Generative AI Mastery Program

"From Zero to Production-Ready GenAI Engineer"

Skills and Tools: #Python #AWS #PromptEngineering #Embeddings #VectorDatabase #GraphDatabase #Cypher #RAG #AgenticAI #LangChain #LangGraph #AgentMemory #ResearchAgents #DeepAgents #MCPServer #A2AProtocol #AWS Bedrock #AWSAgentCore #3RealtimeProjects

Outcomes & Value: #ZeroToHero #BestInMarket #ProudGenAIEngineer #RealWorldProjects #CloudExecution #ProductionReady #MoreKnowledgeLessFee #DoubleYourSalary

Duration: 3.5 Months
with flexibility to extend by 15 days to ensure mastery — we focus on expertise, not just timelines.

Python

Python is the most widely used programming language, especially in Generative AI development. It's an ocean of possibilities — but the good news is, you don't need to know everything to start building powerful AI solutions.

Variables	Basic Building blocks of programing
Functions	Reusable blocks of code that perform specific tasks and improve efficiency.
Object Oriented Programing: • Class • Objects • Inheritance • Abstraction • Encapsulation • Polymorphism	organizes code into reusable "objects" built from classes, making software more modular, scalable, and easier to maintain.

Containers: • List • Tuple • Named Tuple • Sets • Named Tuples • Dicts	Containers in Python are data structures that help store, organize, and manage collections of values efficiently.
Exception handling	Gracefully manage and recover from runtime errors.
File handling	Read, write, and manipulate data stored in files.

AWS

AWS has been the undisputed leader in cloud computing for the last 10+ years. Now booming in Generative AI, it provides the most powerful tools and infrastructure to build, train, and scale AI solutions with ease.

S3	Durable object storage for everything from data lakes to media, at massive scale and low cost
VPC (Virtual Private Cloud)	Your own isolated network slice in AWS to control traffic, security, and connectivity
Subnets, Routing Tables, Security Groups	Essential building blocks of cloud networking: define segments, routes & access rules.
ECS(Fargate)	Run containerised applications server-less, without managing the underlying servers or clusters.
App Load Balancer	Distributes incoming HTTP/S traffic intelligently to your backend containers or instances.
API Gateway	Gateway to expose, secure and scale your APIs effortlessly.
Bedrock	AWS's managed foundation model service: build GenAI apps quickly with pre-trained models and agent tools.
Agentic Core	Agentic infrastructure layer for deploying and scaling AI agents with reliability and security in Bedrock.
Lambda	Serverless computing to run code in response to events with zero server management.
ECR	Manage your containerized workloads with orchestration, cluster management & autoscaling.
RDS – Postgres	Fully managed relational database (PostgreSQL) with automated backups, scaling & patches.
IAM	IAM ensures secure access control — who can do what, when, with AWS resources.

Art of Prompt Engineering(High quality prompts leads to high quality outputs)

Prompt Engineering has become the bridge between human intent and AI output — mastering it lets you shape, guide, and control GenAI with precision.

LLM's Overview

- What is LLM and how does it work?
- Why Prompting engineering?
- key components of LLM
    - Context window
    - Prompt
    - Completion
    - Tokens
        - What are tokens?
        - How text converts to tokens? How are they relevant to prompt engineering?
        - What is BPE and how does it work?
    - Comparing different LLMs like GPT, Grok, Claude, Mistral
    - End to end prompt execution flow

Prompt Engineering

- Principles of prompt engineering
    - **Principle 1**: Write clear and specific instructions
        - **Tactic 1**: Use delimiters to clearly indicate distinct parts of the input
        - **Tactic 2**: Ask for a structured output
        - **Tactic 3:** Ask the model to check whether conditions are satisfied
        - **Tactic 4**: "Few-shot" prompting
    - **Principle 2**: Give the model time to "think"
        - **Tactic 1**: Specify the steps required to complete a task
        - **Tactic 2**: Instruct the model to work out its own solution before rushing to a conclusion
- Iterative prompt engineering
- Summarize Text
- Transformations
    - Classifications of text
    - Translation
- Expanding

Advanced Prompting Techniques:

- COSTAR prompting framework
- Chain-of-thought prompting
- Tree-of-thought prompting

Project 1: Text 2 SQL

Generative AI capability where a model converts natural language input (like English questions) into SQL queries. This lets non-technical users (business analysts, managers, etc.) access and analyse structured data without writing SQL themselves.

Text-to-SQL agents often reach **80-90%+ accuracy** on standard benchmarks like Spider and BIRD, but in real-world deployments their success can drop significantly unless best practices are followed. In this project, we teach you gold-standard prompt-engineering techniques, schema awareness, and external context usage for robust implementation. Using e-commerce data, you'll build an end-to-end system (Frontend + Backend) on AWS — backend built with Fast API, frontend with Streamlit, data stored in RDS, containerized via ECR & ECS, load balanced with ALB, and using OpenAI LLMs for query generation and summarization.

RAG – Retrived Augmented Generation

LLMs have a limited **context window**, i.e. they can only "see" a fixed number of tokens of input + retrieved text at a time. As documents or datasets grow, it becomes impossible to fit *everything* into the prompt. Without RAG, either you truncate (lose useful info) or you pay huge cost/latency to use models with bigger windows. **RAG** solves this by first retrieving only the *most relevant* chunks of information from external knowledge bases or documents, then feeding them into the model *at inference/runtime*. That way, the LLM can answer with up-to-date, domain-specific facts without being retrained or overburdened.

#Extraction Methodologies

A RAG pipeline's success hinges on how efficiently data is extracted — noisy, fragmented or poorly structured input can derail even the best models.

You Learn:

- PyMuPDFLLM
- Llama Parse
- Unstructured

How to efficiently how extract clean text, tables, images, and metadata from documents — accelerating RAG pipelines, reducing errors, and letting you focus on building intelligence rather than wrangling raw data.

#Embedding Models

Embedding models turn words, images, or anything into vectors of meaning — letting GenAI find what's similar, relevant, or useful. With embeddings, your AI system sees relationships: like

how ‘movie’ and ‘film’ are similar, or ‘cat’ and ‘dog’ more than ‘table’ — powering smarter search, RAG, and recommendations.

You Learn:

- SentenceTransformer
- OpenAIEmbeddings
- TitanMultiModalEmbeddings

Convert text to embeddings, Image to embeddings using multimodal and text embeddings models.

#Vector Databases

Vector databases store data as embeddings rather than just text or tables — enabling similarity search that understands meaning, not just keywords. these databases help GenAI systems find relevant context fast, reduce hallucinations, and scale semantic search.

You Learn:

- **Pinecone**
- **S3VectorBuckets**
- **OpenSearch**

#Semantic Search #SimpleRAG #RecomondationSystems #HybridSearch

#Reranking

By inserting a reranker stage, RAG pipelines cut down irrelevant noise, minimize hallucinations, and ensure the final output is more accurate and contextually aligned.

You learn:

- **CohereRerank**

#Graph Database

Graph databases store data as nodes and edges, making relationships first-class—enabling fast, multi-hop queries for recommendations, fraud detection, and more. For use cases where what data is connected matters more than what the data is, graph databases outperform relational systems—offering flexible schema, real-time relationship querying, and superior performance at scale.

You learn:

- **Neo4J**
- **Neptune**

Using ClinicalTrials.gov data, construct a knowledge graph where each trial, condition, intervention, and outcome becomes a node, linked by meaningful relations.

#Langchain

LangChain is an open-source framework for building LLM-powered applications that lets you combine models, prompt engineering, and external data sources seamlessly. It simplifies development of RAG, agents, chatbots, and workflows so you can go from prototype to production with much less overhead.

You Learn:

- **Langchain Library to build RAG using Vector Database, Graph Database**
- **LangSmith (**Observability, debugging, tracing, and eval**)**
- **Lang_eval** (evaluate the output quality)

Chaining LLMs with external data sources & APIs to build modular, production-ready applications. tracing chains, debugging, monitoring performance & logging. evaluation frameworks to measure output quality, set benchmarks, and iterate for better results.

#Gaurdrails

Implementing guardrails ensures your AI behaves responsibly—filtering out bias, reducing hallucinations, securing data, and aligning with values from input to output.

You Learn:

- **AWS Bedrock Guardrails**

How to configure and enforce Guardrails in AWS Bedrock to detect & filter harmful content, block undesirable or restricted topics, and ensure compliance with safety & privacy policies. How to use contextual grounding and Automated Reasoning checks to reduce hallucinations, validate model outputs, and maintain trust in GenAI systems.

Project 2: Clinical Trails Knowledge Base

Transforming ClinicalTrials.gov data into vector + graph databases enables deeper insight into relationships among trials, conditions, interventions, sponsors, and outcomes — making discovery, comparison, and analysis far more powerful than simple tables alone. By doing so, you help researchers ask better questions, accelerate hypothesis generation, improve trial design, and avoid redundant work.

Build a knowledge base using real ClinicalTrials.gov data — with trial PDFs stored in a vector database for semantic search and structured trial metadata modelled as nodes & relationships in a graph database. In this project, you'll build both frontend and backend systems: backend using FastAPI, LangChain, AWS Bedrock + Lambda/API Gateway; containerized with ECR & ECS; storage via S3 and vector buckets; and the frontend for querying and visualizing results.

Agentic AI

Agentic AI takes RAG a step further: it doesn't just fetch context, it uses LLMs to plan, decide, and act with minimal supervision. Whether it's chaining tasks, using tools, or coordinating actions, agentic systems enable real-world automation, adaptivity, and smarter outcomes.

Lang Graph

LangGraph is a stateful orchestration framework for building long-running, multi-actor AI agent workflows — giving you control over state, tool usage, memory, and decision-flows. It bridges the gap between prototype agents and reliable production systems by enabling dynamic control, debugging, and scalable deployment.

You will Learn:

Introduction to Chain, Router, Agent, Agent with memory

State and Memory

- State schema, reducers
- Multi schema
- Trim and Filter messages
- Chatbot w/summarizing messages and memory/ external memory

UX and Human in loop

- Streaming
- Breakpoints
- Editing state and human feedback
- Dynamic break points

- Time travel

Budling your Assistant

- Parallelization
- Sub-graphs
- Map-reduce
- Research Assistant - Build your own deep research agent to handle research tasks.

Long Term memory

- Short vs long-term memory
- Lang-graph store
- Memory Schema + Profile
- Memory schema + collection
- Build and Agent with Long term memory

Deep Agents -- Learn the fundamental characteristics of Deep Agents and how to implement your own Deep Agent for complex, long-running tasks.

#Strands Agents

Strands Agents is an open-source SDK from AWS, built on a *model-first* philosophy for creating AI agents. It simplifies agent development: you define a prompt + tools → Strands handles reasoning, planning, tool execution, state, multi-agent workflows, and deployment.

#MCP(Model Context Protocol) Server:

MCP servers are like universal adapters for AI agents — letting them securely talk to external tools, data, and workflows with a standard protocol, not custom code everywhere. An **MCP server** acts as the bridge: it exposes specific functionality (e.g. database queries, file access, APIs) to the AI agent via MCP, so that agents do *not* need bespoke integrations for every new tool or dataset.

#A2A(Agent 2 Agent Protocol):

Agent-to-Agent (A2A) is an open standard introduced by Google (backed by many partners) to enable AI agents from different vendors or frameworks to communicate securely, share capabilities, and collaborate on tasks

- **Interoperability & collaboration**: Agents can discover each other's "Agent cards" (metadata about what they can do), delegate subtasks, exchange information, and work together—even if built with different underlying tools or infrastructures.
- **Scalable, secure workflows**: With standards for authentication, real-time updates, state tracking, modalities (text, media, etc.), and long-running tasks, A2A allows businesses to build multi-agent systems that scale reliably.

#AWS Agent Core

AWS Agent Core is the foundation for building agentic AI on Bedrock — giving developers tools to create autonomous agents that can plan, reason, and act using AWS and external services.

Project 3: Multi-Agent PubMed Knowledge Orchestration System

In this capstone project, you will design and deploy a **next-gen multi-agent GenAI system** powered by the A2A protocol and MCP servers. The system integrates **search agents, ingestion agents, annotation agents, and analysis agents**, all orchestrated via LangGraph. Agents collaborate autonomously to ingest clinical/scientific documents, normalize entities, enrich a graph database, and answer complex queries with context-aware reasoning..