GENAI

NAME - ANKUR SHARMA

SRN - PES2UG23CS077

1_LangChain_Foundation.ipynb

Core Concepts: Understanding the Framework & Architecture

LangChain is best understood as a bridge between your code and large language models (LLMs). Instead of writing custom API logic for every model provider (OpenAI, Gemini, Groq, etc.), LangChain provides a unified interface.

Think of it as a universal adapter for AI systems.

If you ever switch from one model provider to another, you typically only need to modify one line of configuration, not rewrite your entire application. This flexibility makes your system:

- Easier to maintain
- Faster to experiment with
- Safer for production scaling

Tokens & Context Window

Language models do not read text like humans do. They process tokens, which are small chunks of text - often parts of words rather than full words.

For example:

- "Artificial" might be split into multiple tokens.
- "AI" may be one token.

The context window refers to the maximum number of tokens a model can process (or “remember”) at once.

If your conversation exceeds this limit:

- Older messages may be forgotten
- Responses may lose context

Understanding token limits is critical when building real-world AI systems like chatbots or document analyzers.

Temperature: Controlling Creativity

The temperature parameter (0.0-1.0) controls how random or creative the model's output will be.

You can think of it like choosing a personality:

- Low Temperature (0.0) → Predictable, structured, consistent

  "The Accountant" - precise and reliable

- High Temperature (1.0) → Creative, varied, imaginative

  "The Poet" - expressive and less predictable

For:

- Customer support → Use lower temperature
- Brainstorming or storytelling → Use higher temperature

Pipeline Construction

Instead of writing nested function calls, LCEL lets you compose logic in a readable, linear format.

This improves:

- Code clarity
- Debugging
- Reusability

AI systems in production often require multiple steps:

- Prompt formatting
- Model invocation
- Post-processing
- Validation

LCEL makes this structured and manageable.

Composability: Modular Design

One of LangChain's greatest strengths is composability.

Because each component is modular:

- You can swap the model without touching the prompt.
- You can add a spell-checker step.
- You can insert logging or validation.
- You can replace the parser.

This architecture makes your AI application flexible and scalable

2_Prompt_Engineering.ipynb

Core Concept: Stochasticity & Control

Large Language Models (LLMs) are probabilistic systems. They don’t “know” answers in a human sense - instead, they predict the next most likely token based on patterns learned during training. Every word generated is chosen from a probability distribution.

Because of this, outputs are inherently stochastic (random to some degree). Even the same prompt can produce slightly different answers depending on temperature and sampling settings.

Prompt engineering is essentially the art of influencing those probabilities. By carefully structuring instructions, adding constraints, and clarifying expectations, we increase the likelihood that the model generates the response we want.

Instead of changing the model itself, we change the input design to guide the output.

The CO-STAR Framework

To reduce ambiguity and gain more control, the notebook introduces the CO-STAR framework, a structured way to design prompts:

- Context - Background information the model needs
- Objective - What you want the model to accomplish
- Style - The writing style (formal, casual, technical, etc.)
- Tone - Emotional tone (professional, friendly, persuasive, etc.)
- Audience - Who the response is for
- Response Format - The required output structure (bullet points, JSON, paragraph, etc.)

Using CO-STAR makes prompts clearer and more deterministic. It reduces vague outputs and improves reliability, especially in production applications.

3_Advanced_Prompting.ipynb

Core Concept: Chain of Thought (CoT)

Standard LLMs often struggle with tasks requiring multi-step reasoning, like math problems or logic puzzles. Typically, the model predicts the next token immediately, which can lead to errors.

Chain of Thought (CoT) solves this by making the model think step-by-step. Instead of jumping straight to an answer, it generates intermediate reasoning steps - effectively “thinking out loud.”

Key Benefits:

- Allows the model to attend to its own reasoning process
- Reduces mistakes in multi-step tasks

Core Concept: Non-Linear Reasoning

Sometimes reasoning is not strictly linear. Advanced prompting techniques allow models to explore multiple possibilities before deciding.

Tree of Thoughts (ToT)

- The model generates several branches of possible solutions
- A Judge step evaluates which branch is best
- Useful for problem-solving tasks where multiple paths exist

Graph of Thoughts (GoT)

- Information is split and processed in parallel
- For example: generating story plots in different genres (Sci-Fi, Romance, Horror) separately
- Outputs are aggregated into a final coherent result
- Helps with creativity, multi-domain tasks, and large-scale reasoning

4_RAG_and_Vector_Stores.ipynb

Core Concept: Retrieval-Augmented Generation (RAG)

LLMs sometimes produce hallucinations, generating text that sounds plausible but is factually incorrect. RAG mitigates this by providing external knowledge during generation - like an “open-book test” for the model.

How It Works:

1. Query a knowledge base for relevant information
2. Retrieve the most relevant passages
3. Inject these facts into the model’s prompt
4. Generate a response grounded in real data

This ensures outputs are **accurate, verifiable, and contextually informed**.

Embeddings & Semantic Similarity

Text data is converted into vectors - numerical representations capturing meaning.

- Related concepts produce similar vectors
- Semantic similarity is measured using Cosine Similarity (the angle between vectors)

Example:

- `"Cat"` is closer to `"Dog"` than to `"Car"` in vector space

This vector-based representation is essential for searching and retrieving relevant information in large datasets.

Core Concept: Indexing Algorithms (FAISS)

Vector search needs to be fast and scalable. FAISS provides several indexing strategies:

Flat Index

- Checks every vector (brute-force)
- 100% accurate, but slow for large datasets

IVF (Inverted File Index)

- Clusters vectors into groups (like sorting mail by zip code)
- Searches only relevant clusters, speeding up retrieval

HNSW (Hierarchical Navigable Small World)

- Graph-based index with layers
- Fast navigation: top layers = “express highways,” bottom layers = “local roads”
- Extremely efficient for large-scale retrieval

PQ (Product Quantization)

- Compresses vectors to save memory
- Trades a small amount of accuracy for efficiency
- Useful when working with massive datasets