GENAI

NAME: ANIRUDH MURALEEDHARAN

SRN: PES2UG23CS071

1_LangChain_Foundation.ipynb

Core Concepts: Framework & Architecture

LangChain acts as a connection layer between your application and large language models (LLMs). Rather than writing separate API logic for every provider (OpenAI, Gemini, Groq, etc.), LangChain offers a standardized interface to interact with them.

You can think of it as a universal adapter for AI applications.

If you switch from one model provider to another, usually only a small configuration change is needed instead of rewriting your entire system. This makes applications:

- Easier to maintain
- Faster to experiment with
- More reliable when scaling to production

Tokens & Context Window

LLMs do not process text the way humans read sentences. Instead, they work with tokens, which are smaller pieces of text. A token may represent a full word, part of a word, or even punctuation.

For example:

- “Artificial” may be divided into multiple tokens
- “AI” could be a single token

The context window defines the maximum number of tokens a model can handle at one time.

If the conversation exceeds this limit:

- Earlier messages may be dropped
- The model may lose track of context

Understanding token limits is essential when building systems like chatbots, assistants, or document analysis tools.

Temperature: Adjusting Creativity

The temperature setting (ranging from 0.0 to 1.0) determines how creative or predictable the model's output will be.

You can think of temperature as selecting a personality style:

- Low Temperature (0.0) → Structured, consistent, predictable
  "The Accountant" – precise and dependable
- High Temperature (1.0) → Creative, diverse, imaginative
  "The Poet" – expressive and less predictable

Use cases:

- Customer support → Lower temperature
- Brainstorming or storytelling → Higher temperature

Pipeline Construction

LCEL (LangChain Expression Language) allows you to structure workflows in a clean, readable pipeline instead of writing deeply nested function calls.

This improves:

- Code readability
- Debugging efficiency
- Reusability

Production AI systems often involve multiple steps such as:

- Formatting prompts
- Calling the model
- Processing the output
- Validating responses

LCEL keeps these steps organized and easy to manage.

Composability: Modular Architecture

A major strength of LangChain is its modular design.

Because components are independent:

- You can switch models without rewriting prompts
- You can add extra steps like spell-checking

- You can insert logging or validation
- You can change output parsers easily

This modular approach makes AI systems flexible, maintainable, and scalable.

2_Prompt_Engineering.ipynb

Core Concept: Stochasticity & Control

Large Language Models operate probabilistically. They do not “know” answers like humans; instead, they predict the most likely next token based on patterns learned from training data.

Since each token is selected from a probability distribution, outputs are inherently stochastic — meaning there is some randomness involved. Even with the same prompt, responses can vary depending on temperature and sampling parameters.

Prompt engineering is the process of shaping these probabilities through carefully crafted instructions. By adding structure, constraints, and clarity, we guide the model toward generating the desired response.

Instead of modifying the model itself, we optimize the input design to influence the output.

The CO-STAR Framework

To reduce ambiguity and improve control, the CO-STAR framework provides a structured prompt design method:

- Context – Background information required
- Objective – The specific task to complete
- Style – Writing style (formal, casual, technical, etc.)
- Tone – Emotional tone (professional, friendly, persuasive, etc.)
- Audience – Intended readers
- Response Format – Required structure (bullet points, JSON, paragraph, etc.)

Using CO-STAR makes prompts clearer and more deterministic, improving reliability in real-world applications.

3_Advanced_Prompting.ipynb

Core Concept: Chain of Thought (CoT)

Standard LLM responses can struggle with complex multi-step reasoning tasks such as mathematics or logic puzzles. Often, the model predicts an answer immediately without breaking down the reasoning process.

Chain of Thought prompting encourages the model to generate intermediate reasoning steps before providing a final answer. It essentially “thinks aloud,” solving the problem step-by-step.

Benefits:

- Improves reasoning accuracy
- Reduces errors in multi-step problems
- Allows the model to reflect on intermediate steps

Core Concept: Non-Linear Reasoning

Not all reasoning follows a straight line. Advanced prompting techniques allow models to explore multiple solution paths before selecting the best one.

Tree of Thoughts (ToT)

- The model generates multiple possible solution branches
- A judging step evaluates and selects the best path
- Useful for complex problem-solving tasks

Graph of Thoughts (GoT)

- Information is processed in parallel streams
- Example: generating story ideas in multiple genres simultaneously
- Outputs are combined into a final coherent response
- Helpful for creativity and multi-domain reasoning

4_RAG_and_Vector_Stores.ipynb

Core Concept: Retrieval-Augmented Generation (RAG)

LLMs can sometimes generate hallucinations — responses that sound correct but contain incorrect information.

RAG reduces hallucinations by supplying external knowledge during generation, similar to giving the model access to an open-book reference.

Process:

1. Search a knowledge base
2. Retrieve relevant content
3. Insert retrieved information into the prompt
4. Generate a response grounded in factual data

This approach improves accuracy, reliability, and factual consistency.

Embeddings & Semantic Similarity

Text can be converted into embeddings — numerical vector representations that capture meaning.

- Semantically related words produce similar vectors
- Similarity is typically measured using cosine similarity

For example:

- “Cat” is closer to “Dog” than to “Car” in vector space

Embeddings are fundamental for semantic search and retrieval in large datasets.

Indexing Algorithms (FAISS)

Efficient vector search requires optimized indexing strategies. FAISS provides multiple options:

Flat Index

- Performs brute-force search across all vectors
- 100% accurate
- Slower for large datasets

IVF (Inverted File Index)

- Groups vectors into clusters

- Searches only relevant clusters
- Faster than brute-force search

HNSW (Hierarchical Navigable Small World)

- Uses a layered graph structure
- Upper layers act like highways, lower layers like local roads
- Highly efficient for large-scale retrieval

PQ (Product Quantization)

- Compresses vectors to reduce memory usage
- Slightly reduces accuracy
- Ideal for very large datasets