LLM & GPT Academy

Your Step-by-Step Guide to Language Models,
Prompt Engineering, and Building Custom AI Agents

Level 2

LLM & GPT Academy: Level 2 – Advanced Customization and Deployment

🎯 Target Audience

- Developers, researchers, and advanced hobbyists who've completed Level 1 or have equivalent knowledge.
- Teams building specialized GPTs or deploying LLMs in production.

🎯 Course Goals

- Teach learners to fine-tune and adapt LLMs for domain-specific use cases.
- Explore retrieval-augmented generation (RAG) and multi-modal LLM architectures.
- Cover deployment, scaling, and optimization strategies.
- Emphasize safety, alignment, and explainability in advanced systems.

📖 Syllabus Outline

Unit 1: Advanced LLM Foundations

- **1.1 Transformer Deep Dive**
  - Multi-head attention, positional encoding, and layer stacking
  - Visualizing transformer internals (attention maps, activations)
- **1.2 Large Context Windows & Memory Mechanisms**
  - Long-context models (Claude 3, GPT-4 Turbo, Gemini 1.5)
  - External memory augmentation
- **1.3 Open-Weight LLMs Overview**
  - LLama, Mistral, Falcon, Mixtral, GPT-J, and others
  - Strengths and trade-offs of open vs closed models

📌 *Hands-On*: Explore attention patterns using an interactive visualization tool

Quick Quiz:

1. What are attention heads in transformers?
2. Name one benefit and one drawback of open-weight LLMs.
3. What does a large context window allow an LLM to do?

Unit 2: Fine-Tuning and Adapters

- **2.1 Why Fine-Tune?**
  - When prompt engineering isn't enough
  - Full fine-tuning vs. parameter-efficient tuning
- **2.2 Fine-Tuning Techniques**
  - Full model fine-tuning (pros/cons)
  - LoRA (Low-Rank Adaptation), PEFT (Parameter Efficient Fine Tuning)
  - QLoRA for memory-efficient tuning
- **2.3 Dataset Preparation**
  - Cleaning, tokenization, and formatting for training
- **2.4 Tools and Frameworks**
  - Hugging Face Transformers & PEFT library
  - OpenAI fine-tuning API

📌 *Hands-On*: Fine-tune a small open-weight model on a custom dataset using LoRA

Quick Quiz:

1. When is LoRA preferable over full fine-tuning?
2. Why is dataset cleaning critical before fine-tuning?
3. Name two libraries you can use for fine-tuning LLMs.

Unit 3: Retrieval-Augmented Generation (RAG)

- **3.1 What is RAG?**
  - Combining LLMs with external knowledge bases
- **3.2 Architectures for RAG**
  - Embedding models (OpenAI, Hugging Face sentence-transformers)
  - Vector databases (Pinecone, Weaviate, FAISS)
- **3.3 Implementing a RAG Pipeline**
  - Chunking, embedding, and retrieval strategies
- **3.4 Use Cases**
  - Chatbots with up-to-date knowledge
  - Domain-specific search assistants

📌 *Hands-On*: Build a RAG-powered GPT that can answer questions from custom PDFs

Quick Quiz:

1. What is the main purpose of using a vector database in RAG?
2. Explain the difference between chunking and embedding.
3. List one real-world use case of RAG.

Unit 4: Multi-Modal LLMs

- **4.1 Text + Image Models**
  - CLIP, GPT-4 Vision, Gemini multi-modal
- **4.2 Audio, Video, and Sensor Inputs**
  - Whisper for speech, LLaVA for visual question answering
- **4.3 Designing Multi-Modal Applications**
  - Challenges and design considerations
- **4.4 Tools and APIs**

- OpenAI GPT-4 Vision API, Hugging Face multi-modal pipelines

📌 *Hands-On*: Create a GPT that can describe images and answer follow-up questions

Quick Quiz:

1. What does a multi-modal LLM allow you to do that text-only LLMs cannot?
2. Name one API for building image + text applications.
3. What challenge arises when combining text and audio inputs?

Unit 5: Advanced Deployment and Optimization

- **5.1 Serving LLMs at Scale**
  - Containerization (Docker), API gateways
  - GPU/TPU acceleration and inference optimization
- **5.2 Quantization and Distillation**
  - Reducing model size for edge deployment
- **5.3 Multi-Agent Systems**
  - Architectures for collaborative agents (e.g., Auto-GPT, LangChain agents)
- **5.4 Monitoring and Maintenance**
  - Logging, debugging, and user feedback loops

📌 *Hands-On*: Deploy a lightweight fine-tuned GPT with API access

Quick Quiz:

1. What is quantization in the context of LLM deployment?
2. Why would you use Docker for serving an LLM?
3. Explain one advantage of multi-agent systems.

Unit 6: Safety, Alignment, and Explainability

- **6.1 Alignment Techniques**
  - Reinforcement Learning with Human Feedback (RLHF)
  - Constitutional AI, reward modeling
- **6.2 Explainability Tools**
  - Why did the model output this? Attribution and saliency maps
- **6.3 Adversarial Testing and Jailbreak Prevention**
- **6.4 Real-World Considerations**
  - Legal, privacy, and societal impacts

📌 *Hands-On*: Perform adversarial testing and bias audits on your custom GPT

Quick Quiz:

1. What is RLHF and why is it important?
2. How can saliency maps improve LLM explainability?
3. Give an example of an adversarial prompt.

Unit 7: Capstone Project & Showcase

Build an Advanced GPT Application:

- Example Projects:
  - A domain-specific research assistant with RAG
  - A multi-modal creative assistant (text + images)
  - A fine-tuned GPT for legal/medical summaries
- Deliverables:
  - Project proposal
  - Working prototype
  - Short demo video

🎓 *Showcase your project in the Academy's online community*

📂 Appendices and Extras

- A. Advanced LLM Glossary
- B. Fine-Tuning Cookbook (common pitfalls)
- C. Deployment Templates (Dockerfiles, API configs)
- D. Further Reading: RLHF, LoRA, Multi-Agent Systems

📖 Section 1: Advanced LLM Foundations

1.1 Transformer Deep Dive

Transformers are the backbone of modern LLMs. In this module, we explore their inner workings:

- **Multi-Head Attention**: How the model weighs the importance of words in different contexts.
- **Positional Encoding**: Enabling sequence awareness in transformers.
- **Layer Stacking and Residual Connections**: Building depth while avoiding vanishing gradients.

📊 *Visualization*: Examine attention heatmaps to see how models focus on different parts of input text.

Real-World Analogy: Think of multi-head attention like a group of editors reviewing the same sentence, each focusing on a different aspect (grammar, tone, clarity) before combining feedback.

Quick Quiz:

1. What problem does positional encoding solve in transformers?
2. Why use multiple attention heads instead of one?
3. How do residual connections help deep networks?

📌 *Hands-On Exercise*: Use a pre-built notebook to visualize attention heads on a sample paragraph.

1.2 Large Context Windows & Memory Mechanisms

As LLMs grow, they can "see" more text at once, improving coherence in longer conversations. This module covers:

- **Extended Context Models**: GPT-4 Turbo, Claude 3, Gemini 1.5.
- **Memory Augmentation Techniques**: From sliding windows to external memory stores.

Discussion: How do large context windows change what LLMs can do? Consider use cases like legal document summarization or multi-turn chat history.

Quick Quiz:

1. What is the trade-off of increasing context window size?
2. Name one architecture that extends memory beyond the context window.

📌 *Hands-On Exercise*: Compare responses from a small-context LLM and a large-context LLM on a multi-turn prompt.

1.3 Open-Weight LLMs Overview

Explore the open-source LLM ecosystem and its role in customization.

- **Popular Models**: LLama, Mistral, Falcon, GPT-J, Mixtral.
- **Trade-Offs**: Open weights vs. proprietary models.
- **Community Contributions**: Hugging Face, EleutherAI, and beyond.

Activity: Research an open-weight LLM of your choice and present its strengths, weaknesses, and potential use cases.

Quick Quiz:

1. What's one advantage of open-weight LLMs?
2. Why might a company still choose a closed-source model?

📌 *Hands-On Project*: Fine-tune a small open-weight LLM to classify customer support emails.

✨ *Summary of Section 1:* You've now explored the deep mechanics of LLMs and gained insights into their architecture, memory, and open-source alternatives. These concepts form the foundation for fine-tuning and customizing models in upcoming sections.

📖 Section 2: Fine-Tuning and Adapters

2.1 Why Fine-Tune?

Fine-tuning allows you to customize a pre-trained LLM for a specific domain or task when prompt engineering alone isn't sufficient.

- **Full Model Fine-Tuning**: Adjusting *all* the model's weights to specialize it for your dataset.
- **Parameter-Efficient Fine-Tuning (PEFT)**: Modifying only a small set of parameters while keeping the base model frozen.
- **When to Fine-Tune**:
    - Need for domain expertise (e.g., legal, medical language).
    - Creating a unique tone or behavior for an assistant.

📌 *Case Study*: How LoRA enabled a small startup to create a medical chatbot on a budget.

Quick Quiz:

1. Why might prompt engineering be insufficient for some use cases?
2. What's the main advantage of PEFT over full fine-tuning?

2.2 Fine-Tuning Techniques

- **Full Model Fine-Tuning**
    - Pros: Maximum flexibility.
    - Cons: Requires massive compute resources and large datasets.
- **LoRA (Low-Rank Adaptation)**
    - Inserts small trainable matrices into the model layers.
    - Reduces training cost while achieving impressive results.
- **QLoRA**
    - Combines quantization with LoRA for memory efficiency.

- **Other PEFT Approaches**
    - Prefix Tuning
    - Adapter Layers

📊 *Visualization*: See how LoRA "plugs into" a model during training.

Quick Quiz:

1. What does LoRA add to an LLM during fine-tuning?
2. When would you choose QLoRA over LoRA?

📌 *Hands-On Exercise*: Fine-tune a small open-weight LLM (like DistilGPT-2) using LoRA to create a sentiment classifier.

2.3 Dataset Preparation

"Garbage in, garbage out" applies doubly to fine-tuning. This module covers:

- **Data Cleaning**: Removing noise, duplicates, and irrelevant content.
- **Formatting**: Converting text into the model's expected input-output format.
- **Tokenization**: Breaking text into tokens the model can process.

Activity: Prepare a small dataset for fine-tuning a customer service assistant. Ensure labels and inputs are clean and balanced.

Quick Quiz:

1. Why is tokenization critical before fine-tuning?
2. What are potential risks of using uncleaned data?

2.4 Tools and Frameworks

- **Hugging Face Transformers & PEFT Library**
    - Streamlined APIs for fine-tuning open-weight models.
- **OpenAI Fine-Tuning API**
    - Quickly fine-tune GPT models with minimal setup.

- **Colossal-AI and BitsAndBytes**
    - Advanced tools for efficient training.

📌 *Hands-On Lab*: Use Hugging Face's PEFT library to apply LoRA to a text summarization model.

Quick Quiz:

1. Name two libraries you can use for parameter-efficient fine-tuning.
2. How does Hugging Face simplify fine-tuning workflows?

✨ *Summary of Section 2:* You've now learned how to fine-tune LLMs, from understanding when and why to do it, to selecting techniques like LoRA and preparing datasets effectively. You're ready to customize models for your own tasks.

📖 Section 3: Retrieval-Augmented Generation (RAG)

3.1 What is RAG?

Retrieval-Augmented Generation (RAG) combines the generative power of LLMs with external knowledge bases to produce responses grounded in factual data.

- **Key Idea**: Instead of relying only on pre-trained knowledge, the LLM retrieves relevant documents and uses them to generate answers.
- **Why RAG?**
  - Keeps models up to date without retraining.
  - Reduces hallucinations by grounding answers in retrieved data.

📌 *Example Use Case*: A legal assistant that retrieves case law from a database before answering questions.

Quick Quiz:

1. What problem does RAG solve in LLMs?
2. How does retrieval improve an LLM's performance?

3.2 Architectures for RAG

- **Embedding Models**
  - Convert text into numerical vectors (e.g., OpenAI Embeddings, Hugging Face sentence-transformers).
- **Vector Databases**
  - Store and search embeddings efficiently (e.g., Pinecone, Weaviate, FAISS).
- **RAG Pipelines**
  - Retrieve → Rank → Generate.

📊 *Architecture Diagram*: Visualize the RAG workflow from user query to grounded response.

Quick Quiz:

1. Name two vector databases commonly used in RAG systems.

2. What role do embedding models play in RAG?

3.3 Implementing a RAG Pipeline

Step-by-step:

1. **Document Chunking**: Split large documents into smaller pieces.
2. **Embedding and Storage**: Encode chunks and store in a vector database.
3. **Query Retrieval**: Encode user query, search for relevant chunks.
4. **LLM Generation**: Feed retrieved chunks to the LLM for context-aware answers.

📌 *Hands-On Lab*: Build a RAG pipeline using Hugging Face Transformers and FAISS to create a PDF-based Q&A chatbot.

Quick Quiz:

1. Why is document chunking important in RAG?
2. What happens if you skip the ranking step?

3.4 Use Cases for RAG

- **Chatbots with Up-to-Date Knowledge**: Customer service bots that access internal documentation.
- **Domain-Specific Search Assistants**: Academic, legal, or medical assistants.
- **Enterprise Knowledge Retrieval**: Employees asking natural language questions over company wikis.

Activity: Design a RAG-based assistant for a specific domain (e.g., healthcare, education, finance). Outline its data sources, architecture, and user experience.

Quick Quiz:

1. List one advantage and one limitation of RAG.
2. Give an example of a real-world RAG system.

✨ *Summary of Section 3:* You've learned how RAG enables LLMs to deliver factually grounded answers and explored the architecture and practical implementation of RAG pipelines. You're ready to create knowledge-augmented GPTs.

📖 Section 4: Multi-Modal LLMs

4.1 Text + Image Models

Multi-modal LLMs process and integrate multiple data types, like text and images, to generate richer outputs.

- **Key Models**: CLIP, GPT-4 Vision, Gemini multi-modal.
- **Applications**: Image captioning, visual question answering, and creative tools that blend text and graphics.

📌 *Example*: A travel assistant that analyzes a photo of a landmark and describes its history.

Quick Quiz:

1. What advantage do multi-modal LLMs have over text-only models?
2. Name one multi-modal model that integrates images and text.

4.2 Audio, Video, and Sensor Inputs

Beyond images, some LLMs can handle audio, video, and even sensor data.

- **Audio Processing**: Whisper for speech recognition.
- **Video Understanding**: Early frameworks for analyzing video streams.
- **Sensors and IoT Data**: Potential for robotics and real-world applications.

📌 *Example*: An AI assistant that summarizes a recorded meeting and extracts action items.

Quick Quiz:

1. What is Whisper used for in multi-modal AI?
2. Give one example of a sensor-based multi-modal application.

4.3 Designing Multi-Modal Applications

- **Challenges**
    - Data alignment: Synchronizing inputs from different modalities.

    - Model size and compute: Multi-modal models are often heavier.
- **Best Practices**
    - Start with a single additional modality.
    - Use APIs and pre-trained models for fast prototyping.

📌 *Hands-On Lab*: Create a GPT that takes an image as input and answers user questions about it (using OpenAI GPT-4 Vision API).

Quick Quiz:

1. Why is data alignment critical in multi-modal applications?
2. What's a good first step when prototyping multi-modal systems?

4.4 Tools and APIs for Multi-Modal AI

- **APIs**
    - OpenAI GPT-4 Vision
    - Hugging Face multi-modal pipelines
- **Frameworks**
    - PyTorch + TorchVision for custom pipelines

📌 *Hands-On Project*: Build a simple photo captioning app using CLIP and Hugging Face.

Quick Quiz:

1. Name one API for building multi-modal applications.
2. How can Hugging Face simplify multi-modal development?

✨ *Summary of Section 4:* You've explored multi-modal LLMs, their applications, and the tools needed to create systems that go beyond text. You're now ready to design AI assistants that understand the world in multiple dimensions.

📖 Section 5: Advanced Deployment and Optimization

5.1 Serving LLMs at Scale

Deploying LLMs requires careful planning to handle real-world usage and scaling challenges.

- **Containerization**: Using Docker for portability and reproducibility.
- **API Gateways**: Managing API calls and securing endpoints.
- **Hardware Acceleration**: Leveraging GPUs and TPUs for fast inference.

📌 *Example*: Deploying a customer support GPT on AWS with auto-scaling.

Quick Quiz:

1. Why is containerization important in LLM deployment?
2. What's the role of an API gateway in serving LLMs?

5.2 Quantization and Distillation

Optimize models for deployment on resource-constrained devices.

- **Quantization**: Reduces model size by lowering precision (e.g., 8-bit instead of 32-bit).
- **Knowledge Distillation**: Train a smaller "student" model to mimic a larger "teacher" model.

📌 *Example*: Deploying a quantized GPT to run on a mobile device.

Quick Quiz:

1. What trade-offs come with quantization?
2. How does distillation help in edge deployment?

5.3 Multi-Agent Systems

- **Collaborative Agents**: Combining multiple GPTs for complex workflows.
- **Example Frameworks**: Auto-GPT, LangChain Agents.

- **Use Cases**: Research assistants that divide tasks among specialized agents.

📌 *Hands-On Lab*: Create a multi-agent system where one agent summarizes articles and another drafts emails.

Quick Quiz:

1. What's an advantage of using multi-agent systems?
2. Name a framework that supports multi-agent architectures.

5.4 Monitoring and Maintenance

- **Logging and Debugging**: Track usage and identify errors in production.
- **User Feedback Loops**: Incorporate user corrections to improve outputs.
- **Updating Models**: Strategies for rolling out updates without downtime.

📌 *Activity*: Design a basic monitoring dashboard for your deployed GPT.

Quick Quiz:

1. Why is monitoring critical in production deployments?
2. How can user feedback improve deployed LLMs?

✨ *Summary of Section 5:* You now understand how to deploy, scale, and optimize LLMs for real-world applications while maintaining performance and reliability. You're ready to take your custom GPTs from development to production.

📖 Section 6: Safety, Alignment, and Explainability

6.1 Alignment Techniques

Ensuring that LLMs behave as intended and align with human values.

- **Reinforcement Learning with Human Feedback (RLHF)**: Fine-tuning models based on human preferences.
- **Constitutional AI**: Embedding ethical principles directly into training and inference.
- **Reward Modeling**: Defining and optimizing for desired behaviors.

📌 *Example*: Using RLHF to make a chatbot more polite and helpful.

Quick Quiz:

1. What is the main purpose of RLHF?
2. How does Constitutional AI differ from RLHF?

6.2 Explainability Tools

Making LLMs more transparent by showing how and why they generate outputs.

- **Attribution and Saliency Maps**: Highlight which input tokens influenced the model's response.
- **Chain-of-Thought Prompting**: Encourage the model to "think aloud" step by step.
- **Model Cards and Documentation**: Provide clear descriptions of model behavior and limitations.

📌 *Hands-On Lab*: Use a saliency map tool to analyze why your fine-tuned model generated a specific output.

Quick Quiz:

1. Why is explainability important for sensitive applications?
2. What does a saliency map reveal?

6.3 Adversarial Testing and Jailbreak Prevention

- **Adversarial Prompts**: Craft inputs to test model robustness.
- **Jailbreak Detection**: Prevent users from bypassing safety constraints.
- **Red Teaming**: Simulate attacks to find vulnerabilities.

📌 *Example*: Testing a GPT assistant to ensure it refuses unsafe or biased requests.

Quick Quiz:

1. What is an adversarial prompt?
2. Name one technique to mitigate prompt injection attacks.

6.4 Real-World Considerations

- **Legal and Ethical Implications**: Data privacy, copyright issues, and regulatory compliance.
- **Bias Auditing**: Identifying and mitigating harmful stereotypes in outputs.
- **User Trust and Transparency**: Disclosing AI involvement in applications.

📌 *Activity*: Audit your custom GPT for bias using a checklist of potential issues.

Quick Quiz:

1. Why is bias auditing essential before deploying an LLM?
2. How can transparency improve user trust?

✨ *Summary of Section 6:* You've learned how to make LLMs safer, more aligned with human values, and easier to understand. This knowledge equips you to deploy responsible and trustworthy AI systems.

📖 Section 7: Capstone Project & Showcase

7.1 Build Your Advanced GPT Application

This capstone gives you the opportunity to apply everything learned in Level 2. You will design, build, and present a fully functional custom GPT system.

- **Step 1: Define Your Mission**
  - Choose a domain (e.g., education, healthcare, customer support).
  - Write a mission statement for your GPT (e.g., "A legal assistant GPT that retrieves case law and provides grounded answers.")
- **Step 2: Outline Features and Architecture**
  - Select core features: Fine-tuning, RAG, multi-modal capabilities, etc.
  - Sketch your system architecture.
- **Step 3: Develop and Integrate Components**
  - Prepare datasets for fine-tuning or retrieval.
  - Implement RAG or multi-modal pipelines.
  - Add safety and explainability mechanisms.

📌 *Hands-On Challenge*: Build a prototype GPT that combines at least **two advanced features** from this course (e.g., RAG + Multi-Modal, or Fine-Tuning + Explainability).

7.2 Deliverables

- **Project Proposal**: Brief document outlining your GPT's mission, target users, and architecture.
- **Working Prototype**: A live or locally deployable GPT system.
- **Demo Video**: 3–5 min walkthrough of your GPT in action.
- **Reflection**: Lessons learned and challenges encountered.

7.3 Showcase and Feedback

- **Peer Reviews**: Share your project with fellow learners for constructive feedback.
- **Instructor Evaluation**: Receive expert feedback on technical design, ethical considerations, and usability.
- **Optional Community Submission**: Publish your project in the LLM & GPT Academy online gallery.

📌 *Activity*: Present your GPT live in a virtual showcase session.

✨ *Summary of Section 7:* This capstone consolidates your advanced LLM skills and demonstrates your ability to design, build, and deploy sophisticated AI systems. You now have a portfolio project that highlights your expertise in cutting-edge GPT customization.

📂 Appendices and Extras

A. Advanced LLM Glossary

- **LoRA (Low-Rank Adaptation)**: A parameter-efficient fine-tuning method that inserts trainable matrices into frozen model layers.
- **RAG (Retrieval-Augmented Generation)**: A method combining retrieval of external knowledge with LLM generation.
- **RLHF (Reinforcement Learning with Human Feedback)**: Aligns LLM behavior with human preferences.
- **Quantization**: Reducing model precision for faster and smaller deployments.
- **Saliency Map**: A tool to visualize which inputs influenced an LLM's output.

B. Fine-Tuning Cookbook (Common Pitfalls)

- **Pitfall**: Overfitting on small datasets.
    - **Solution**: Use data augmentation and validation sets.
- **Pitfall**: Forgetting base model knowledge.
    - **Solution**: Apply parameter-efficient techniques like LoRA.
- **Pitfall**: Poor data formatting.
    - **Solution**: Follow best practices for tokenization and input-output structuring.

C. Deployment Templates

- **Dockerfile Template**

```
FROM python:3.10-slim
WORKDIR /app
COPY requirements.txt ./
RUN pip install -r requirements.txt
COPY . ./
CMD ["python", “app.py"]
```

- **API Config Example (FastAPI)**

```
From fastapi import FASTAPI
app = FastAPI()

@app.post("/generate")
def generate_response(prompt: str):
    # Add LLM inference code here
    return {"response": "Hello, world!”}
```

D. Further Reading: RLHF, LoRA, Multi-Agent Systems

- OpenAI Blog: *“Fine-Tuning GPT Models”*
- Hugging Face Docs: *Parameter-Efficient Fine-Tuning (PEFT)*
- Research Papers:
  - *“LoRA: Low-Rank Adaptation of Large Language Models”*
  - *“RAG: Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks”*
- Frameworks:
  - Auto-GPT and LangChain for multi-agent orchestration.

✨ *These appendices provide extra tools, definitions, and resources for advanced learners building production-ready AI systems*.