DAY 4: LLMOPS & MODEL CUSTOMIZATION

Adding Observability and Fine-Tuning

Ensuring Reliability, Traceability, and Domain Expertise

Course Roadmap: From Foundations to Capstone

01

Understanding the Fundamental

LLM architecture, history, and core prompt engineering.

02

Making Agents Powerful

RAG & Tool usage

03

Working with Frameworks

LangGraph, and building complex chains.

AutoGen, MCP

04

Observability & Fine Tuning

Tracing (LangSmith), evaluation, and custom model tuning.

05

Capstone Project

End-to-end implementation and final presentation.

Day 5 Mini Project Details

- Pick a project
- Create team of 3-4 members
- Follow the guidelines given in document
- Each team will need to present the solution

Day 4 Roadmap

Key Focus Areas

Today is dedicated to the infrastructure required to move an LLM prototype into a robust enterprise application.

- **Agentic RAG Pipeline** Implementing Planner → Retriever → Answerer flow (RAG Github Repositories Overview)
- **Practical Fine-Tuning** Using adapters (PEFT) for domain specificity and comparing outputs
- **Evaluation Strategies:** How to evaluate LLM Applications
- **LLMOps Foundations (Tracing)** Logging prompts, tracking latency/cost, and tracing workflows, PII redaction layers

Langchain Tool usage with Autogen

Task 1

- Multi Tool calling Agent using API

Task 2

- Enhancement in SQL Agent

Agentic RAG

Agentic RAG essentially combines

- **RAG (Retrieval-Augmented Generation)**: The technique of retrieving relevant information from external knowledge sources to augment LLM responses
- **Agentic AI patterns**: Giving the AI autonomous decision-making capabilities to plan, reason, and take actions

Agentic RAG Implementation: Planner-Retriever-Answerer

Challenge: Single-step failures

In complex questions, asking the LLM to search and answer in one step often leads to poor performance. We use a **Planner-Executor** approach

This pipeline separates the three key cognitive tasks, assigning each to a specialized **Node** in our framework.

Planner (LLM Node)
Generates search query from user prompt.

↓

Retriever (Tool Node)
Fetches context from Vector Store based on query.

↓

Answerer (LLM Node)
Generates final answer using retrieved context.

Hands On

Popular Github Repos on RAG

LightRAG

https://lightrag.github.io/

https://github.com/HKUDS/LightRAG

RAGAnything

https://github.com/HKUDS/RAG-Anything

AutoRAG

https://github.com/Marker-Inc-Korea/AutoRAG

Fine Tuning

What is Pre-training?

https://jalammar.github.io/how-gpt3-works-visualizations-animations/

https://huggingface.co/transformers/v3.5.1/pretrained_models.html

Why Fine-Tuning?

From Generalist to Specialist

LLMs like GPT-4o are 'generalists.' Fine-tuning makes them a 'specialist' for your specific domain.

We do not fine-tune to give the model *new facts* (that's RAG). We fine-tune to teach it *how to behave*.

🎓 Key Business Drivers

- **Adhere to Format:** Force output into specific JSON schemas or compliance formats.
- **Master Vernacular:** Teach specific legal, medical, or internal engineering jargon.
- **Steer Tone:** Ensure the AI adopts the precise corporate tone (e.g., "Professional Advisor").

Deployment Paths: API vs. Local Fine-Tuning

Feature	API-Based (e.g., GPT-4o)	Local / Private (e.g., Llama 3)
Data Privacy	Data sent to provider (SLA protected).	Data never leaves your VPC/On-prem (Maximum Privacy).
Hardware Required	None (Managed by Provider).	Requires NVIDIA GPUs (A100/H100 or T4 for inference/PEFT).
Technical Complexity	Low (Upload JSONL → Start Job).	High (VRAM management, Hyperparameter tuning).
Customization	Limited to provider's platform/models.	Full control over every weight, layer, and training loop.

Types of Fine-Tuning

- **Full Fine-Tuning:** Retrains all parameters of the model, which is powerful but computationally expensive.
- **Parameter-Efficient Fine-Tuning (PEFT):** Freezes most of the original model and adds or trains only a small number of new parameters, making it much faster and cheaper.

Fine Tuning with OpenAI

Fine Tuning with OpenAI

Supervised fine-tuning (SFT):

This technique employs traditional supervised learning using "**input-output**" pairs to adjust model parameters. The training process adjusts model weights to minimize the difference between predicted and target outputs across the provided examples. The model will replicate features that it finds in provided pairs.

Sample Dataset:
https://huggingface.co/datasets/knkarthick/samsum

Fine Tuning with OpenAI

DPO:

This technique uses pairwise comparisons (e.g., preferred and rejected example responses) to optimize a model to favor certain outputs over others. The model learns to replicate the preference patterns found in the provided comparison data.

Input: "Explain quantum entanglement like I'm five."
Accept: "Imagine two magic dice. If you roll one and get a six, the other one instantly turns into a six too, no matter how far away it is!"
Reject:"Quantum entanglement is a physical phenomenon that occurs when a group of particles are generated or interact in ways such that the quantum state of each particle cannot be described independently."

Example Question	Chosen Response	Rejected Response
Q1: *How do I review your product?*	To submit a product review, please visit your account dashboard, select the product, and click ‘Write a review.’ Share your honest experience, rate key features, and submit when ready.	Yo, just leave some quick stars or whatever, it’s chill!
Q2: *How do I review your product?*	We welcome your feedback! In the ‘Reviews’ section on the product page, click ‘Leave a Review,’ rate it, and add your comments about what you liked or areas for improvement.	Just scribble something—doesn’t matter what, honestly.
Q3: *How to troubleshoot this particular error?*	To address the error ‘X101,’ first clear your cache, then verify your internet connection. If the issue remains, follow our step-by-step guide at [Support → Troubleshooting → Error X101].	Just reboot it, I guess. If it doesn't work, you're on your own!

https://arxiv.org/pdf/2305.18290

Hands on

Fine Tuning with Open Source Models

The Problem with Traditional Fine-Tuning

Large Language Models (LLMs) like GPT, LLaMA, or Mistral contain **billions of parameters**.

For example:

- A **7B model** has ~7 billion numbers (weights)
- Each weight is usually stored as a floating-point number can be affected

If we try to **fine-tune the entire model**

- We must update **every single weight**
- This requires:
  - Very large GPUs
  - Long training times
  - High cost

“Full fine-tuning is powerful, but impractical for most teams.”

PEFT

What Is PEFT?

PEFT stands for Parameter-Efficient Fine-Tuning. Instead of training all parameters, we:

- Freeze the original model
- Train only a small number of new parameters

Think of it like this: Instead of retraining the whole brain, we attach a small, trainable module.

PEFT allows

- Fine-tune large models on small GPUs
- Train faster
- Reduce cost

What Is LoRA?

LoRA stands for Low-Rank Adaptation. Instead of changing existing model weights:

- LoRA adds small trainable matrices
- These matrices adjust how the model behaves
- The original weights remain frozen

Hands on

LLMOps Foundations (Observability and Tracing)

LLMOps Foundations

Why Trace Production LLMs?

Unlike standard software, LLM errors (like hallucination or incorrect tool choice) are non-deterministic. Tracing provides the essential audit trail.

- **Debugging:** Identify which step (Prompt, Retriever, LLM call) failed or introduced noise.
- **Cost/Latency:** Track token usage and time-per-step for optimization.
- **Compliance:** Log every prompt and response for auditing purposes.

Tracing with LangSmith

LangSmith is a platform specifically designed for LLM workflow tracing. It allows:

- **Waterfall View:** Visualizing the sequence and duration of every node in a chain.
- **Input/Output Logging:** Capturing the raw prompt and response for every LLM call.
- **Dataset Management:** Creating test suites for model evaluation.

Hands on

Prompt Versioning

What Prompt Versioning Is

Prompt versioning = treating prompts as production artifacts, not strings

You should version:

- System prompts
- Planner prompts
- Tool instructions
- Output format constraints
- Safety instructions

Basically:

Anything that influences model behavior

Prompt Drift

What Prompt Drift Is

Prompt drift = model behavior changes over time without code changes

Causes:

- Prompt edits
- Hidden dependencies (retrieved context)
- Tool output changes
- Model version updates (e.g., `gpt-4o` silently updated)

How to Detect Prompt Drift

1. Version every prompt
2. Log prompt versions (LangSmith does this)
3. Run prompt on golden set regularly
4. Compare outputs across versions

Data Drift

What Data Drift Is

Data drift = the data your system sees changes over time

In RAG systems and tools, this is extremely common.

Source	Drift Example
Documents	Policies updated
Vector DB	New embeddings added
APIs	Schema changes

What is Concept Drift?
Example

Concept Drift

What Concept Drift Is

Concept drift = the meaning of the task itself changes

Example: Medical Domain

Before

- "Patient follow-up" = phone calls

Now

- "Patient follow-up" = digital monitoring + alerts

Same words.
Different meaning.

Model appears wrong, but it's actually outdated.

Enterprise Guardrails

1. Prompt Versioning

- Semantic versioning
- Logged in LangSmith
- Stored in repo, not code comments

2. Golden Test Set (LLM Tests)

Maintain:

- 10–50 questions
- Expected output characteristics

Run:

- On prompt change
- On model update
- On data refresh

3. Human-in-the-Loop Reviews

Especially for:

Compliance
Medical

LLM Evaluation Frameworks

Question:

Is LLM Evaluation is easier than traditional ML?

Unlike traditional ML:

- There is no single “right answer”
- Multiple valid outputs exist
- Quality is contextual

3 Pillars of LLM Evaluation

1. Lexical / Statistical
2. Embedding-based (Semantic)
3. LLM-as-a-Judge (Model-based)

Lexical / Statistical

Classic NLP metrics:

- BLEU
- ROUGE
- Exact match
- Keyword overlap
- N-gram similarity

What It Measures Well

- Surface similarity
- Coverage of expected terms
- Faithful copying

What It Fails At

- Paraphrasing
- Reasoning quality
- Tone / style
- Factual grounding

Lexical / Statistical

Use Case	Why
Summarization	Check content coverage
Extraction	Verify fields exist
Compliance	Ensure mandatory phrases
Templates	Format enforcement

Hands - on

Embedding Based

Sentence Similarity (Bi-Encoder)

How it works

1. Encode reference text → vector
2. Encode candidate text → vector
3. Compute cosine similarity

Models Used

- Sentence-BERT (SBERT)
- MiniLM
- OpenAI embeddings
- BGE / Instructor

A hallucinated answer can still be highly similar semantically.

BERTScore (Token-Level Semantic Matching)

BERTScore compares tokens across texts using contextual embeddings.

Instead of:

- Whole sentence → one vector

It does:

- Token → embedding
- Token-to-token matching
- Precision / Recall / F1

Cross Encoder

What a Cross-Encoder Is

A model that reads BOTH texts together and outputs a score.

Unlike bi-encoders:

- Texts are **not embedded separately**
- They are processed **jointly**

LLM Based

LLM-as-a-Judge

Evaluate the answer for:

- Accuracy

- Completeness

- Hallucinations

Score each from 1–5.

What It's Good At

- Reasoning quality
- Faithfulness (especially in RAG)
- Style and tone
- Safety checks

What It's Bad At

Bias (judge agrees with itself)

Pairwise Comparison (A/B Evaluation) & Reference-Based Grading

What It Is

Ask an LLM to choose the better of two outputs.

Example Prompt

```
Which answer better addresses the question?

Answer A: ...

Answer B: ...

Explain briefly.
```

What It Is

Judge compares output against a **gold reference**.

Example Prompt

```
Given the reference answer, score how closely the response matches it.
```

RAG Specific Measures

RAG quality = Retrieval quality × Generation faithfulness.

Recall@K : Did we retrieve the documents that actually contain the answer?
If the correct document is in the top-K results → success
Recall@5 = 0.82, 82% of the time, the answer existed in top-5 chunks

Precision@K: What it Measures“How much of what we retrieved was actually relevant?”

High precision: Less noise and Less hallucination risk

Faithfulness / Groundedness: Are the claims in the answer supported by retrieved context?
LLM-as-judge (answer vs context)
Cross-encoder (answer vs context)

Security Measures

PII Leakage

What it is

- Personally Identifiable Information in inputs/outputs/logs: names, emails, phone, SSN, card numbers, addresses

Mitigations

- **Before LLM + before logging**: PII detectors → redact/tokenize (format-preserving)
- **Minimize data retention;** encrypt at rest; role-based access to logs; structured audit fields

Tooling

- PII detectors (pattern + ML); masking libraries; hashing/tokenization utilities
- Test fixtures with synthetic PII; fail CI if any PII surfaces in captured traces

https://microsoft.github.io/presidio/installation/#using-pip
https://huggingface.co/spaces/presidio/presidio_demo

Prompt Injection / Tool Abuse

What it is

- Inputs trying to override system rules, exfiltrate secrets, or force dangerous tool calls (e.g., shell, network, file IO)

How to measure

- Injection score (heuristics + LLM judge), blocked-attempt rate, tool-call denial rate

Mitigations

- **Input scanner** for injection patterns; **capabilities allow-list** with strict arg schemas

- High-risk tools → confirmation step/HITL; network allow-lists & URL sanitization; sandbox readers

Tooling

- Your injection detector (regex + judge)

- Strict tool schemas, signature checks, and per-tool policies