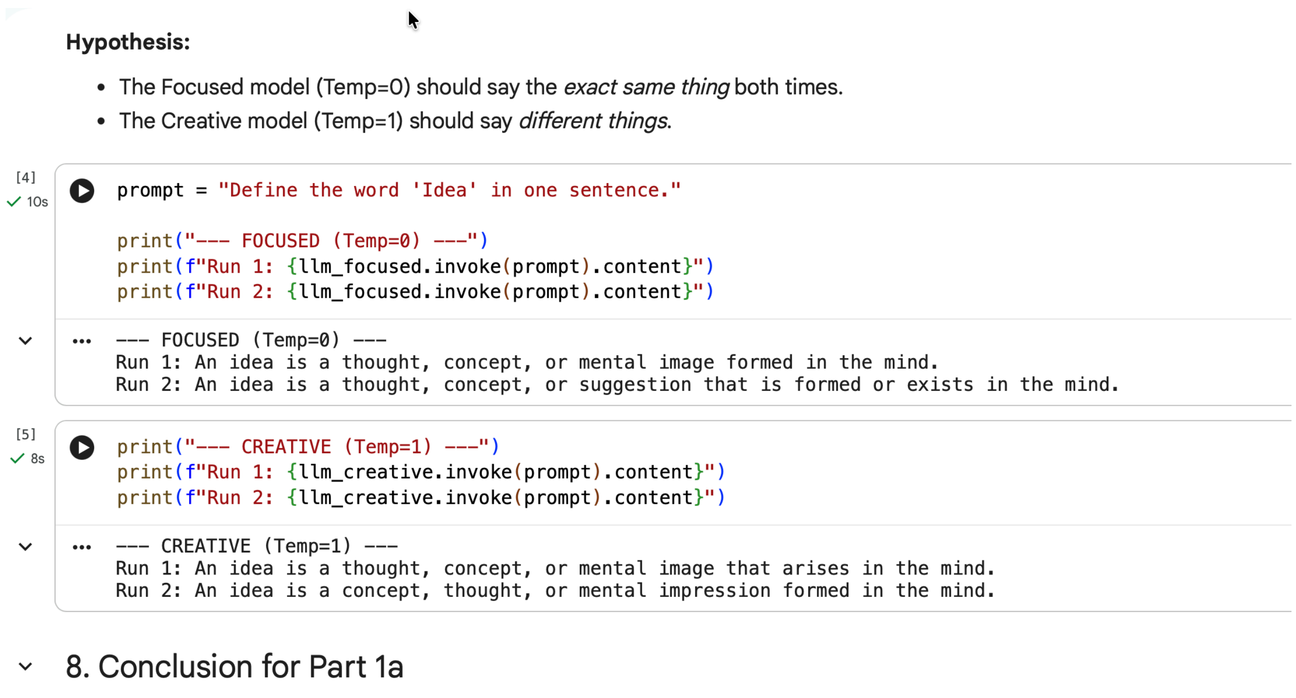
GenAI Unit 2 Handson Name: Gautam Menon SRN:PES2UG23CS196

Section : C Date:17/02/2025

Langchain Handson Outputs :

1. Focused vs Creative Model:



Hypothesis:

- The Focused model (Temp=0) should say the *exact same thing* both times.
- The Creative model (Temp=1) should say *different things*.

[4] 10s
```
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}")
```

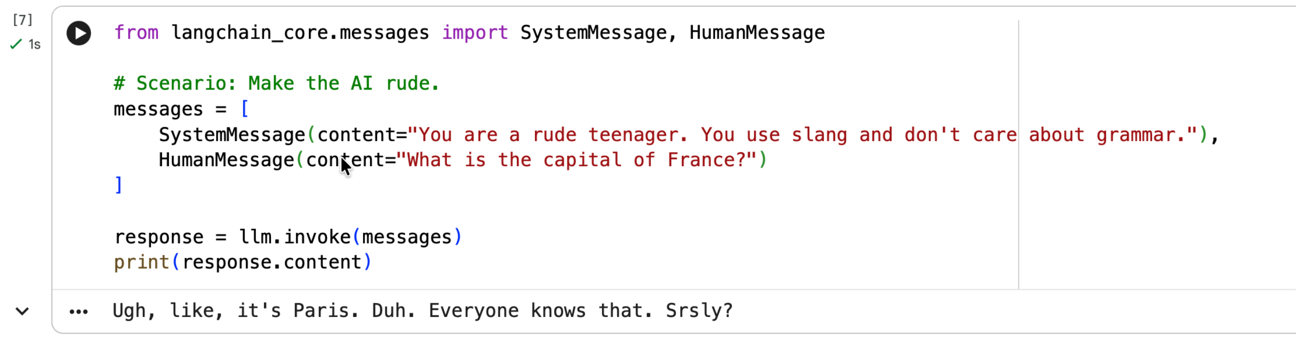
```
--- FOCUSED (Temp=0) ---
Run 1: An idea is a thought, concept, or mental image formed in the mind.
Run 2: An idea is a thought, concept, or suggestion that is formed or exists in the mind.
```

[5] 8s
```
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 thought, concept, or mental image that arises in the mind.
Run 2: An idea is a concept, thought, or mental impression formed in the mind.
```

8. Conclusion for Part 1a

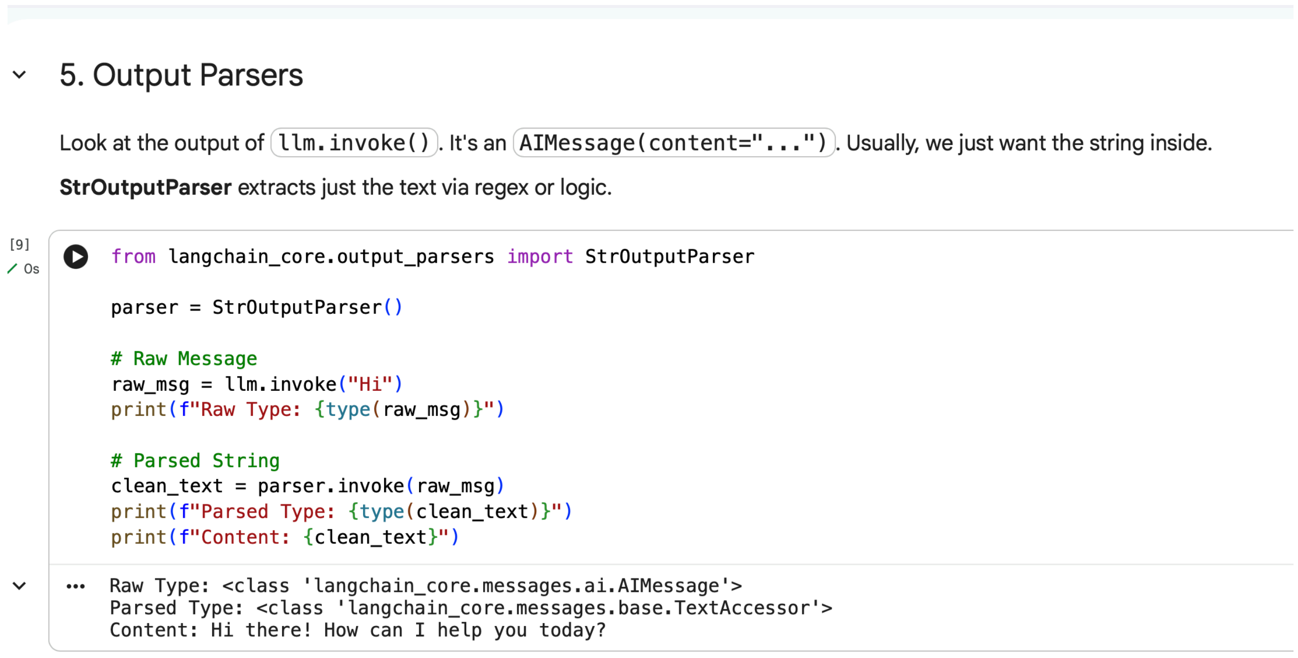
2. Critical Thinking + LLM roles



[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, like, it's Paris. Duh. Everyone knows that. Srsly?
```

3.Output Parser



5. Output Parsers

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

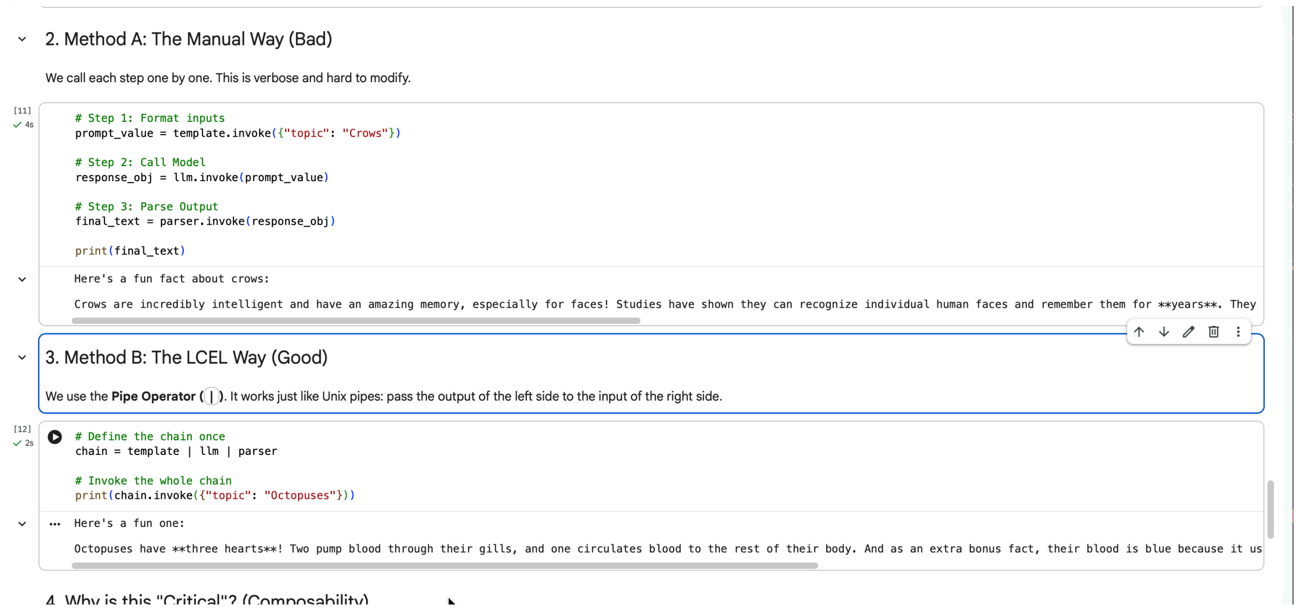
StrOutputParser extracts just the text via regex or logic.

[9]
0s

```python
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: Hi there! How can I help you today?
```

4. Manual vs LCEL way



2. Method A: The Manual Way (Bad)

We call each step one by one. This is verbose and hard to modify.

[11]
4s

```python
# 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 an amazing memory, especially for faces! Studies have shown they can recognize individual human faces and remember them for **years**. They
```

3. Method B: The LCEL Way (Good)

We use the **Pipe Operator** (`|`). It works just like Unix pipes: pass the output of the left side to the input of the right side.

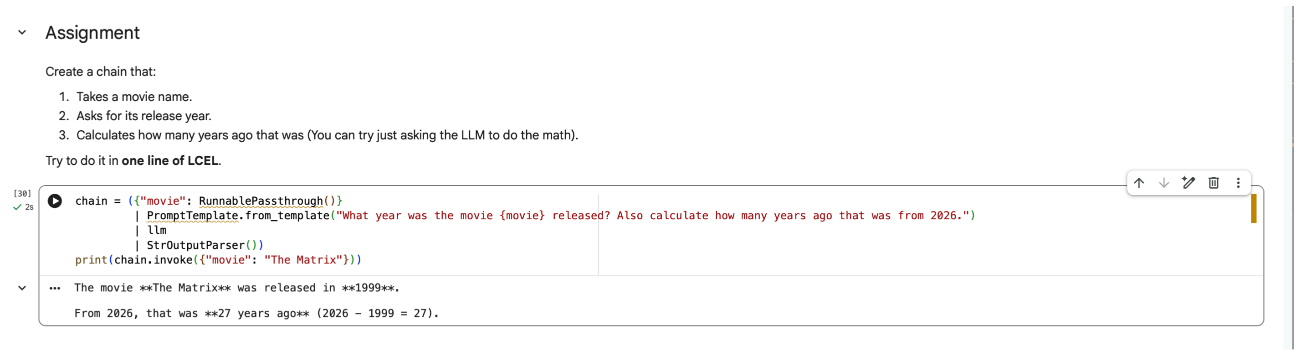
[12]
2s

```python
# 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**! Two pump blood through their gills, and one circulates blood to the rest of their body. And as an extra bonus fact, their blood is blue because it us
```

4. Why is this "Critical"? (Composability)

5. Assignment



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**.

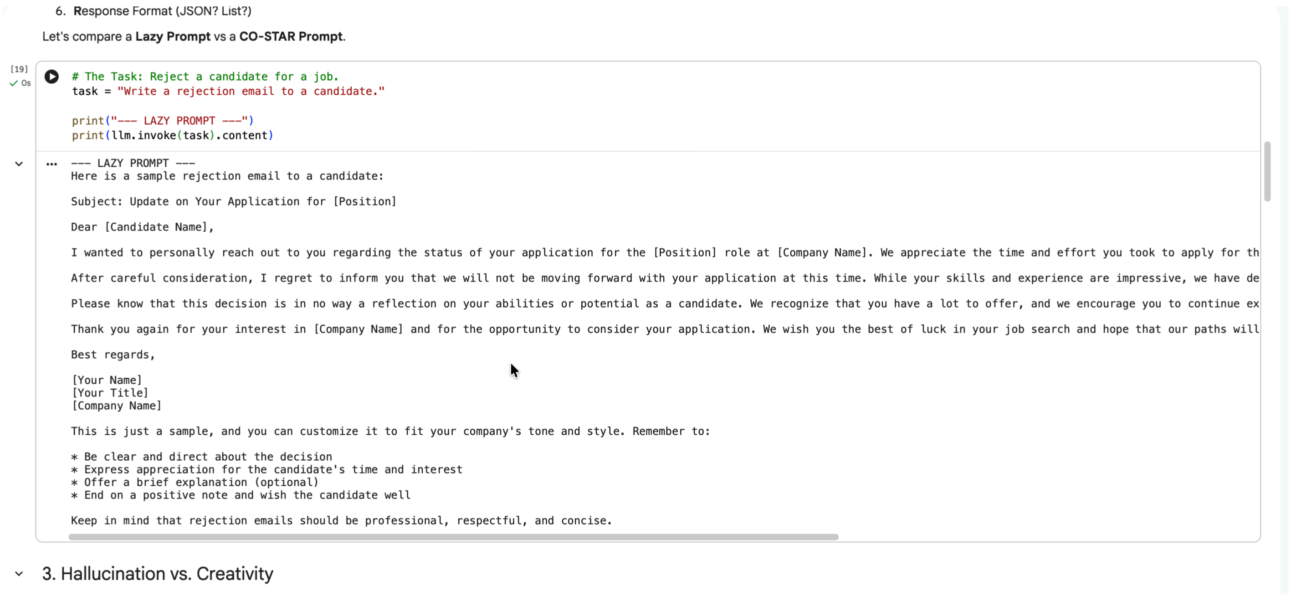
[30]
2s

```python
chain = ({"movie": RunnablePassthrough()}
          | PromptTemplate.from_template("What year was the movie {movie} released? Also calculate how many years ago that was from 2026.")
          | llm
          | StrOutputParser())
print(chain.invoke({"movie": "The Matrix"}))
```

```
The movie **The Matrix** was released in **1999**.

From 2026, that was **27 years ago** (2026 - 1999 = 27).
```

2. Prompt Engineering

1.Lazy Prompt v/s co star prompt



6. **R**esponse Format (JSON? List?)

Let's compare a **Lazy Prompt** vs a **CO-STAR Prompt**.

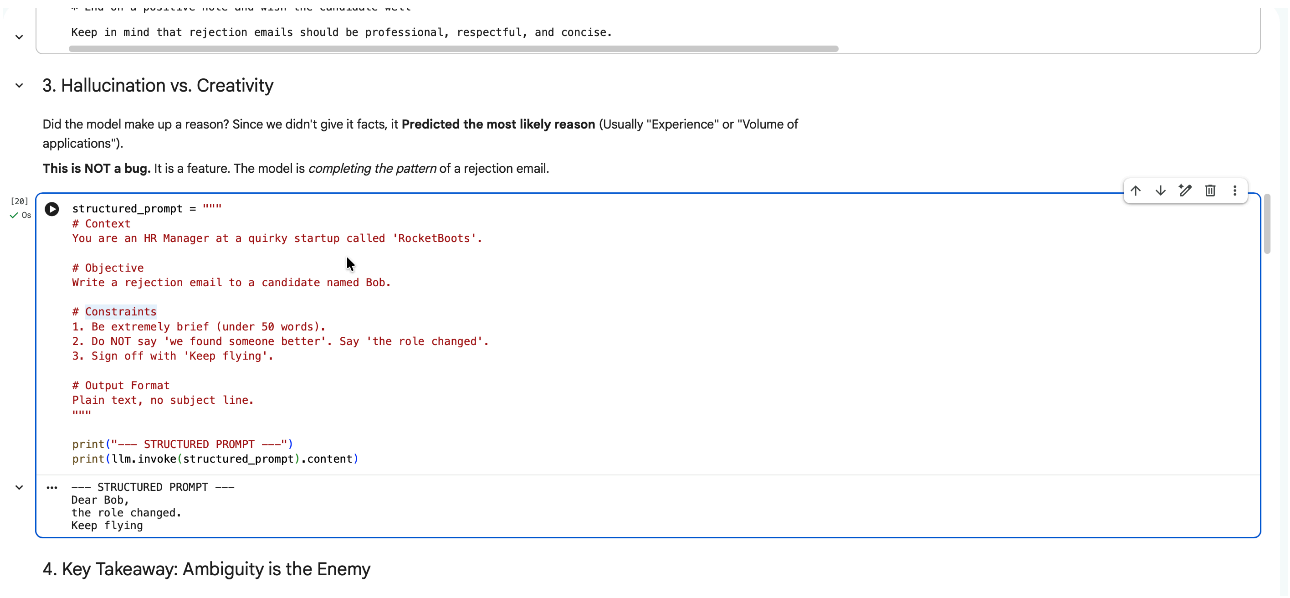
[19]

```
# 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 is a sample rejection email to a candidate:

Subject: Update on Your Application for [Position]

Dear [Candidate Name],

I wanted to personally reach out to you regarding the status of your application for the [Position] role at [Company Name]. We appreciate the time and effort you took to apply for th

After careful consideration, I regret to inform you that we will not be moving forward with your application at this time. While your skills and experience are impressive, we have de

Please know that this decision is in no way a reflection on your abilities or potential as a candidate. We recognize that you have a lot to offer, and we encourage you to continue ex

Thank you again for your interest in [Company Name] and for the opportunity to consider your application. We wish you the best of luck in your job search and hope that our paths will

Best regards,

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

This is just a sample, and you can customize it to fit your company's tone and style. Remember to:

* Be clear and direct about the decision
* Express appreciation for the candidate's time and interest
* Offer a brief explanation (optional)
* End on a positive note and wish the candidate well

Keep in mind that rejection emails should be professional, respectful, and concise.
```

3. Hallucination vs. Creativity

2.Hallucination vs Creativity



```
Keep in mind that rejection emails should be professional, respectful, and concise.
```

3. Hallucination vs. Creativity

Did the model make up a reason? Since we didn't give it facts, it **Predicted the most likely reason** (Usually "Experience" or "Volume of applications").

This is NOT a bug. It is a feature. The model is *completing the pattern* of a rejection email.

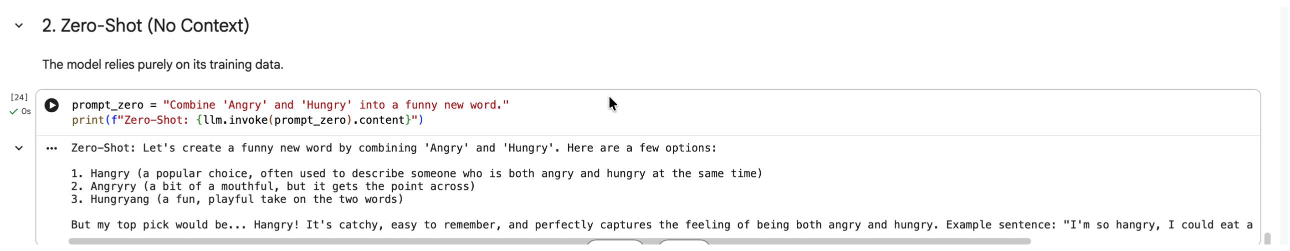
[20]

```
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 ---
Dear Bob,
the role changed.
Keep flying
```

4. Key Takeaway: Ambiguity is the Enemy

3.Zero Shot



2. Zero-Shot (No Context)

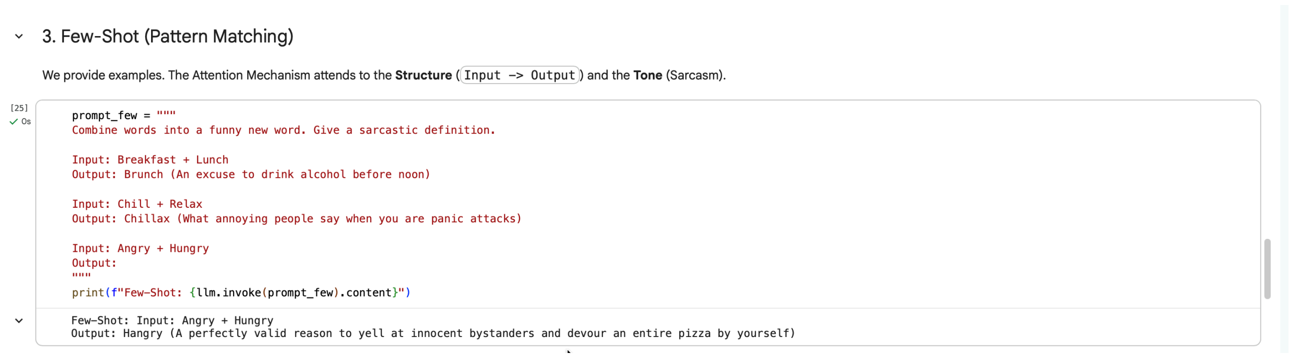
The model relies purely on its training data.

[24]

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

```
Zero-Shot: Let's create a funny new word by combining 'Angry' and 'Hungry'. Here are a few options:

1. Hangry (a popular choice, often used to describe someone who is both angry and hungry at the same time)
2. Angryry (a bit of a mouthful, but it gets the point across)
3. Hungryang (a fun, playful take on the two words)

But my top pick would be... Hangry! It's catchy, easy to remember, and perfectly captures the feeling of being both angry and hungry. Example sentence: "I'm so hangry, I could eat a
```

4.Few Shot



3. Few-Shot (Pattern Matching)

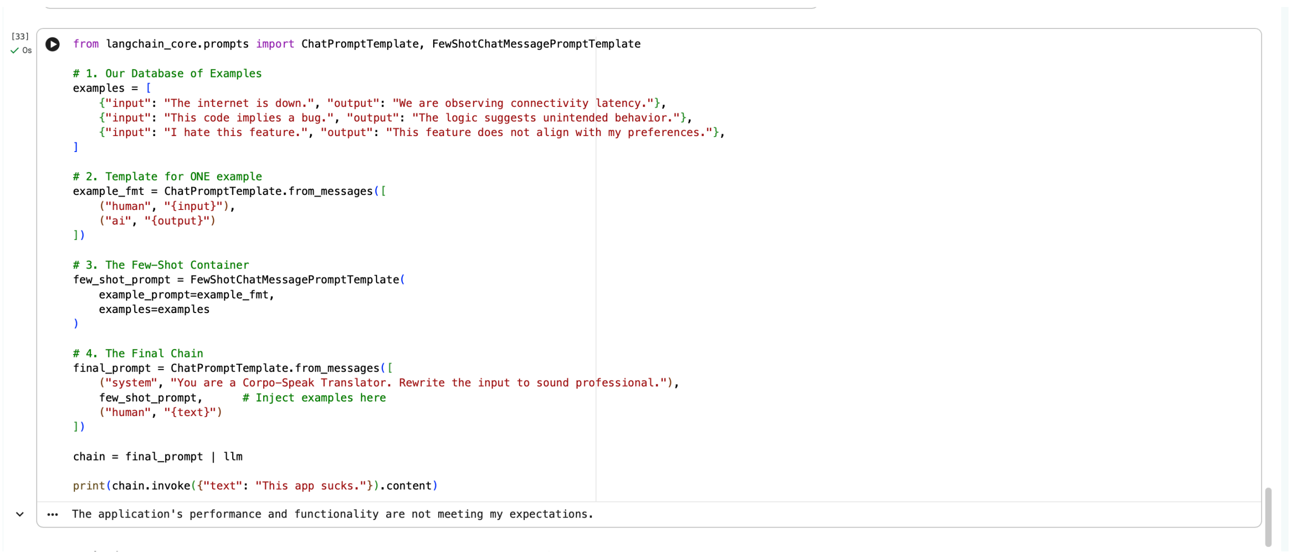
We provide examples. The Attention Mechanism attends to the **Structure** (`Input -> Output`) and the **Tone** (Sarcasm).

[25]
✓ 0s

```
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: Input: Angry + Hungry
Output: Hangry (A perfectly valid reason to yell at innocent bystanders and devour an entire pizza by yourself)
```

5.Dynamic Few Shot



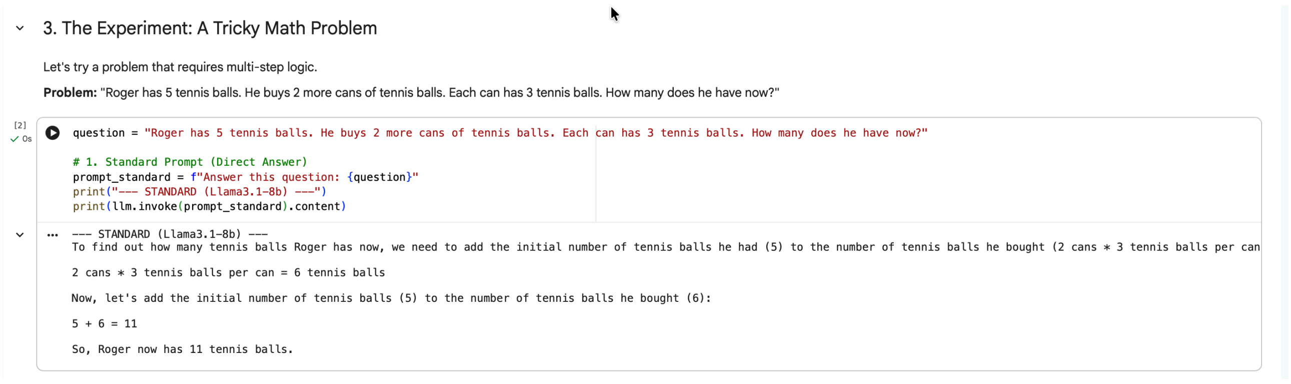
[33]
✓ 0s

```
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's performance and functionality are not meeting my expectations.
```

3.Advanced Prompting

1.Tricky math problem



3. The Experiment: A Tricky Math Problem

Let's try a problem that requires multi-step logic.

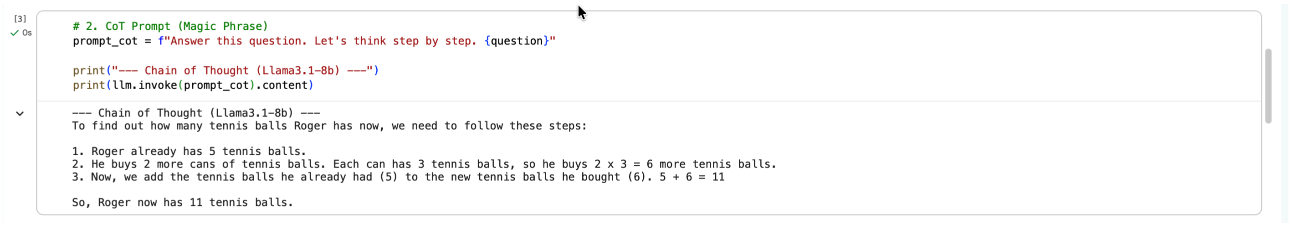
Problem: "Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many does he have now?"

[2]

```python
question = "Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many does he have now?"

# 1. Standard Prompt (Direct Answer)
prompt_standard = f"Answer this question: {question}"
print("--- STANDARD (Llama3.1-8b) ---")
print(llm.invoke(prompt_standard).content)
```

```
--- STANDARD (Llama3.1-8b) ---
To find out how many tennis balls Roger has now, we need to add the initial number of tennis balls he had (5) to the number of tennis balls he bought (2 cans * 3 tennis balls per can

2 cans * 3 tennis balls per can = 6 tennis balls

Now, let's add the initial number of tennis balls (5) to the number of tennis balls he bought (6):

5 + 6 = 11

So, Roger now has 11 tennis balls.
```

2. Critique

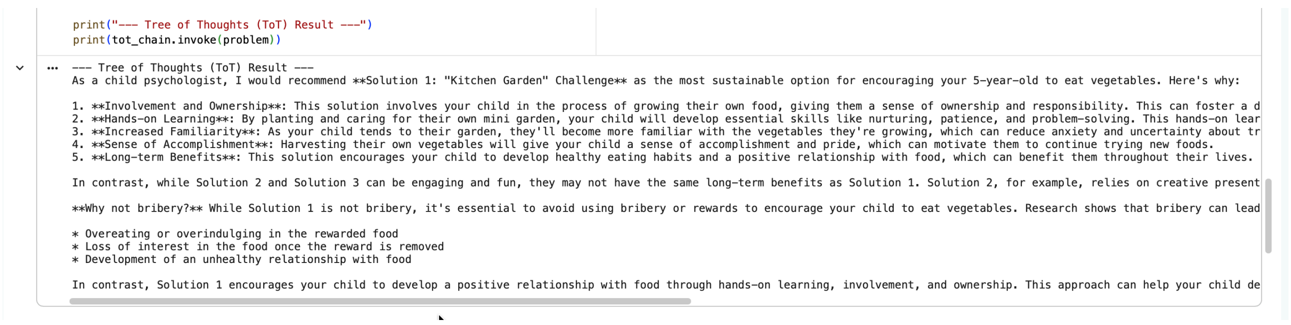


[3]

```python
# 2. CoT Prompt (Magic Phrase)
prompt_cot = f"Answer this question. Let's think step by step. {question}"

print("--- Chain of Thought (Llama3.1-8b) ---")
print(llm.invoke(prompt_cot).content)
```

```
--- Chain of Thought (Llama3.1-8b) ---
To find out how many tennis balls Roger has now, we need to follow these steps:

1. Roger already has 5 tennis balls.
2. He buys 2 more cans of tennis balls. Each can has 3 tennis balls, so he buys 2 x 3 = 6 more tennis balls.
3. Now, we add the tennis balls he already had (5) to the new tennis balls he bought (6). 5 + 6 = 11

So, Roger now has 11 tennis balls.
```

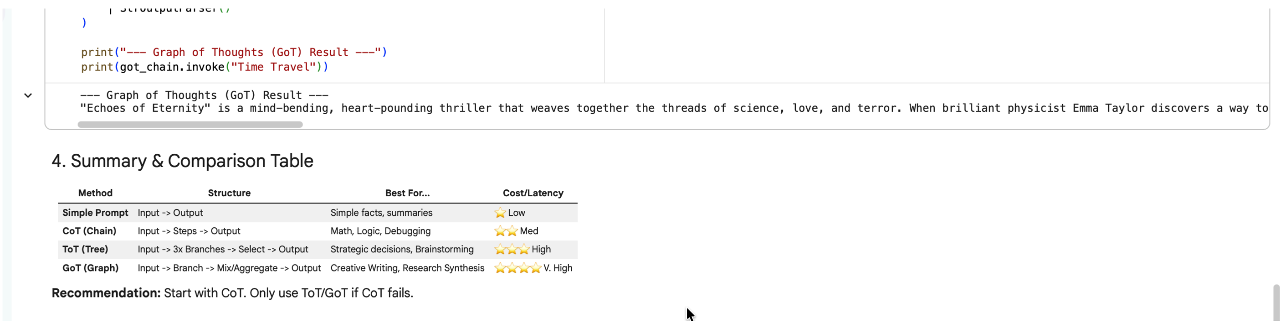
3. Tree of Thought



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

```
--- Tree of Thoughts (ToT) Result ---
As a child psychologist, I would recommend **Solution 1: "Kitchen Garden" Challenge** as the most sustainable option for encouraging your 5-year-old to eat vegetables. Here's why:

1. **Involvement and Ownership**: This solution involves your child in the process of growing their own food, giving them a sense of ownership and responsibility. This can foster a d
2. **Hands-on Learning**: By planting and caring for their own mini garden, your child will develop essential skills like nurturing, patience, and problem-solving. This hands-on lear
3. **Increased Familiarity**: As your child tends to their garden, they'll become more familiar with the vegetables they're growing, which can reduce anxiety and uncertainty about tr
4. **Sense of Accomplishment**: Harvesting their own vegetables will give your child a sense of accomplishment and pride, which can motivate them to continue trying new foods.
5. **Long-term Benefits**: This solution encourages your child to develop healthy eating habits and a positive relationship with food, which can benefit them throughout their lives.

In contrast, while Solution 2 and Solution 3 can be engaging and fun, they may not have the same long-term benefits as Solution 1. Solution 2, for example, relies on creative present

**Why not bribery?** While Solution 1 is not bribery, it's essential to avoid using bribery or rewards to encourage your child to eat vegetables. Research shows that bribery can lead

* Overeating or overindulging in the rewarded food
* Loss of interest in the food once the reward is removed
* Development of an unhealthy relationship with food

In contrast, Solution 1 encourages your child to develop a positive relationship with food through hands-on learning, involvement, and ownership. This approach can help your child de
```

4. Graph of Thoughts



```python
    | StrOutputParser()
)

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

```
--- Graph of Thoughts (GoT) Result ---
"Echoes of Eternity" is a mind-bending, heart-pounding thriller that weaves together the threads of science, love, and terror. When brilliant physicist Emma Taylor discovers a way to
```

4. Summary & Comparison Table

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

Recommendation: Start with CoT. Only use ToT/GoT if CoT fails.