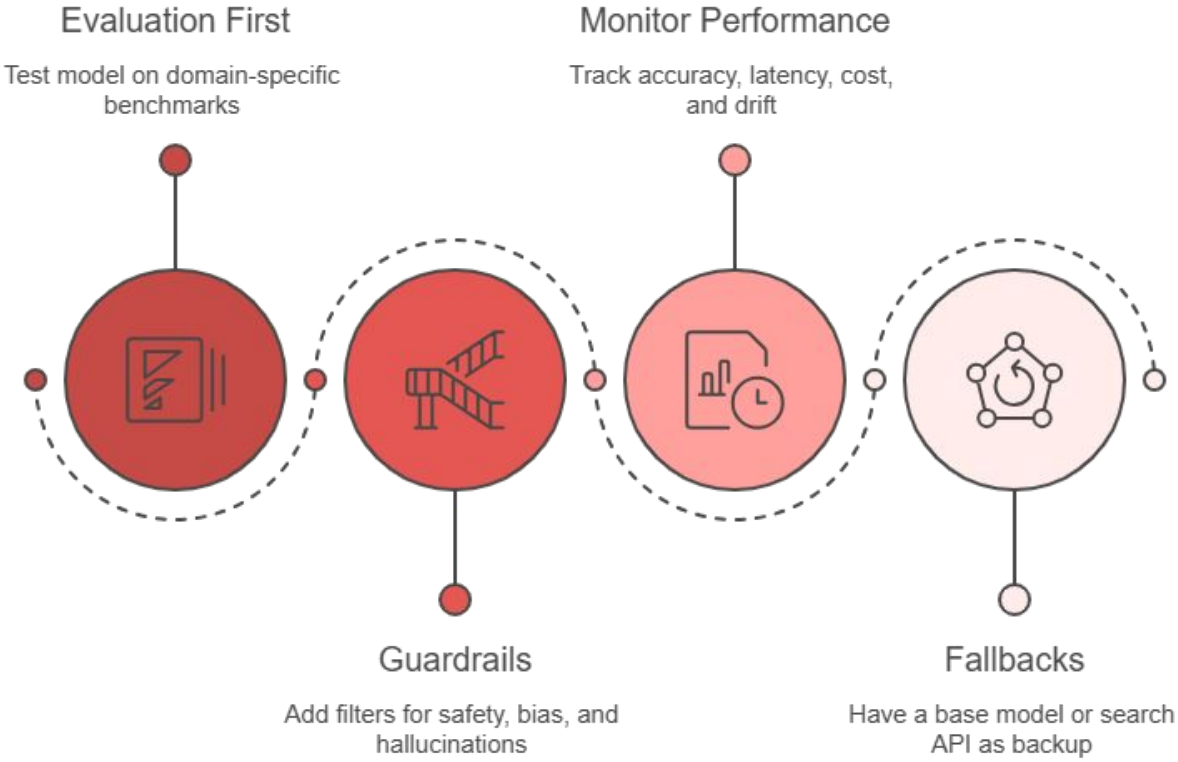
Conclusion

Using Your Fine-Tuned Model in Agents

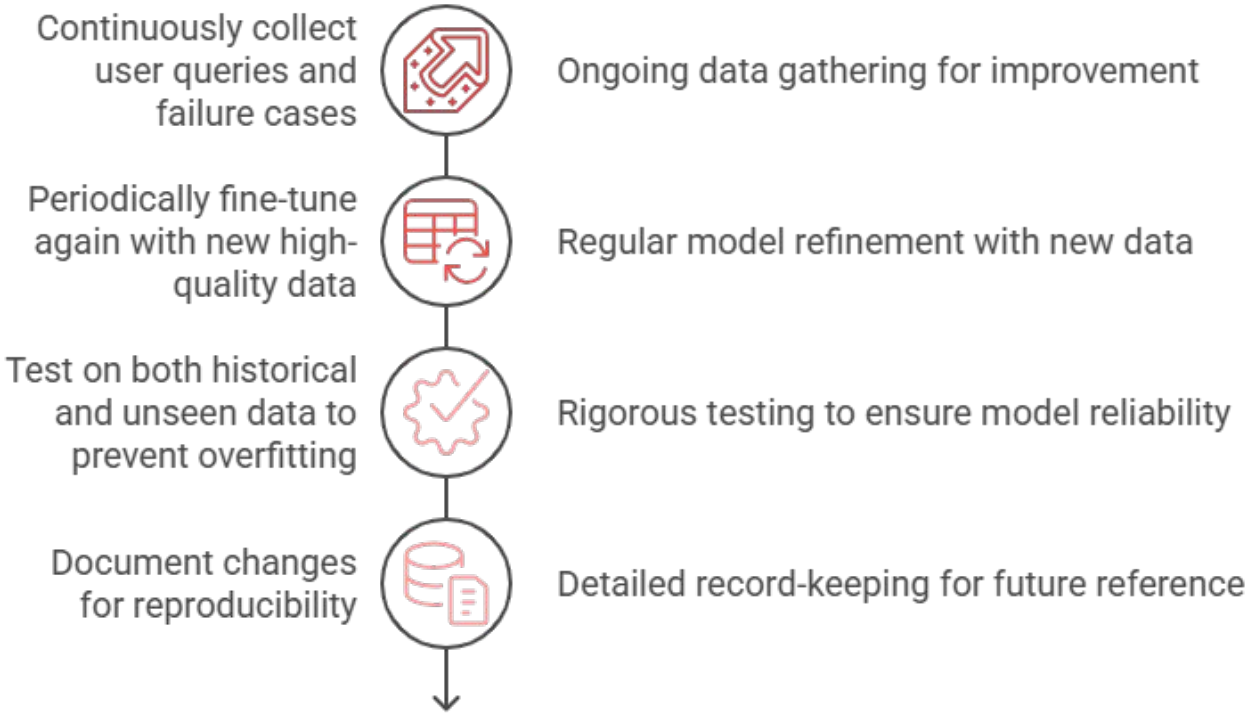
- Wrap the model in an **agent** framework (LangChain, LangGraph, Haystack, Semantic Kernel)
- Provide **tools, memory,** and **reasoning** logic to extend capabilities
- Ensure prompts leverage the **domain knowledge** from fine-tuning
- Combine with **retrieval (RAG)** for up-to-date or large knowledge bases



Best Practices for Deployment



Maintenance & Iteration



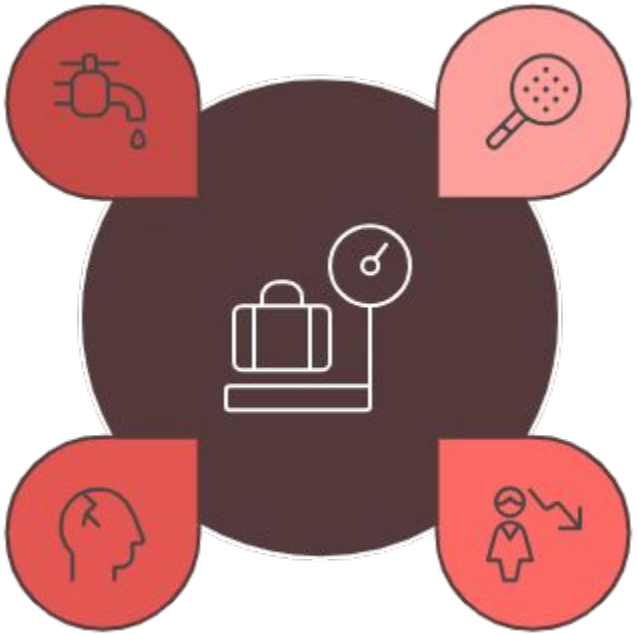
Avoiding LLM Fine-Tuning Pitfalls

Data Leakage

Overlap between training and validation datasets

Catastrophic Forgetting

Loss of initial knowledge during fine-tuning



Overfitting

Training on small datasets or excessive epochs leading to poor generalization

Underfitting

Insufficient training or low learning rates causing inadequate learning

AMERICAN UNIVERSITY OF BEIRUT

Thank you!