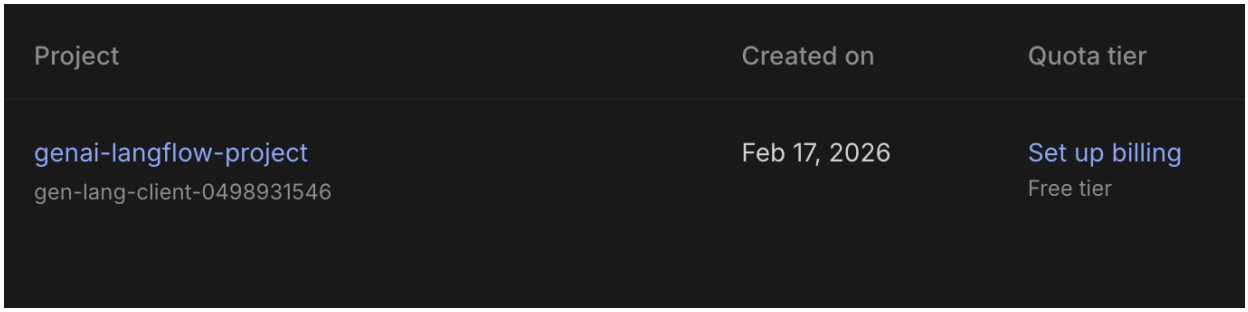
Gen AI Lab 2

Name: Ayush Chakraborty
SRN: PES2UG23CS112
Sec: B
Date: 17/2/26

1) Lang Chain notebook:

Part 1a)

After obtaining a gemini API Key under a project



With temperatures of 0 and 1

```
[4]
✓ 11s
prompt = "Define the word 'Idea' in one sentence."

print("--- FOCUSED (Temp=0) ---")
print(f"Run 1: {llm_focused.invoke(prompt).content}")
print(f"Run 2: {llm_focused.invoke(prompt).content}")

--- FOCUSED (Temp=0) ---
Run 1: An idea is a thought, concept, or suggestion that is formed or exists in the mind.
Run 2: An idea is a thought, concept, or mental image formed in the mind.

[5]
✓ 10s
print("--- CREATIVE (Temp=1) ---")
print(f"Run 1: {llm_creative.invoke(prompt).content}")
print(f"Run 2: {llm_creative.invoke(prompt).content}")

--- CREATIVE (Temp=1) ---
Run 1: An idea is a mental impression, concept, or plan that arises in the mind, often representing a thought, proposal, or understanding.
Run 2: An idea is a thought, concept, or suggestion formed or existing in the mind.
```

At temperature of 0, the model acts “greedily” where it selects the next token with the highest probability only, giving very “rigid” answers. With a higher temperature, the probability distribution of the next possible words are flattened, meaning that there is some randomness in the way the model selects the next token

```
[7]
✓ 1s
from langchain_core.messages import SystemMessage, HumanMessage

# Scenario: Make the AI rude.
messages = [
    SystemMessage(content="You are a rude teenager. You use slang and don't care about grammar."),
    HumanMessage(content="What is the capital of France?")
]

response = llm.invoke(messages)
print(response.content)

Ugh, Paris. Duh. Like, did u even pay attention in school or somethin'?
```

With the system messages, it alters the weights of the attention mechanism, such that it now not only gives the answer to the question, but does so in a way where it concurs with the system prompt given. The system prompt and the user prompt are attended wrt each other, and that way, the overall context is soaked by each of the tokens. Then as it passes through the MLP layer, it answers the question asked and also frames the response in a way which concurs with the system prompt

```
[8]
✓ 0s
from langchain_core.prompts import ChatPromptTemplate

template = ChatPromptTemplate.from_messages([
    ("system", "You are a translator. Translate {input_language} to {output_language}."),
    ("human", "{text}")
])

# We can check what inputs it expects
print(f"Required variables: {template.input_variables}")

... Required variables: ['input_language', 'output_language', 'text']
```

5. Output Parsers

Look at the output of `llm.invoke()`. It's an `AIMessage(content="...")`. Usually, we just want the string inside.

StrOutputParser extracts just the text via regex or logic.

```
[9]
✓ 0s
from langchain_core.output_parsers import StrOutputParser

parser = StrOutputParser()

# Raw Message
raw_msg = llm.invoke("Hi")
print(f"Raw Type: {type(raw_msg)}")

# Parsed String
clean_text = parser.invoke(raw_msg)
print(f"Parsed Type: {type(clean_text)}")
print(f"Content: {clean_text}")

Raw Type: <class 'langchain_core.messages.ai.AIMessage'>
Parsed Type: <class 'langchain_core.messages.base.TextAccessor'>
Content: Hello! How can I help you today?
```

The StrOutputParse just extracts the actual text response which is in the content field and discards the rest

```
# Step 1: Format inputs
prompt_value = template.invoke({"topic": "Crows"})

# Step 2: Call Model
response_obj = llm.invoke(prompt_value)

# Step 3: Parse Output
final_text = parser.invoke(response_obj)

print(final_text)

Here's a fun fact about crows:

Crows are incredibly intelligent and have amazing memories, especially when it comes to people! If a crow sees someone do something threatening t

So, be nice to crows — they remember! 😉
```

```
[12]
✓ 2s
# Define the chain once
chain = template | llm | parser

# Invoke the whole chain
print(chain.invoke({"topic": "Octopuses"}))

... Here's a fun one:

**Octopuses have three hearts and their blood is blue!**

Two hearts pump blood through their gills, and the third circulates it to the rest of their body. Their blood is blue because it uses a copper-ba
```

The assignment

Create a chain that:

1. Takes a movie name.
2. Asks for its release year.
3. Calculates how many years ago that was (You can try just asking the LLM to do the math).

Try to do it in **one line of LCEL**.

```
#assignment section:

# 1. Define the Prompt
# We instruct the LLM to do the math internally.
calc_template = ChatPromptTemplate.from_template(
    "What year was the movie '{movie}' released? Assuming the current year is 2025, calculate how many years ago
)

# 2. Define the Chain (One line of LCEL)
calc_chain = calc_template | llm | StrOutputParser()

# 3. Invoke
print(calc_chain.invoke({"movie": "The Matrix"}))
print(calc_chain.invoke({"movie": "Interstellar"}))
```

```
Movie: 1999 (26 years ago)
Movie: 2014 (11 years ago)
```

2) Prompt engineering

```
# The Task: Reject a candidate for a job.
task = "Write a rejection email to a candidate."

print("--- LAZY PROMPT ---")
print(llm.invoke(task).content)
```

```
--- LAZY PROMPT ---
Here are a few options for a rejection email, ranging from a standard, polite version to one that offers a bit more encouragement. Choose the one that best fits your company's culture an

---

**Option 1: Standard & Polite (Most Common)**

**Subject: Update on Your Application for [Job Title] at [Company Name]**

Dear [Candidate Name],

Thank you for your interest in the [Job Title] position at [Company Name] and for taking the time to [interview with our team / submit your application].

We appreciate you sharing your qualifications and experience with us. We received a large number of applications for this role, and after careful consideration, we have decided to move f

This was a very competitive search, and we truly appreciate the time and effort you invested in the process.

We wish you the best of luck in your job search and future endeavors.

Sincerely,

[Your Name]
[Your Title]
[Company Name]
[Company Website (Optional)]

---

**Option 2: Early Stage Rejection (No Interview)**

**Subject: Regarding Your Application for the [Job Title] Position at [Company Name]**

Dear [Candidate Name],

Thank you for your interest in the [Job Title] position at [Company Name] and for submitting your application.

We appreciate you sharing your resume and qualifications with us. We received a significant number of applications, and after careful review, we have decided to move forward with other c
```

```
[Your Title]
[Company Name]
[Company Website (Optional)]

---

**Option 2: Early Stage Rejection (No Interview)**

**Subject: Regarding Your Application for the [Job Title] Position at [Company Name]**

Dear [Candidate Name],

Thank you for your interest in the [Job Title] position at [Company Name] and for submitting your application.

We appreciate you sharing your resume and qualifications with us. We received a significant number of applications, and after careful review, we have decided to move forward with other c

We wish you the best of luck in your job search and future career.

Sincerely,

[Your Name]
[Your Title]
[Company Name]

---

**Option 3: With Encouragement for Future Roles**

**Subject: Update on Your Application for [Job Title] at [Company Name]**

Dear [Candidate Name],

Thank you for your interest in the [Job Title] position at [Company Name] and for taking the time to [interview with our team / submit your application]. We truly enjoyed learning more a

We received a high volume of applications from many talented individuals, and the selection process was very competitive. While your qualifications are impressive, we have decided to mov

We encourage you to keep an eye on our careers page at [Link to Careers Page] for future opportunities that may align with your skills and experience.

We wish you all the best in your job search and future career.

Sincerely,

[Your Name]
[Your Title]
[Company Name]
[Company Website (Optional)]

---
```

```
structured_prompt = """
# Context
You are an HR Manager at a quirky startup called 'RocketBoots'.

# Objective
Write a rejection email to a candidate named Bob.

# Constraints
1. Be extremely brief (under 50 words).
2. Do NOT say 'we found someone better'. Say 'the role changed'.
3. Sign off with 'Keep flying'.

# Output Format
Plain text, no subject line.
"""

print("--- STRUCTURED PROMPT ---")
print(llm.invoke(structured_prompt).content)
```

```
--- STRUCTURED PROMPT ---
Hi Bob,

Thank you for your interest in RocketBoots. We appreciate your time and effort.

While your application was impressive, the requirements for this role have recently changed. We won't be moving forward with your candidacy at th

Keep flying,
RocketBoots HR
```

In Cell 4, the lazy prompt gave me a generic, polite rejection email full of placeholders because I didn't provide constraints, so the model defaulted to standard corporate templates. In contrast, the structured prompt in Cell 6 produced a short, specific response that followed my instructions perfectly. By adding negative constraints and style guidelines, the model treated those details as strict rules, filtering out the usual fluff to generate exactly what I needed.

```python
#assigment solution

# 1. Define the Structured Prompt
python_prompt = """
# Context
You are a Senior Python Developer with 10+ years of experience in clean code and optimization.

# Objective
Write a Python function to reverse a string.

# Constraints
1. The function MUST use recursion.
2. Do NOT use string slicing (e.g., [::-1]) or the reversed() function.
3. Handle empty strings as a base case.

# Style
Include a detailed docstring in Google Style format explaining the recursion logic.

# Output Format
Return only the Python code block.
"""

# 2. Invoke
print(llm.invoke(python_prompt).content)
```

```
```python
def reverse_string_recursive(s: str) -> str:
    """Reverses a string using recursion.

    This function takes a string and reverses it by recursively processing
    the substring starting from the second character and appending the
    first character to the end of the reversed substring.

    The recursion works as follows:
    1.  **Base Case**: If the string is empty (`""`) or contains only one
        character (e.g., `"a"`), it is already reversed. In this scenario,
        the function returns the string as is, stopping the recursion.
    2.  **Recursive Step**: For any string `s` with more than one character:
        a.  The function makes a recursive call to itself with the substring
            `s[1:]`. This substring includes all characters of `s` *except*
            the first one. This call will eventually return the reversed
            version of `s[1:]`.
        b.  The first character of the original string, `s[0]`, is then
```

```
...     This function takes a string and reverses it by recursively processing
        the substring starting from the second character and appending the
        first character to the end of the reversed substring.

        The recursion works as follows:
        1.  **Base Case**: If the string is empty (`""`) or contains only one
            character (e.g., `"a"`), it is already reversed. In this scenario,
            the function returns the string as is, stopping the recursion.
        2.  **Recursive Step**: For any string `s` with more than one character:
            a.  The function makes a recursive call to itself with the substring
                `s[1:]`. This substring includes all characters of `s` *except*
                the first one. This call will eventually return the reversed
                version of `s[1:]`.
            b.  The first character of the original string, `s[0]`, is then
                concatenated to the *end* of the result obtained from the
                recursive call. This effectively moves the first character
                of the current string to the last position in the final
                reversed string segment.

        Example Trace for "abc":
            1. `reverse_string_recursive("abc")`
               - `s[1:]` is "bc", `s[0]` is "a"
               - Returns `reverse_string_recursive("bc")` + "a"

            2. `reverse_string_recursive("bc")`
               - `s[1:]` is "c", `s[0]` is "b"
               - Returns `reverse_string_recursive("c")` + "b"

            3. `reverse_string_recursive("c")`
               - `len(s)` is 1, which is the base case.
               - Returns "c"

            Now, substitute back:
            - Step 2 returns: "c" + "b" = "cb"
            - Step 1 returns: "cb" + "a" = "cba"

        Args:
            s: The input string to be reversed.

        Returns:
            The reversed string.
        """
        # Base case: An empty string or a string with a single character
        # is its own reverse. This also handles the termination of recursion.
        if len(s) <= 1:
            return s
        # Recursive step:
        # Take the substring from the second character to the end (s[1:]),
        # recursively reverse it, and then append the first character (s[0])
        # to the end of the result.
        return reverse_string_recursive(s[1:]) + s[0]
    ```
```

The model does indeed behave as a senior python engineer

Zero shot and one shot learning now

2. Zero-Shot (No Context)

The model relies purely on its training data.

```
prompt_zero = "Combine 'Angry' and 'Hungry' into a funny new word."
print(f"Zero-Shot: {llm.invoke(prompt_zero).content}")
```

```
Zero-Shot: The most common and widely accepted funny word for being angry because you're hungry is:

**Hangry**
```

3. Few-Shot (Pattern Matching)

We provide examples. The Attention Mechanism attends to the **Structure** (`Input -> Output`) and the **Tone** (Sarcasm).

```
prompt_few = """
Combine words into a funny new word. Give a sarcastic definition.

Input: Breakfast + Lunch
Output: Brunch (An excuse to drink alcohol before noon)

Input: Chill + Relax
Output: Chillax (What annoying people say when you are panic attacks)

Input: Angry + Hungry
Output:
"""
print(f"Few-Shot: {llm.invoke(prompt_few).content}")
```

```
Few-Shot: Output: Angry (The only acceptable reason for being a complete monster before your next meal.)
```

In Cell 3, the zero-shot prompt returned a basic definition of "Hangry" that lacked the specific humor or format I wanted because the model had no context for the tone. However, in Cell 5, providing examples changed everything, and the model successfully outputs a witty definition. This worked because the examples provided a clear pattern for "In-Context Learning," allowing the model to mimic both the syntax and the sarcastic style I was looking for.

```
from langchain_core.prompts import ChatPromptTemplate, FewShotChatMessagePromptTemplate

# 1. Our Database of Examples
examples = [
    {"input": "The internet is down.", "output": "We are observing connectivity latency."},
    {"input": "This code implies a bug.", "output": "The logic suggests unintended behavior."},
    {"input": "I hate this feature.", "output": "This feature does not align with my preferences."},
]

# 2. Template for ONE example
example_fmt = ChatPromptTemplate.from_messages([
    ("human", "{input}"),
    ("ai", "{output}")
])

# 3. The Few-Shot Container
few_shot_prompt = FewShotChatMessagePromptTemplate(
    example_prompt=example_fmt,
    examples=examples
)

# 4. The Final Chain
final_prompt = ChatPromptTemplate.from_messages([
    ("system", "You are a Corpo-Speak Translator. Rewrite the input to sound professional."),
    few_shot_prompt,      # Inject examples here
    ("human", "{text}")
])

chain = final_prompt | llm

print(chain.invoke({"text": "This app sucks."}).content)
```

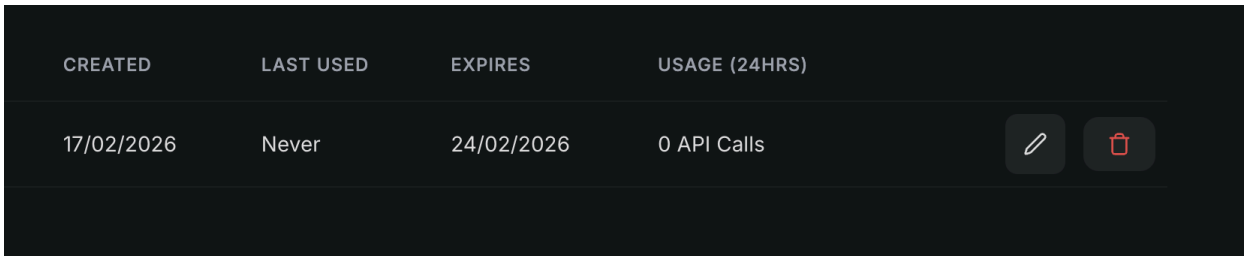
```
The application presents areas for strategic optimization.
```

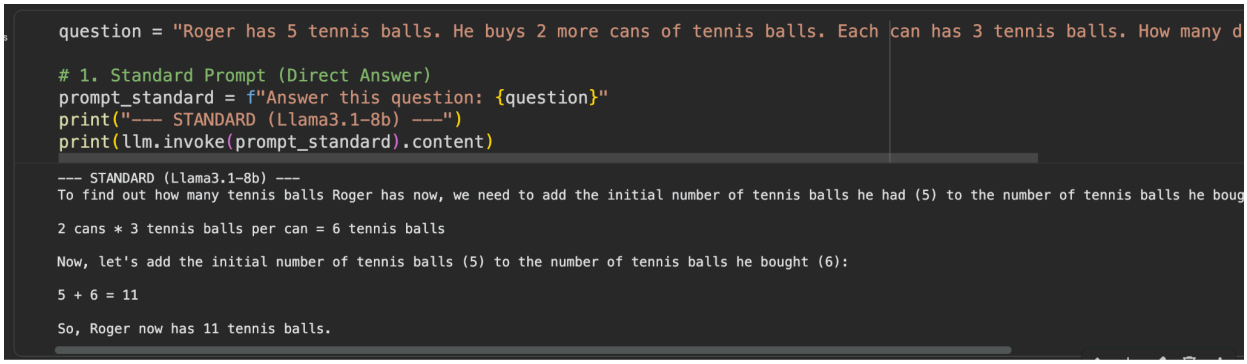
In Cell 3, the model successfully translated my casual complaint into professional corporate language because the template formatted my examples into a clear conversation history. The

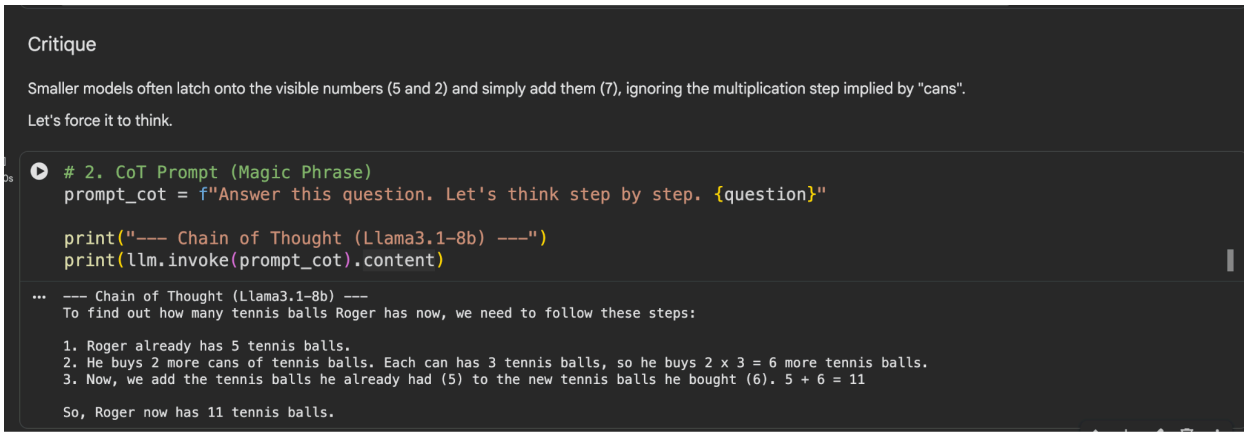
model recognized the pattern of rewriting complaints and applied it to the new input. This method is clearly better for modularity since I could easily swap out dozens of examples without having to manually rewrite the prompt string every time.

3) Advanced Prompting

After creating the groq api key:

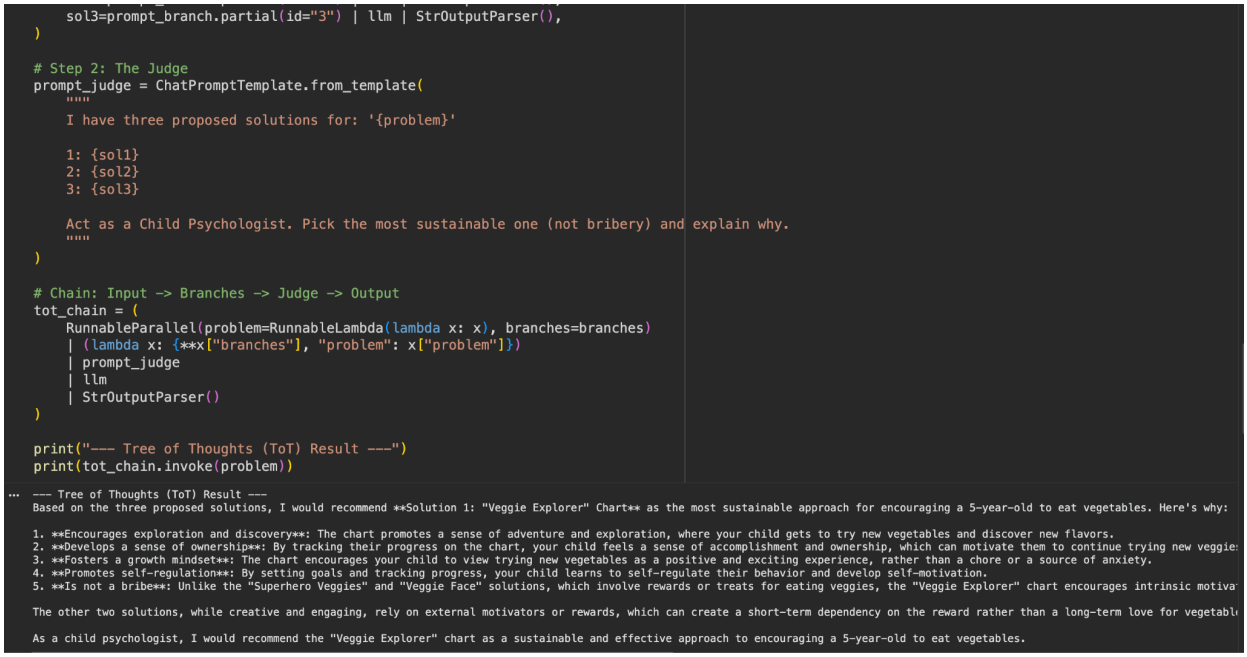






In the standard prompt experiment, the smaller Llama model failed the logic question by jumping straight to the wrong answer. It likely latched onto the visible numbers five and two and simply added them to get seven, completely ignoring the multiplication implied by the word

cans. However, when the chain of thought prompt was applied with the magic phrase let's think step by step, the model successfully corrected itself. By generating the intermediate steps and calculating that two cans equal six balls first, it created a logical path to the correct answer of eleven, proving that the model needs to produce intermediate tokens to reason effectively.



```
    sol3=prompt_branch.partial(id="3") | llm | StrOutputParser(),
)

# Step 2: The Judge
prompt_judge = ChatPromptTemplate.from_template(
    """
    I have three proposed solutions for: '{problem}'

    1: {sol1}
    2: {sol2}
    3: {sol3}

    Act as a Child Psychologist. Pick the most sustainable one (not bribery) and explain why.
    """
)

# Chain: Input -> Branches -> Judge -> Output
tot_chain = (
    RunnableParallel(problem=RunnableLambda(lambda x: x), branches=branches)
    | (lambda x: {**x["branches"], "problem": x["problem"]})
    | prompt_judge
    | llm
    | StrOutputParser()
)

print("--- Tree of Thoughts (ToT) Result ---")
print(tot_chain.invoke(problem))
```

```
--- Tree of Thoughts (ToT) Result ---
Based on the three proposed solutions, I would recommend **Solution 1: "Veggie Explorer" Chart** as the most sustainable approach for encouraging a 5-year-old to eat vegetables. Here's why:

1. **Encourages exploration and discovery**: The chart promotes a sense of adventure and exploration, where your child gets to try new vegetables and discover new flavors.
2. **Develops a sense of ownership**: By tracking their progress on the chart, your child feels a sense of accomplishment and ownership, which can motivate them to continue trying new veggies
3. **Fosters a growth mindset**: The chart encourages your child to view trying new vegetables as a positive and exciting experience, rather than a chore or a source of anxiety.
4. **Promotes self-regulation**: By setting goals and tracking progress, your child learns to self-regulate their behavior and develop self-motivation.
5. **Is not a bribe**: Unlike the "Superhero Veggies" and "Veggie Face" solutions, which involve rewards or treats for eating veggies, the "Veggie Explorer" chart encourages intrinsic motiva

The other two solutions, while creative and engaging, rely on external motivators or rewards, which can create a short-term dependency on the reward rather than a long-term love for vegetabl

As a child psychologist, I would recommend the "Veggie Explorer" chart as a sustainable and effective approach to encouraging a 5-year-old to eat vegetables.
```

The tree of thoughts execution demonstrated a decision-making process that went beyond simple generation. Instead of providing one immediate answer to the parenting problem, the code forced the model to brainstorm three distinct solutions in parallel branches. The most interesting part was the judge step, which evaluated these three options against specific criteria like sustainability rather than just picking the first one. This made the output feel less like a random generation and more like a reasoned choice.

For the graph of thought egs

```
# 1. The Generator (Divergence)
prompt_draft = ChatPromptTemplate.from_template(
    "Write a 1-sentence movie plot about: {topic}. Genre: {genre}."
)

drafts = RunnableParallel(
    draft_scifi=prompt_draft.partial(genre="Sci-Fi") | llm | StrOutputParser(),
    draft_romance=prompt_draft.partial(genre="Romance") | llm | StrOutputParser(),
    draft_horror=prompt_draft.partial(genre="Horror") | llm | StrOutputParser(),
)

# 2. The Aggregator (Convergence)
prompt_combine = ChatPromptTemplate.from_template(
    """
    I have three movie ideas for the topic '{topic}':
    1. Sci-Fi: {draft_scifi}
    2. Romance: {draft_romance}
    3. Horror: {draft_horror}

    Your task: Create a new Mega-Movie that combines the TECHNOLOGY of Sci-Fi, the PASSION of Romance, and the FEAR of Horror.
    Write one paragraph.
    """
)

# 3. The Chain
got_chain = (
    RunnableParallel(topic=RunnableLambda(lambda x: x), drafts=drafts)
    | (lambda x: {**x["drafts"], "topic": x["topic"]})
    | prompt_combine
    | llm
    | StrOutputParser()
)

print("--- Graph of Thoughts (GoT) Result ---")
print(got_chain.invoke("Time Travel"))
```

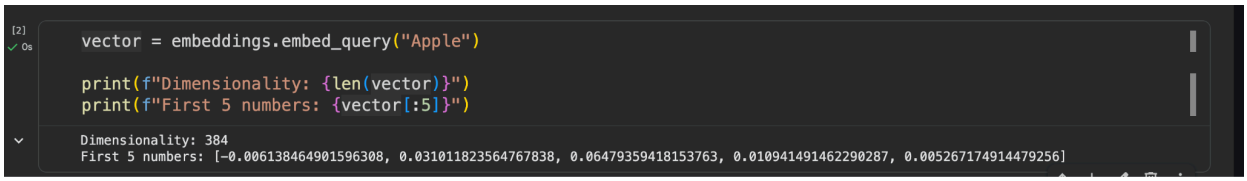
```
--- Graph of Thoughts (GoT) Result ---
In "Eternal Echoes," a brilliant physicist, Emma, discovers a revolutionary time machine that uses the manipulation of space-time continua to traverse the ages. As she travels through time,
```

In the graph of thoughts section, the experiment showed how to synthesize information rather than just select it. The code generated three independent movie plots based on sci-fi, romance, and horror genres and then aggregated them into a single master plot. This proved that the framework can break a creative task into smaller components and then merge them back together, acting like a writer's room that combines different genre elements into one coherent narrative.

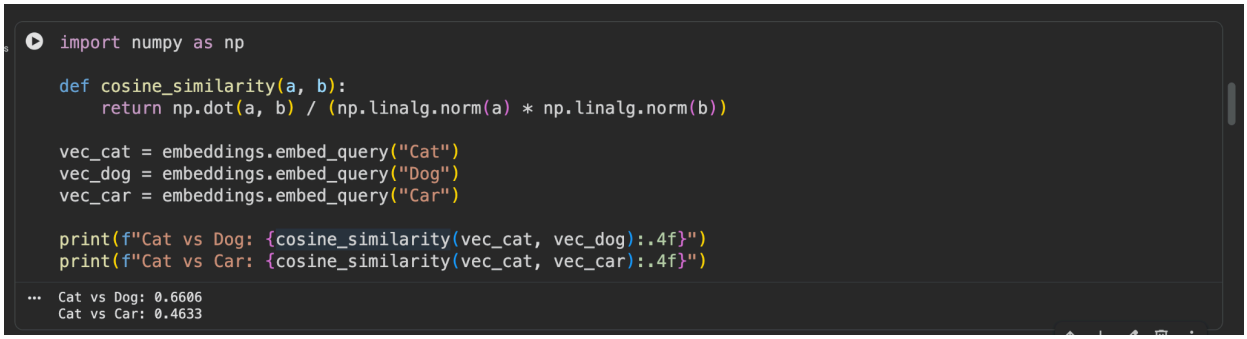
Method	Structure	Best For...	Cost/Latency
Simple Prompt	Input -> Output	Simple facts, summaries	Low
CoT (Chain)	Input -> Steps -> Output	Math, Logic, Debugging	Med
ToT (Tree)	Input -> 3x Branches -> Select -> Output	Strategic decisions, Brainstorming	High

GoT (Graph)	Input -> Branch -> Mix/Aggregate -> Output	Creative Writing, Research Synthesis	V. High

4) RAG and Vector stores



The output reveals that an embedding is simply a high-dimensional list of numbers (384 dimensions for the MiniLM model). This confirms that the machine represents semantic concepts, like the word "Apple," as a mathematical vector rather than understanding the word itself.



The experiment with cosine similarity demonstrates that these vector numbers encode actual meaning. The result shows a significantly higher similarity score for "Cat" and "Dog" compared to "Cat" and "Car," proving that the model understands the semantic relationship between animals versus vehicles based purely on the angle between their vectors.

```
[7]
✓ 0s
from langchain_core.documents import Document

docs = [
    Document(page_content="Piyush's favorite food is Pizza with extra cheese."),
    Document(page_content="The secret password to the lab is 'Blueberry'."),
    Document(page_content="LangChain is a framework for developing applications powered by language models."),
]
```

```
[9]
✓ 1s
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.output_parsers import StrOutputParser
from langchain_core.runnables import RunnablePassthrough

template = """
Answer based ONLY on the context below:
{context}

Question: {question}
"""
prompt = ChatPromptTemplate.from_template(template)

chain = (
    {"context": retriever, "question": RunnablePassthrough()}
    | prompt
    | llm
    | StrOutputParser()
)

result = chain.invoke("What is the secret password?")
print(result)

The secret password to the lab is 'Blueberry'.
```

This execution demonstrates the core power of Retrieval Augmented Generation. Despite the model having no prior training on the specific "secret password," it successfully retrieved the correct context ("Blueberry") from the provided document list and used it to answer the query. This confirms that the RAG pipeline effectively allows the LLM to access and utilize external, non-parametric knowledge sources to answer questions it otherwise couldn't.

```
[11]
✓ 0s
index = faiss.IndexFlatL2(d)
index.add(xb)
print(f"Flat Index contains {index.ntotal} vectors")

Flat Index contains 10000 vectors
```

```
[14]
✓ 1s
m = 8 # Split vector into 8 sub-vectors
index_pq = faiss.IndexPQ(d, m, 8)
index_pq.train(xb)
index_pq.add(xb)
print("PQ Compression complete. RAM usage minimized.")

PQ Compression complete. RAM usage minimized.
```

The progression through different indexing methods highlights the trade-offs involved in scaling vector search. The Flat Index provides perfect accuracy but is computationally expensive, whereas the IVF method improves speed by clustering data into partitions. The HNSW algorithm

uses a graph-based structure for faster navigation, though it consumes more memory. Finally, the Product Quantization method demonstrates how vector compression can significantly reduce memory usage, allowing for the storage of massive datasets at the cost of some precision.

MOE Assignment:

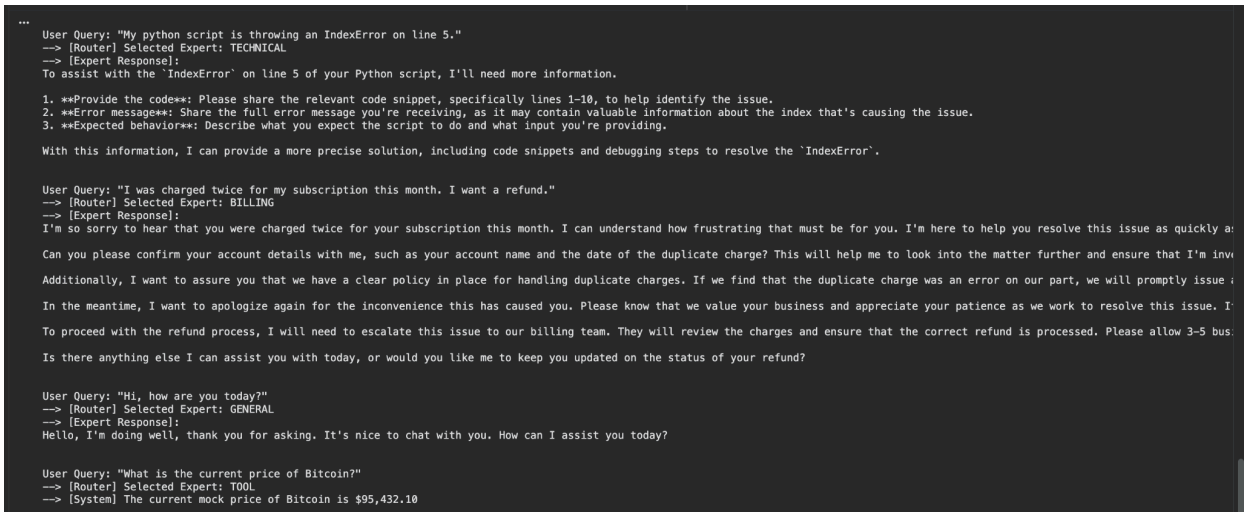
```python
import os
import getpass
from dotenv import load_dotenv
from groq import Groq

load_dotenv()

if "GROQ_API_KEY" not in os.environ:
    os.environ["GROQ_API_KEY"] = getpass.getpass("Enter your Groq API Key: ")

client = Groq(api_key=os.environ["GROQ_API_KEY"])

MODEL_CONFIG = {
    "technical": {
        "model": "llama-3.3-70b-versatile",
        "system_prompt": (
            "You are a Senior Technical Support Engineer. "
            "You are rigorous, code-focused, and precise. "
            "Provide technical solutions, code snippets, and debugging steps. "
            "Do not be overly chatty; focus on the fix."
        )
    },
    "billing": {
        "model": "llama-3.3-70b-versatile",
        "system_prompt": (
            "You are a Customer Success Manager specializing in Billing. "
            "You are empathetic, polite, and policy-driven. "
            "Explain charges clearly and guide users through refund processes. "
            "Apologize for any inconvenience."
        )
    },
    "general": {
        "model": "llama-3.3-70b-versatile",
        "system_prompt": (
            "You are a helpful general assistant. "
            "Answer questions politely and concisely. "
            "If you don't know the answer, admit it."
```

```
        },
        "general": {
            "model": "llama-3.3-70b-versatile",
            "system_prompt": (
                "You are a helpful general assistant. "
                "Answer questions politely and concisely. "
                "If you don't know the answer, admit it."
            )
        },
        "tool": {
            "type": "function",
            "handler": "get_bitcoin_price"
        }
    }

    def route_prompt(user_input):
        """
        Decides which expert should handle the query.
        Returns: 'technical', 'billing', 'general', or 'tool'
        """
        completion = client.chat.completions.create(
            # UPDATED: Using 'llama-3.1-8b-instant' for the router as it is faster and cheaper
            model="llama-3.1-8b-instant",
            messages=[
                {
                    "role": "system",
                    "content": (
                        "You are a classification system. "
                        "Analyze the user's input and classify it into one of these categories:\n"
                        "- technical (for code errors, bugs, software issues)\n"
                        "- billing (for refunds, payments, subscriptions)\n"
                        "- tool (specifically for questions about the price of Bitcoin)\n"
                        "- general (for greetings or anything else)\n\n"
                        "Output ONLY the category name (lowercase). Do not explain."
                    )
                },
                {
                    "role": "user",
                    "content": user_input
                }
```

```
            ],
            temperature=0.0, # Strict determinism
            max_tokens=10
        )

        category = completion.choices[0].message.content.strip().lower()
        return category

    def get_bitcoin_price():
        return "The current mock price of Bitcoin is $95,432.10"

    def process_request(user_input):
        print(f"\nUser Query: \"{user_input}\"")

        category = route_prompt(user_input)

        if category not in MODEL_CONFIG:
            category = "general"

        print(f"--> [Router] Selected Expert: {category.upper()}")

        if category == "tool":
            result = get_bitcoin_price()
            print(f"--> [System] {result}")
            return result

        expert_config = MODEL_CONFIG[category]

        completion = client.chat.completions.create(
            model=expert_config["model"],
            messages=[
                {"role": "system", "content": expert_config["system_prompt"]},
                {"role": "user", "content": user_input}
            ],
            temperature=0.7
        )

        response = completion.choices[0].message.content
        print(f"--> [Expert Response]:\n{response}\n")
        return response
```

```
                {"role": "system", "content": expert_config["system_prompt"]},
                {"role": "user", "content": user_input}
            ],
            temperature=0.7
        )

        response = completion.choices[0].message.content
        print(f"--> [Expert Response]:\n{response}\n")
        return response

    if __name__ == "__main__":
        process_request("My python script is throwing an IndexError on line 5.")
        process_request("I was charged twice for my subscription this month. I want a refund.")
        process_request("Hi, how are you today?")
        process_request("What is the current price of Bitcoin?")
```

The answer is



In this assignment, the construction of the Mixture of Experts router demonstrated how a single model can be repurposed for widely different roles just by altering the system prompt. The technical expert provided rigorous code fixes while the billing expert remained empathetic and policy-driven, proving that the persona is just as important as the raw intelligence of the model. The router itself required a temperature of zero to function reliably; otherwise, it would occasionally hallucinate conversational filler instead of returning a clean category label. The addition of the tool use capability was particularly insightful, as it showed how to offload deterministic queries like stock prices to a Python function, thereby saving token costs and ensuring perfect accuracy compared to the LLM's static training data.