GENAI HANDSON -2

NAME : ANJALI GUNTI	SRN : PES2UG23CS075	SECTION : B

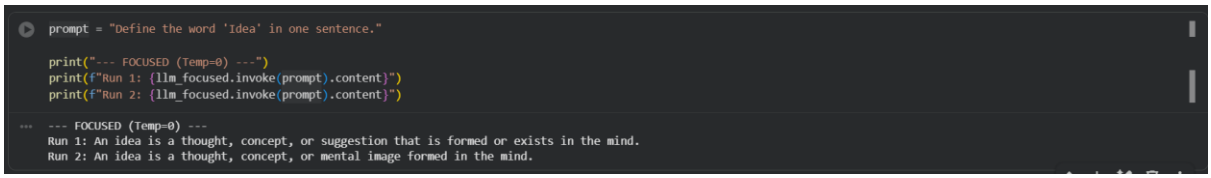
1. LangChain_Foundation

1. The LangChain Philosophy

- **Abstraction:** LangChain acts as a universal adapter, allowing developers to switch models (e.g., OpenAI to Gemini) by changing just one line of code.
- **Security:** Avoid hardcoding API keys; use getpass or environment variables to keep credentials secure.

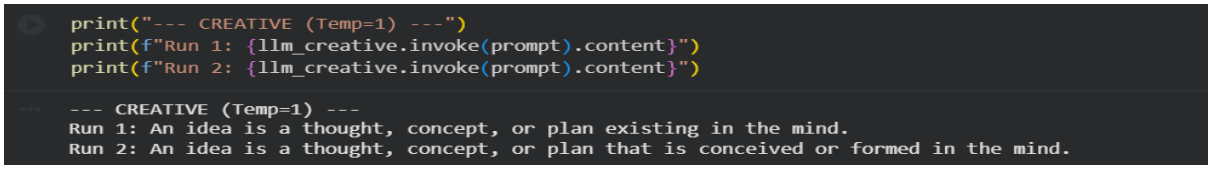
2. The Technical "Brain" (Tokens & Temperature)

- **Tokens:** AI processes text in chunks called tokens; these are the units for billing and determine the **Context Window** (the model's memory limit).
- **Temperature Control:** This setting adjusts the randomness of responses:
    - **Temp 0 (The Accountant):** Produces deterministic, identical results—ideal for facts and math.
    - **Temp 1 (The Poet):** Responses maintain a similar structure and meaning but differ slightly in wording.
    - **Temp 2 (High Diversity):** Demonstrates maximum creativity with significant variation in sentence construction and detail.

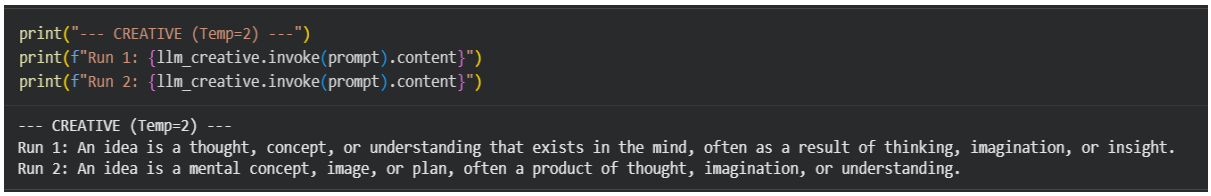


When temp is 0 and 1 respectively

temp 1



Temp 2:



3. The LCEL Pipeline (LangChain Expression Language)

The pipe operator (|) is the "secret sauce" that makes AI logic **composable**.

- **The Chain:** chain = template | llm | parser.
- **Modular Logic:** You can easily swap components or add steps (like a "cynical critic" persona) without rewriting the entire application.
- **Streaming:** Supports word-by-word output, providing a smoother user experience.

4. Key Experimental Observations

- **Predictability:** At lower temperatures, the model prioritizes consistency.
- **Creative Range:** As temperature increases toward 2.0, the model moves from simple synonym swapping to entirely different narrative structures.

2. Prompt_Engineering

1. The LangChain Philosophy

- **Abstraction:** LangChain acts as a universal adapter, allowing developers to switch models (e.g., OpenAI to Gemini) by changing just one line of code.
- **Security:** Avoid hardcoding API keys; use getpass or environment variables to keep credentials secure.

2. The Technical "Brain" (Tokens & Temperature)

- **Tokens:** AI processes text in chunks called tokens; these are the units for billing and determine the **Context Window** (the model's memory limit).
- **Temperature Control:** This setting adjusts the randomness of responses:
    - **Temp 0 (The Accountant):** Produces deterministic, identical results—ideal for facts and math.
    - **Temp 1 (The Poet):** Responses maintain a similar structure and meaning but differ slightly in wording.
    - **Temp 2 (High Diversity):** Demonstrates maximum creativity with significant variation in sentence construction and detail.

3. The LCEL Pipeline (LangChain Expression Language)

The pipe operator (|) is the "secret sauce" that makes AI logic **composable**.

- **The Chain:** chain = template | llm | parser.
- **Modular Logic:** You can easily swap components or add steps (like a "cynical critic" persona) without rewriting the entire application.
- **Streaming:** Supports word-by-word output, providing a smoother user experience.

4. Key Experimental Observations

- **Predictability:** At lower temperatures, the model prioritizes consistency.
- **Creative Range:** As temperature increases toward 2.0, the model moves from simple synonym swapping to entirely different narrative structures.

3. Advanced_Prompting

1. Chain of Thought (CoT): The Inner Monologue

- **Concept:** Standard LLMs often fail at complex math or logic because they try to jump straight to the answer.
- **Mechanism:** CoT forces the model to "think out loud" by generating intermediate steps before the final response.
- **Latent Reasoning:** Because LLMs are "Next Token Predictors," generating intermediate tokens allows the model to **attend** to its own previous "thoughts" to compute a more accurate final answer.
- **Magic Phrase:** Adding "Let's think step by step" can effectively "debug" a model's logic.

2. Tree of Thoughts (ToT): Branching Logic

- **Concept:** CoT is linear, but complex reasoning is often non-linear. ToT is like a chess player considering multiple possible moves before making a decision.
- **Implementation:**
    1. **Branch Generator:** Generate multiple unique solutions to a problem in parallel.
    2. **The Judge:** Use a specific persona (e.g., a Child Psychologist) to evaluate the branches and pick the most sustainable or effective one.

3. Graph of Thoughts (GoT): Aggregation & Mixing

- **Concept:** Information is treated as a network where ideas can split, process independently, and then **aggregate** (mix) back together.
- **Workflow:**
    1. **Split:** Generate independent drafts (e.g., different movie genres).
    2. **Aggregate:** A "Mixer" model reads all versions and creates a "Master" version combining the best elements of each.

4. Strategy & Comparison Table

The notebook provides a recommendation to always **start with CoT** and only move to ToT or GoT if simpler methods fail due to the increased cost and latency.

Method	Structure	Best For...	Cost/Latency
Simple Prompt	Input -> Output	Simple facts, summaries	⭐ Low
CoT (Chain)	Input -> Steps -> Output	Math, Logic, Debugging	⭐ ⭐ Med
ToT (Tree)	Input -> 3x Branches -> Select -> Output	Strategic decisions, Brainstorming	⭐ ⭐ ⭐ High
GoT (Graph)	Input -> Branch -> Mix/Aggregate -> Output	Creative Writing, Research Synthesis	⭐ ⭐ ⭐ ⭐ V. High

5. Key Practical Learning

- **Model Selection:** For testing prompt engineering, use smaller models (like Llama 3.1-8b). Larger models (Gemini Pro/GPT-4) are often "too smart" and solve riddles instantly, making it harder to see the actual impact of your prompt structure.
- **LCEL Tools:** Advanced prompting relies on LangChain tools like RunnableParallel for branching and RunnableLambda for custom logic during the chain.

4. RAG_and_Vector_Stores

1. The "Why" of RAG

- **Knowledge Cutoffs:** LLMs are frozen in time based on their training data.
- **Hallucinations:** Without external data, models often confidently state incorrect facts.
- **The RAG Fix:** By retrieving relevant data and injecting it into the prompt, you provide the model with a "source of truth" to reference.

2. Embeddings: Giving Meaning to Math

- **Vector Representation:** An embedding translates human words into lists of numbers (vectors).
- **Semantic Proximity:** In a vector space, words with similar meanings (e.g., "Apple" and "Banana") are placed mathematically close to each other.
- **Dimensions:** High-quality models use hundreds of dimensions (e.g., 384 for all-MiniLM-L6-v2) to capture complex features like "is it a fruit?" or "is it tech-related?".

3. The RAG Pipeline

The standard "Naïve RAG" workflow follows a structured path:

1. **Ingestion:** Split documents into chunks and convert them into embeddings.
2. **Storage:** Store these vectors in a database like **FAISS**.
3. **Retrieval:** When a user asks a question, convert that question into a vector and find the "closest" matches in the database.
4. **Generation:** Feed the question + the retrieved chunks to the LLM to get an answer grounded in the data.

4. Vector Store Indexing Methods

Index Type	Search Method	Pros	Cons
Flat	Brute force (checks every vector)	100% Accurate	Very slow on large datasets
IVF	Clusters/Partitions	Much faster than Flat	Can miss the exact match (Approximate)
HNSW	Graph-based (Linked nodes)	Extremely fast; industry standard	High memory usage
PQ	Compression	Massive memory savings	Loss of some accuracy

Efficiently searching millions of vectors requires specialized "Indexing" strategies.

5. Key Practical Learnings

- **Distance Metrics:** The system uses math (like **Cosine Similarity**) to determine how "close" two ideas are in vector space.
- **FAISS:** This library allows you to build these indexes locally or in the cloud for lightning-fast retrieval during AI conversations.

This notebook demonstrates the implementation of a **Mixture of Experts (MoE) Router**, a sophisticated architecture designed to handle diverse customer support queries by directing them to specialized "expert" personas.

5. The MoE Architecture

1. The system is built on a three-tier logical structure:

- **The Router**: An LLM acting as a classifier. It uses temperature=0 to ensure **deterministic and consistent** routing of user queries into specific categories.
- **The Orchestrator**: The central logic layer (the process_request function) that manages the workflow, calls the router, selects the appropriate expert configuration, and handles any necessary tool calls.
- **The Experts**: Specialized personas (Technical, Billing, General, and Tool-Use) defined by unique **system prompts** and temperature settings, all running on the same base model (e.g., llama-3.3-70b-versatile).

2. Expert Configurations (MODEL_CONFIG)

Each expert is defined by a specific personality and behavioral guideline:

Expert	Persona / Role	Key Characteristics
Technical	Support Engineer	Precise, code-focused, identifies root causes.
Billing	Professional Specialist	Compassionate, polite, follows refund/payment policies.
General	Helpful Assistant	Warm tone, conversational, handles greetings.
Tool-Use	Data Assistant	Fetches and explains real-time data clearly.

3. Key Technical Implementations

- **Deterministic Routing**: The router is strictly instructed to return **only** a single category word without punctuation to ensure the orchestrator can use it as a dictionary key.
- **Mock Tool Integration**: The system includes a mock_fetch_price function that simulates an external API call for cryptocurrency prices, demonstrating how RAG-like capabilities can be integrated into a router.
- **Dynamic Response Generation**: For tool-based queries, the expert doesn't just show raw data; it receives the tool's output and uses it to craft a natural, human-friendly response.

4. Summary of Work Flow

1. **Route**: User query (e.g., "Bitcoin price") $\rightarrow$ Router $\rightarrow$ "tool_use".
2. **Act**: Orchestrator sees "tool_use" $\rightarrow$ Calls mock_fetch_price().
3. **Generate**: Tool Data + System Prompt $\rightarrow$ LLM $\rightarrow$ Final conversational answer.