GenAI Hands On 2

Name: Arnav Sinha
SRN: PES2UG23CS092
Sec: B

1_LangChain_Foundation.ipynb

LangChain as an AI Application Framework
LangChain acts as a bridge between application code and multiple LLM providers. Instead of writing different API logic for each provider, LangChain provides a unified interface. This makes AI systems easier to maintain, scale, and switch across providers with minimal changes.

Tokens and Context Window
We observed that language models process text as tokens rather than full words. The context window determines how many tokens the model can retain during processing. If this limit is exceeded, earlier parts of the conversation may be forgotten. Understanding token limits is critical when designing chatbots and document-processing systems.

Temperature and Output Behaviour
The temperature parameter controls the randomness of model outputs:

- Low temperature → deterministic and consistent responses
- High temperature → creative and diverse responses

This shows how model behaviour can be tuned based on the application domain.

LCEL and Pipeline Construction
LangChain Expression Language (LCEL) allows developers to build readable and modular pipelines instead of nested API calls. We observed that LCEL improves:

- Code clarity
- Debugging capability
- Reusability

Composability and Modularity
LangChain components are modular, allowing easy replacement of models, prompts, or output parsers. This modular architecture supports scalable production systems.

2_Prompt_Engineering.ipynb

LLMs as Probabilistic Systems
We observed that LLMs do not “know” answers but generate text based on probability distributions. Prompt engineering helps guide this probability to produce desired outputs.

Structured Prompt Design
The CO-STAR framework was introduced to improve prompt clarity:

- Context
- Objective
- Style
- Tone
- Audience
- Response Format

Using structured prompts significantly reduces ambiguity and improves output reliability.

Hard Prompting vs Soft Prompting
Hard prompting involves carefully crafting instructions in the prompt, while soft prompting involves adjusting model parameters such as temperature. Combining both approaches allows better control and personalization of LLM behaviour.

Zero-Shot vs Few-Shot Learning
Few-shot prompting provides examples within the prompt and improves consistency, while zero-shot relies entirely on pretrained knowledge. Few-shot prompting showed better reliability in complex tasks.

3_Advanced_Prompting.ipynb

Chain of Thought (CoT)
We observed that prompting the model to reason step-by-step improves accuracy in multi-step tasks such as mathematics and logic. CoT enables the model to generate intermediate reasoning before producing a final answer.

Tree of Thoughts (ToT)
ToT allows the model to explore multiple reasoning paths and evaluate the best solution. This approach is useful for decision-making and optimization problems.

Graph of Thoughts (GoT)
GoT extends reasoning by using graph-like structures where reasoning paths can branch, merge, and backtrack. This method supports complex planning and multi-domain reasoning tasks.

Impact on Reasoning Quality
Advanced prompting techniques significantly improve logical consistency and reduce hallucinations.

4_RAG_and_Vector_Stores.ipynb

Why RAG is Needed
LLMs may hallucinate or provide outdated information. RAG solves this by retrieving relevant external data before generating responses, making outputs more accurate and trustworthy.

Embeddings and Semantic Similarity
Text is converted into numerical vectors called embeddings. Similar concepts produce vectors that are closer in vector space. Cosine similarity is used to measure semantic similarity between vectors.

Vector Database Indexing Techniques
We explored multiple indexing strategies:

- **Flat Index:** Accurate but slow for large datasets
- **IVF:** Clusters vectors for faster retrieval
- **HNSW:** Graph-based indexing for efficient navigation
- **Product Quantization:** Compresses vectors to reduce memory usage

These techniques balance speed, accuracy, and memory efficiency.

RAG Pipeline Understanding
The RAG workflow includes:

1. Data ingestion
2. Embedding generation
3. Vector storage
4. Retrieval
5. Response generation

We observed that the quality of retrieved data directly affects final output quality.