PESU I/O Course Plan

Course Name:

RAGs to Rich AIs

with LLM Agents

By:

Samarth P and Vyoman Jain

RAGs to Rich AIs – Course Plan

Instructor Name: Samarth P and Vyoman Jain

SRN: PES2UG22CS495 and PES2UG22CS672

Branch: CSE

Semester: V

Course Duration: 30 Hours (20 hours of in-person mentoring + 10 hours of self learning)

Prerequisites for the course:

1. Basic knowledge of Python

Deliverables from the course:

1. Understand what LLMs are and how they can be used for different real world applications
2. Understand what Retrieval Augmented Generation (RAG) is and why it is used
3. Learn about and implement the different types of RAG - naive, advanced, agentic and Graph using LangChain / LlamaIndex / Crew.ai
4. Have a working idea of what happens in a typical LLM pipeline application enhanced with RAG and what production-level codebases look like
5. Recognize the strengths and weaknesses of different types of RAG and where current research is being concentrated

Final Project:

Students should choose a topic of their choice and select corresponding data sources to build and implement an agentic RAG pipeline.

Video Shoot Content:

Week 1: This video introduces a Generative AI, LLMs and explains how to use them for real-world applications and the need for Retrieval Augmented Generation.
Week 2: Overview of RAG architecture and implementation of naive RAG.
Week 3: Advantages of Agentic RAG and implementation using LlamaIndex / LangChain.
Week 4: Overview of emerging RAG architectures - multi-agent RAG, GraphRAG, HybridRAG

Day 0 – Student Onboarding

1. Introduce ourselves and learn about our students
2. Provide a brief introduction to the world of Generative AI covering what LLMs are, an overview of the Transformer architecture and current applications.
3. Task: Explore different popular LLMs out there and setup either Ollama for running open-source models locally (if system is good enough), else set up a free Groq account for making API calls to hosted models

Day 1 – What is RAG and why is it used?

1. Topics to be taught:

- Understanding why LLMs need RAG or other techniques like Fine-tuning, context-caching
- Introduction to Retrieval Augmented Generation
- Overview of the RAG architecture

2. Tasks to be completed:

- Explore the resources (papers, code, blogs, videos) shared by us.

Day 2 – How RAG Works

1. Topics to be taught:

- Explanation of how RAG works
- Introduction to the different components of RAG architecture
- Types of RAG architecture

2. Tasks to be completed:

- Small tutorial and best practices for Python usage in production

3. Weekly Assignment 1:

- Quiz on topics learnt so far and on resources shared by us

Day 3 – Deep Dive into RAG Architecture Part 1: Preprocessing:

1. Topics to be taught:

- Data collection/loading for RAG by scraping / parsing documents
- Chunking of data for indexing
- Indexing of data and introduction to embedding

2. Tasks to be completed:

- Explore LangChain and LlamaIndex frameworks

Day 4 – Deep Dive into RAG Architecture Part 2: Retrieval and Vector DBs:

1. Topics to be taught:

- Understanding what embedding data actually means and demonstration using Jina.AI
- Introduction to Vector databases and how they work
- Understanding a little bit of the Maths behind how retrieval of context in vector DBs works and demonstration using Pinecone

2. Tasks to be completed:

- Implement semantic search for a dataset shared by us

Day 5 – Deep Dive into RAG Architecture Part 3: Generation

1. Topics to be taught:

- Prompt Engineering for RAG
- Context passing to the LLM
- Implementation of a naive RAG application in class to help with the weekly assignment

2. Weekly Assignment 2:

- Implement any simple application using naive RAG

Day 6 – Intro to Advanced RAG Part 1: Pre-Retrieval Techniques

1. Topics to be taught:

- Query Rewriting / Transformation
- Different chunking strategies - fixed-length, recursive and semantic chunking
- Query Routing to different indices

2. Tasks to be completed:

- Implement some of these techniques in your naive RAG application and observe the difference in output quality

Day 7 – Intro to Advanced RAG Part 2: Post-Retrieval Techniques

1. Topics to be taught:

- Reranking of retrieved chunks for optimized context passing to the LLM
- Comparing different reranking models and how to choose one of them
- Context compression and filtering for improved efficiency and performance

2. Tasks to be completed:

- Implement some of these techniques in your naive RAG application and observe the difference in output quality

Day 8 – Intro to Agentic RAG applications:

1. Topics to be taught:

- Limitations of naive RAG and how Agentic applications can benefit us
- Introduction to Agentic RAG architectures
- Demonstration of an agentic application

2. Tasks to be completed:

- **Final Project Discussion** - Topic selection and Team Formation

Day 9 – Hands On: Using LlamaIndex / LangChain to build an agentic application with tool calling

1. Topics to be taught:

- How to use frameworks like LlamaIndex and LangChain

2. Tasks to be completed:

- Using LlamaIndex or LangChain to build an agentic RAG application

3. Weekly Assignment 3:

- Build your own web-scraping agents for up-to-date Question Answering

Day 10 – Hands On: Using Crew.ai to build multi-agent systems + doubt session

1. Topics to be taught:

- Introduction to multi-agent frameworks like Crew.ai and Autogen
- Tutorial on how to build a multi-agent system using Crew.ai

2. Tasks to be completed:

- Implement a Research assistant which helps people with literature reviews using Crew.ai

Day 11- Graph RAG & other emerging architectures

1. Topics to be taught:

- Introduction to GraphRAG - an new and open-source framework from Microsoft
- Brief overview of other emerging architectures and techniques such as HybridRAG, RAFT, LangGraph

2. Tasks to be completed:

- Work on the final project and ask any doubts in class

Day 12 – Project Presentation:

Tasks to be completed:

- Students present their final projects
- All projects are evaluated by us

RAGs to Rich AIs with LLM Agents

Week 1 Video Script

Vyoman: Hello there! Welcome to the beginning of your Gen AI journey and our course, “RAGs to Rich AIs with LLM Agents”. I’m Vyoman …

Samarth: and I’m Samarth! This week we’ll be introduced to what is Generative AI, what LLMs are, why RAG is needed and an overview of RAG architecture.

Vyoman: So, let's get right into it! Generative AI is the hot new field of artificial intelligence that focuses on models which can create new content, whether it's text, images, or even music. Behind the scenes, these amazing capabilities of Gen AI models are powered by Deep Learning models such as neural networks, and I’m sure all of us have had some experience using these through ChatGPT, Copilot, etc.

Samarth: This brings us to LLMs or Large Language Models. This subset of GenAI are sophisticated models trained on vast amounts – millions to billions of GBs of data, allowing them to understand and generate human-like text. They're based on an architecture called Transformers, which we'll dive into during our in-class sessions.

Vyoman: LLMs have shown incredible capabilities in various tasks like writing, translation, and even coding. However, they do have limitations. LLMs are usually limited to their pre-trained knowledge and data for answering user queries which makes it impossible for them to give up-to-date or domain-specific answers. Also, we have all experienced their tendency to generate false or outdated information, often referred to as "hallucinations".

Samarth: This is where Retrieval Augmented Generation, or RAG, comes into play. RAG is a technique that enhances LLMs by providing them with relevant, up-to-date information from external sources. It's like giving the AI a set of reliable reference materials to work with.

Vyoman: Exactly! RAG helps to ground the AI's responses in factual, current information. You can sort of think of it like it makes answering queries from a closed book exam to an open book exam for LLMs. This is crucial for applications where accuracy and reliability are paramount, such as in business, education, or research contexts and has quickly become the industry standard of making LLMs useful.

Samarth: Now, let's take a quick look at the RAG architecture. A typical RAG pipeline consists of the following stages: Data Preparation, Data Indexing, Information Retrieval and LLM Inference.

Vyoman: Let’s break down these stages:

1. Data Preparation: We collect and process the data that will serve as our external knowledge source. This could be anything from documents and websites to databases. We clean this data and break it down into smaller, manageable chunks.

2. Data Indexing: Once we have our cleaned and chunked data, we need to make it searchable. We do this by creating embeddings - these are essentially numerical representations of our text chunks. These embeddings capture the semantic meaning of the text, allowing for more effective retrieval later.

Samarth:

3. Information Retrieval: This is where the magic happens. When a user asks a question, we first convert that question into an embedding. Then, we search our indexed data for chunks that are most similar to this question embedding. This is typically done using vector similarity search in a vector database.
4. LLM Inference: Finally, we take the retrieved information and pass it to our LLM along with the original question. The LLM then uses this context to generate a response that's both relevant to the question and grounded in the retrieved information.

Vyoman: It's worth noting that this pipeline can be further optimized with techniques like query rewriting, reranking, and more, which we'll explore later in the course.

Samarth: We'll also explore advanced RAG techniques and even venture into the exciting world of AI agents. By the end of this course, you'll have the skills to build sophisticated AI systems that can leverage external knowledge to provide accurate and helpful responses.

Vyoman: We're excited to guide you through this journey into the cutting-edge of AI technology. See you in class, where we'll start building our first RAG system!

Samarth: Don't forget to check out the resources we've shared and set up your development environment. Until next time!

RAGs to Rich AIs with LLM Agents

Week 2 Video Script

Samarth: Welcome back, everyone! In our previous session, we introduced the concept of Retrieval Augmented Generation or RAG. Today, we're going to take a deep dive into the RAG architecture and its components.

Vyoman: We'll be covering everything from data preparation to the final generation step. Let's start by quickly revisiting the RAG architecture. As we learned earlier, RAG consists of four main stages: Data Preparation, Data Indexing, Information Retrieval, and LLM Inference.

Samarth: Let's begin with Data Preparation. This stage is crucial because it determines the quality of information our RAG system can work with. The first step here is collecting our data sources.

Vyoman: Collecting data involves extracting text from various file formats like PDFs, web pages, or databases. We can use specialized libraries like BeautifulSoup, Selenium, PyMuPDF or tools like LlamaParse, r.jina.ai to scrape data and handle different file types and ensure we're getting clean, usable text.

Samarth: Once we have our raw text, the next step is chunking. Chunking is the process of breaking down our text into smaller, manageable pieces. This is important because most vector databases and LLMs have input size limitations and smaller more relevant chunks allow for better answers at lower token usage.

Vyoman: There are several chunking strategies we can use. The simplest is fixed-length chunking, where we split the text into chunks of a predetermined size. More advanced methods include semantic chunking, which tries to keep related information together.

Samarth: After chunking, we move on to the Data Indexing stage. This is where we create embeddings for our text chunks and store them in a vector database. Embeddings are essentially a way to represent text as a list of numbers, capturing the semantic meaning of the text. We use large language models specifically trained for creating embeddings, like OpenAI's text-embedding-ada-002.

Vyoman: Once we have our embeddings, we store them in a vector database. But what exactly is a vector database? A vector database is a specialized database designed to store and quickly retrieve these high-dimensional vectors or embeddings. They use sophisticated algorithms to perform fast similarity searches, which is crucial for our Information Retrieval stage.

Samarth: Popular vector databases include Pinecone, Weaviate, and Qdrant. Each has its own strengths, and choosing the right one depends on your specific use case.

Vyoman: Now that we have our data prepared and indexed, let's talk about the Information Retrieval stage. This is where we take a user's query, convert it into an embedding, and use our vector database to find the most relevant chunks of information.

Samarth: The final stage is LLM Inference, where we combine the retrieved information with the user's query to generate a response. This is where prompt engineering comes into play.

Vyoman: Prompt engineering is both an art and a science. It involves crafting the input to the LLM in a way that guides it to produce the most accurate and relevant response. A typical RAG prompt might include the user's question, the retrieved context, and instructions for how to use that context.

Samarth: Let's put it all together with a basic, naïve RAG implementation. When a user asks a question, we first embed that question and use it to retrieve relevant chunks from our vector database. We then construct a prompt that includes the question, the retrieved chunks, and instructions for the LLM. This prompt is then sent to the LLM, which generates a response. And there you have it – a basic RAG system!

Vyoman: Of course, there are many ways to optimize and improve this basic implementation, which we'll explore in our upcoming sessions on advanced RAG techniques. We'll look at strategies like query rewriting, re-ranking, and more. But for now, this gives you a solid foundation in how RAG works under the hood.

Samarth: We hope this deep dive has given you a clearer understanding of the RAG architecture. In our hands-on session this week, we'll be implementing our very own RAG system!

Vyoman: Thanks for tuning in, everyone. Don't forget to check out the additional resources we've provided. See you next time!