Lab 2 - Evaluating Foundation Models with LangChain

In this lab, we evaluate the foundation models using two patient cases.

We will explore different evaluation approaches and compare the responses of GPT-4o and Gemini-2.0-flash.

We will experiment the following four approaches when labeled groud-truth data is available:

1. Production efficiency
2. Semantic similarity
3. AI as a judge
4. NLP-based Metrics

1 Set up the enviroment

In [ ]:
```
# Install required packages in silent mode
%pip install -U -q langchain-together langchain-google-genai tabulate nlt
```

In [ ]:
```
# Load required packages
from dotenv import load_dotenv
import pandas as pd
import numpy as np
import os
from IPython.display import Markdown
from tabulate import tabulate
from langchain_openai import OpenAIEmbeddings
from langchain_core.prompts import PromptTemplate
from langchain_openai import ChatOpenAI
from langchain.schema import SystemMessage, HumanMessage
from langchain.evaluation import load_evaluator
from langchain_together import ChatTogether
from langchain_google_genai import ChatGoogleGenerativeAI
import time
from contextlib import contextmanager
```

In [ ]:
```
# Load environment variables from a .env file
load_dotenv()

# Access your API Keys
openai_api_key = os.getenv("OPENAI_API_KEY")
together_api_key = os.getenv("TOGETHER_API_KEY")
google_api_key = os.getenv("GEMINI_API_KEY")
```

2 Initialize models

We initialize three foundation models. DeepSeek-V3 will be used as an AI judge.

In [ ]:
```python
# Initialize the GPT-4o model from OpenAI
llm_gpt = ChatOpenAI(
    model_name="gpt-4o",
    temperature=0,
    max_retries=2,
    timeout= 120, # Sets a request timeout of 120 seconds
    openai_api_key=openai_api_key)

# Initialize the DeepSeek-V3 using Together API
llm_deepseek = ChatTogether(
    model="deepseek-ai/DeepSeek-V3",
    temperature=0,
    max_retries=2,
    timeout= 120,
    together_api_key= together_api_key
)

# Initialize the Google Gemini-2.0-Flash model
llm_google = ChatGoogleGenerativeAI(
    model="gemini-2.0-flash",
    temperature=0,
    max_retries=2,
    timeout= 120,
    google_api_key= google_api_key
)
```

3 Define system prompt and cases

- The system prompt mimics an clinic pharmacist specializing in MTM.
- Each case has two questions (see case background)

In [ ]:
```python
# Define the system prompt
system_prompt_template = PromptTemplate.from_template("""
You are an experienced clinical pharmacist specializing in Medication The
Provide a clear, honest, and well-structured response in a single paragra
This is a hypothetical case for educational and testing purposes only.
Case Scenario: {scenario}
""")
# Prompt engineering strategies may vary among different models.
```

Case 1

This is a relatively simple case.

In [ ]:
```python
# Define the case scenario
case1 = """
Scenario:
Mrs. Johnson is a 65-year-old woman recently diagnosed with hypertension.
She has been prescribed Lisinopril 10 mg once daily.
She reports feeling well and is compliant with her medication.
During her routine check-up, her blood pressure was recorded at 142/88 mm

Medical Assessment:
- **Primary Medical Concerns:**
  - Patient reports feeling well and is compliant with her medication.
  - pressure reading: 142/88 mmHg.
```

```
- **Past Medical History (PMH):**
  - Hypertension.

- **Allergies:**
  - None.

- **Current Medications:**
  - Lisinopril 10 mg once daily.
"""

# Format the system prompt
case1_system_prompt = system_prompt_template.format(scenario=case1)
```

Question 1

In [ ]:
```
# Define the question 1
case1_question1 = "Are there any significant drug-drug interactions?"

# Define the ground truth answer
case1_ground_truth1 = """
No drug-drug interactions
"""
```

Question 2

In [ ]:
```
# Second question
case1_question2 = "Which medication changes should be made based on the p

# Define the ground truth answer from expert
case1_ground_truth2 = """
Increase lisinopril to 20 mg once daily,
since blood pressure is not well controlled a change to the current regim
Increasing the dose of lisinopril should be effective in controlling bloo
"""
```

Case 2

This is a more complex case.

In [ ]:
```
case2 = """
Scenario:
ME is a 41-year-old man diagnosed with HIV approximately 6 years ago.
He is on antiretroviral therapy and was recently diagnosed with diabetes
After starting new medications, he developed significant fatigue, nausea,
He presented to the emergency department with these symptoms approximatel

Medical Assessment
- **Primary Medical Concerns:**
  - Patient presents with significant fatigue and nausea
  - Physical examination reveals purple stretch marks, small bruises
  - Development of moon face and shoulder hump
  - Reports difficulty breathing and wheezing, worsened by physical activ

- **Past Medical History (PMH):**
  - HIV (diagnosed 6 years ago)
  - tes (recently diagnosed)
```

```
  - Hypertension (recently diagnosed)
  - Asthma

- **Allergies:**
  - None

- **Current Medications:**
  - Darunavir/cobicistat (Prezcobix)
  - Dolutegravir (Tivicay)
  - Metformin 500 mg twice daily
  - Lisinopril 5 mg daily
  - Fluticasone
  - Albuterol
"""

case2_system_prompt = system_prompt_template.format(scenario=case2)
```

Question 1

In [ ]:
```
case2_question1 = "Are there any significant drug-drug interactions?"

case2_ground_truth1 = """
-Inhaled corticosteroids (fluticasone) and pharmacokinetic boosters (daru
ME's reaction originated from a cytochrome P450 3A4 interaction between c
The interaction increases cortisol levels within the body while inhibitin
As a result, ME developed Cushing syndrome. Bruising easily, a fatty hump
Patients with HIV are susceptible to a considerable number of drug intera
It is important to review a patient's medical history before initiating n
-Dolutegravir (Tivicay) may increase the serum concentration of Metformin
"""
```

Question 2

In [ ]:
```
case2_question2 = "Which medication changes should be made based on the p

case2_ground_truth2 = """
To avoid this reaction, the preferred alternative to fluticasone is beclo
- This regimen follows asthma guidelines (2020) stating this patient requ
- To manage interaction of Dolutegravir (Tivicay) and Metformin consider

"""
```

4 Response generation and efficiency evaluation

This step generate responses to different questions and compare their efficiencies using the two metrics below:

- **Inference Cost:** The cost associated with running an LLM to generate responses.
- **Response Time:**
  `Response Time` = `end_time` - `start_time`

In [ ]:
```
# Define a class to track the usage of the model
class ModelUsageTracker:
    def __init__(self, price_per_1m_tokens):
```

```
        """
        Initializes a tracker for monitoring API usage.

        Parameters:
        - price_per_1m_tokens: Defines the $ cost per 1 million tokens.
            - Example 1: Separate input/output pricing: {"input": 10, "ou
            - Example 2: A single flat rate for all tokens: 20

        Attributes:
        - start_time: timestamp when the API call starts.
        - end_time: timestamp when the API call ends.
        - prompt_tokens: number of input tokens used.
        - completion_tokens: number of output tokens generated.
        - total_tokens: sum of prompt and completion tokens.
        - total_cost: Cost calculated based on token usage.

        """
        self.price_per_1m_tokens = price_per_1m_tokens
        self.start_time = None
        self.end_time = None
        self.prompt_tokens = 0
        self.completion_tokens = 0
        self.total_tokens = 0
        self.total_cost = 0.0
        self.response_time = 0.0

    def track_usage(self, usage):
        """
        Extracts token usage from API response and calculates the cost.
        """
        if usage:
            # Extract input (prompt) and output (completion) tokens
            self.prompt_tokens = usage.get("prompt_tokens", usage.get("in
            self.completion_tokens = usage.get("completion_tokens", usage
            self.total_tokens = usage.get("total_tokens", self.prompt_tok

            # Calculate the total cost based on pricing
            if isinstance(self.price_per_1m_tokens, dict):  # Separate pr
                input_cost = (self.prompt_tokens / 1_000_000) * self.pric
                output_cost = (self.completion_tokens / 1_000_000) * self
                self.total_cost = input_cost + output_cost
            elif isinstance(self.price_per_1m_tokens, (int, float)):  # F
                self.total_cost = (self.total_tokens / 1_000_000) * self.

    def get_summary(self):
        """
        Return a dictionary with tracked details.
        """
        return {
            "Prompt Tokens": self.prompt_tokens,
            "Completion Tokens": self.completion_tokens,
            "Total Tokens": self.total_tokens,
            "Total Cost (USD)": round(self.total_cost, 6),
            "Response Time (s)": round(self.response_time, 2)
        }

    def display_summary(self):
        """
        Display usage summary in plain text format.
        """
```

```
        print(f"""
------------------------------------------------------------
Model Usage Summary
------------------------------------------------------------
- Prompt Tokens:      {self.prompt_tokens}
- Completion Tokens:  {self.completion_tokens}
- Total Tokens:       {self.total_tokens}
- Total Cost: $       {self.total_cost:.6f}
- Response Time:      {self.response_time:.2f} seconds
""")

# Define a context manager to automatically track API call usage
@contextmanager
def usage_tracker(price_per_1m_tokens):
    """
    Context manager for tracking model usage, response time, and cost.
      - Automatically starts tracking before API call.
      - Stops tracking after API call completes and calculates response t
      - Displays usage summary upon exit.
    """
    tracker = ModelUsageTracker(price_per_1m_tokens)
    tracker.start_time = time.time()  # Record start time before API call
    yield tracker  # Provide the tracker instance
    tracker.end_time = time.time()  # Record end time after API call
    tracker.response_time = tracker.end_time - tracker.start_time # Compu
    # Display usage summary
    tracker.display_summary()
```

Case 1 Question 1

In [ ]:

```
# GPT4o
# Step 1: Initialize a list to store model usage data for analysis
gpt_case1_usages1 = []

# Step 2: Use the `usage_tracker` context manager to track API call usage
with usage_tracker({"input": 2.50, "output": 10.00}) as tracker_gpt:
    """
    Pricing: $2.50 per 1M input tokens, $10.00 per 1M output tokens.
    """

    # Step 3: Define the input messages
    messages = [
        SystemMessage(content=case1_system_prompt),
        HumanMessage(content=case1_question1)
    ]

    # Step 4: Invoke the GPT-4o model
    case1_gpt_response1 = llm_gpt.invoke(messages)

    # Step 5: Display the response in Markdown format
    display(Markdown(case1_gpt_response1.content))

    # Step 6: Track token usage from the response
    tracker_gpt.track_usage(case1_gpt_response1.response_metadata["token_

# Step 7: Capture the final summary and store it in the list
gpt_case1_usages1.append(tracker_gpt.get_summary())
```

In this scenario, Mrs. Johnson is currently taking only Lisinopril 10 mg once daily, and there are no other medications mentioned that could interact with it. Therefore, based on the information provided, there are no drug-drug interactions to consider. However, it is important to be aware that Lisinopril, an ACE inhibitor, can interact with other medications if they are introduced in the future. Common interactions include nonsteroidal anti-inflammatory drugs (NSAIDs), which can reduce the antihypertensive effect of Lisinopril, and potassium supplements or potassium-sparing diuretics, which can increase the risk of hyperkalemia. It is crucial to review any new medications or supplements with a healthcare provider to avoid potential interactions.

```
------------------------------------------------------------
Model Usage Summary
- Prompt Tokens: 231
- Completion Tokens: 153
- Total Tokens: 384
- Total Cost: $0.002107
- Response Time: 3.31 seconds
```

In [ ]:
```python
# Google Gemini
gemini_case1_usages1 = []

with usage_tracker({"input": 0.1, "output": 0.4}) as tracker_gemini: # Cu
    messages = [
        SystemMessage(content=case1_system_prompt),
        HumanMessage(content=case1_question1)
    ]
    case1_gemini_response1 = llm_google.invoke(messages)

    display(Markdown(case1_gemini_response1.content))

    tracker_gemini.track_usage(case1_gemini_response1.usage_metadata)

gemini_case1_usages1.append(tracker_gemini.get_summary())
```

Based on the provided information, the patient is currently only taking Lisinopril. Therefore, there are no drug-drug interactions to assess at this time. However, it is crucial to consider potential interactions if any new medications, including over-the-counter drugs or supplements, are added to her regimen in the future.

```
------------------------------------------------------------
Model Usage Summary
- Prompt Tokens: 245
- Completion Tokens: 66
- Total Tokens: 311
- Total Cost: $0.000051
- Response Time: 0.92 seconds
```

Case 1 Question 2

In [ ]:
```python
# GPT4o
display("GPT4o response:")

gpt_case1_usages2 = []

e_tracker({"input": 2.50, "output": 10.00}) as tracker_gpt:
```

```
    messages = [
        SystemMessage(content=case1_system_prompt),
        HumanMessage(content=case1_question2)
    ]

    case1_gpt_response2 = llm_gpt.invoke(messages)

    display(Markdown(case1_gpt_response2.content))

    tracker_gpt.track_usage(case1_gpt_response2.response_metadata["token_

gpt_case1_usages2.append(tracker_gpt.get_summary())

display("--------------------------------")

# Gemini
display("Gemini response:")

gemini_case1_usages2 = []

with usage_tracker({"input": 0.1, "output": 0.4}) as tracker_gemini:

    messages = [
        SystemMessage(content=case1_system_prompt),
        HumanMessage(content=case1_question2)
    ]
    case1_gemini_response2 = llm_google.invoke(messages)

    display(Markdown(case1_gemini_response2.content))

    tracker_gemini.track_usage(case1_gemini_response2.usage_metadata)

gemini_case1_usages2.append(tracker_gemini.get_summary())
```

```
'GPT4o response:'
```

In assessing Mrs. Johnson's current condition, her blood pressure reading of 142/88 mmHg indicates that while her systolic pressure is slightly above the target range for most patients with hypertension, her diastolic pressure is within an acceptable range. Given that she is compliant with her Lisinopril 10 mg once daily and reports feeling well, it may be beneficial to consider a slight adjustment to her medication regimen to achieve better blood pressure control. One option could be to increase the dose of Lisinopril to 20 mg once daily, as this is a common next step in managing hypertension when the initial dose is insufficient. However, before making any changes, it is important to evaluate her overall cardiovascular risk, kidney function, and any potential side effects she may experience with a higher dose. Additionally, lifestyle modifications such as dietary changes, increased physical activity, and weight management should be reinforced as part of her comprehensive hypertension management plan. It is crucial to monitor her blood pressure closely following any medication adjustment to ensure efficacy and safety.

```
------------------------------------------------------------
Model Usage Summary
- Prompt Tokens: 235
- Completion Tokens: 209
- Total Tokens: 444
- Total Cost: $0.002678
- Response Time: 7.22 seconds

'------------------------------'
'Gemini response:'
```

Given Mrs. Johnson's blood pressure reading of 142/88 mmHg despite being compliant with Lisinopril 10mg daily, her hypertension is not currently controlled. As a first step, I would consider increasing the Lisinopril dosage, typically titrating upwards to a maximum of 40mg daily, while closely monitoring her blood pressure and for any signs of hypotension or other side effects like cough or angioedema. If the blood pressure remains uncontrolled with the maximum dose of Lisinopril, or if she experiences intolerable side effects, adding a second antihypertensive medication from a different class, such as a thiazide diuretic (e.g., hydrochlorothiazide) or a calcium channel blocker (e.g., amlodipine), would be the next appropriate step, while also reinforcing lifestyle modifications like diet and exercise.

```
------------------------------------------------------------
Model Usage Summary
- Prompt Tokens: 251
- Completion Tokens: 179
- Total Tokens: 430
- Total Cost: $0.000097
- Response Time: 2.25 seconds
```

Case 2 Question 1

In [ ]:

```python
# GPT4o
display("GPT4o response:")

gpt_case2_usages1 = []

with usage_tracker({"input": 2.50, "output": 10.00}) as tracker_gpt:
    messages = [
        SystemMessage(content=case2_system_prompt),
        HumanMessage(content=case2_question1)
    ]
    case2_gpt_response1 = llm_gpt.invoke(messages)

    display(Markdown(case2_gpt_response1.content))

    tracker_gpt.track_usage(case2_gpt_response1.response_metadata["token_

gpt_case2_usages1.append(tracker_gpt.get_summary())

display("------------------------------------------------------------

# Gemini
display("Gemini response:")

gemini_case2_usages1 = []
```

```python
with usage_tracker({"input": 0.1, "output": 0.4}) as tracker_gemini:
    messages = [
        SystemMessage(content=case2_system_prompt),
        HumanMessage(content=case2_question1)
    ]

    case2_gemini_response1 = llm_google.invoke(messages)

    display(Markdown(case2_gemini_response1.content))

    tracker_gemini.track_usage(case2_gemini_response1.usage_metadata)

gemini_case2_usages1.append(tracker_gemini.get_summary())
```

```
'GPT4o response:'
```

In this scenario, the patient is experiencing symptoms that may be indicative of drug-drug interactions or side effects from his current medication regimen. The combination of darunavir/cobicistat (Prezcobix) and dolutegravir (Tivicay) is generally well-tolerated, but cobicistat, a pharmacokinetic enhancer, can increase the levels of other drugs metabolized by the liver, potentially affecting the metabolism of other medications. Cobicistat can inhibit CYP3A4, which may lead to increased levels of fluticasone, a corticosteroid, potentially causing Cushing's syndrome-like symptoms such as moon face, shoulder hump, and purple stretch marks. This could also explain the patient's fatigue, nausea, and bruising. Additionally, the use of fluticasone with a CYP3A4 inhibitor like cobicistat can exacerbate asthma symptoms due to systemic corticosteroid effects. Metformin and lisinopril are not typically affected by cobicistat, but the patient's new onset of diabetes and hypertension requires careful monitoring. The patient's symptoms warrant a review of his medication regimen, particularly the use of fluticasone, and consideration of alternative asthma management strategies to avoid systemic corticosteroid exposure.

```
------------------------------------------------------------
Model Usage Summary
- Prompt Tokens: 327
- Completion Tokens: 248
- Total Tokens: 575
- Total Cost: $0.003298
- Response Time: 4.76 seconds

'--------------------------------'
'Gemini response:'
```

Given the patient's presentation of fatigue, nausea, purple stretch marks, easy bruising, moon face, shoulder hump, difficulty breathing, and wheezing, in the context of newly diagnosed diabetes and hypertension, and recent initiation of multiple medications including Darunavir/cobicistat, Dolutegravir, Metformin, Lisinopril, Fluticasone, and Albuterol, a strong suspicion for drug-induced Cushing's syndrome arises, most likely from the inhaled fluticasone potentiated by the cobicistat component of Prezcobix. Cobicistat is a strong CYP3A4 inhibitor, which can significantly increase the systemic absorption and bioavailability of fluticasone, leading to iatrogenic Cushing's syndrome. While other drug interactions are possible, such as potential interactions between antiretrovirals and metformin, the constellation of symptoms strongly points towards the fluticasone/cobicistat interaction as the primary concern. The patient's asthma treatment should be re-evaluated, and alternative asthma medications that do not have significant interactions with cobicistat should be considered.

```
------------------------------------------------------------
Model Usage Summary
- Prompt Tokens: 331
- Completion Tokens: 220
- Total Tokens: 551
- Total Cost: $0.000121
- Response Time: 2.24 seconds
```

Case 2 Question 2

In [ ]:

```
# GPT4o
display("GPT4o response:")

gpt_case2_usages2 = []

with usage_tracker({"input": 2.50, "output": 10.00}) as tracker_gpt:
    messages = [
        SystemMessage(content=case2_system_prompt),
        HumanMessage(content=case2_question2)
    ]

    case2_gpt_response2 = llm_gpt.invoke(messages)

    display(Markdown(case2_gpt_response2.content))

    tracker_gpt.track_usage(case2_gpt_response2.response_metadata["token_

gpt_case2_usages2.append(tracker_gpt.get_summary())

display("------------------------------------------------------------------

# Gemini
display("Gemini response:")

gemini_case2_usages2 = []

with usage_tracker({"input": 0.1, "output": 0.4}) as tracker_gemini:
    messages = [
        SystemMessage(content=case2_system_prompt),
        HumanMessage(content=case2_question2)
```

```
    case2_gemini_response2 = llm_google.invoke(messages)

    display(Markdown(case2_gemini_response2.content))

    tracker_gemini.track_usage(case2_gemini_response2.usage_metadata)

gemini_case2_usages2.append(tracker_gemini.get_summary())
```

```
'GPT4o response:'
```

Based on the patient's presentation and current medication regimen, it is likely that the patient is experiencing symptoms consistent with Cushing's syndrome, potentially due to an interaction between his HIV medications and fluticasone, a corticosteroid. The combination of darunavir/cobicistat (a protease inhibitor and a CYP3A inhibitor) with inhaled fluticasone can lead to increased systemic corticosteroid levels, causing the physical changes and symptoms observed. To address this, it would be prudent to discontinue fluticasone and consider alternative asthma management strategies that do not involve corticosteroids, such as leukotriene receptor antagonists or adjusting the use of albuterol. Additionally, the patient's fatigue and nausea could be related to the new onset of diabetes and hypertension, or side effects from metformin or lisinopril, so monitoring and adjusting these medications as needed, based on blood glucose and blood pressure control, would be advisable. Close follow-up is necessary to reassess symptoms and ensure effective management of all conditions.

```
------------------------------------------------------------
Model Usage Summary
- Prompt Tokens: 331
- Completion Tokens: 204
- Total Tokens: 535
- Total Cost: $0.002868
- Response Time: 4.90 seconds

'--------------------------------'
'Gemini response:'
```

Based on the patient's presentation of fatigue, nausea, purple striae, easy bruising, moon face, shoulder hump, difficulty breathing, and wheezing, along with his history of HIV, diabetes, hypertension, and asthma, the most likely diagnosis is iatrogenic Cushing's syndrome induced by the fluticasone inhaler. The priority is to immediately discontinue the fluticasone inhaler and switch to an alternative asthma management strategy, such as a leukotriene receptor antagonist or increasing the frequency of albuterol use as needed, while closely monitoring his respiratory status. Given the recent diagnoses of diabetes and hypertension, it's also important to re-evaluate the necessity and dosages of metformin and lisinopril, considering potential interactions with the antiretroviral regimen and the impact of Cushing's syndrome on glucose and blood pressure control. Furthermore, assessing cortisol levels would help confirm the diagnosis of Cushing's syndrome.

```
------------------------------------------------------------
Model Usage Summary
- Prompt Tokens: 337
- Completion Tokens: 187
- Total Tokens: 524
- Total Cost: $0.000109
- Response Time: 2.16 seconds
```

5 Semantic similarity evaluation

This section introduces a metric for measuring semantic similarity.

More specifically, we use:

- **Cosine Similarity**: measures semantic similarity between model responses and reference answers by comparing their vector representations in embedding space.
- It is very sensitive to contextual differences in the text and does not evaluate factual correctness.

Its calculation has two steps:

1. Covert reference answers and generated responses into vector embeddings using an LLM's embedding model.
2. Compute `cosine similarity`. Closer to 1 means higher similarity.

Check other similarity metrics available in LangChain Evaluation Chain class `EmbeddingDistance`.

In [ ]:
```python
# Initialize the embedding model
embedding_model = OpenAIEmbeddings(model="text-embedding-3-large")
```

Case 1 Question 1

In [ ]:
```python
# Calculate the cosine similarity between the ground truth and the model'

# Step 1: Initialize the evaluator for cosine similarity measurement
cosine_evaluator = load_evaluator(
    "embedding_distance", # Use embedding-based distance evaluation
    embeddings=embedding_model, # the embedding model initialized in the
    distance_metric="cosine"  # Specifies cosine distance as the metric (
)

# Step 2: Compute cosine similarity for GPT-4o model
gpt_case1_cosine_similarity1 = 1 - cosine_evaluator.evaluate_strings(
    prediction=case1_gpt_response1.content, # Generated response
    reference=case1_ground_truth1           # Ground truth answer
    )['score']

# Step 3: Compute cosine similarity for Gemini 2.0 Flash model
gemini_case1_cosine_similarity1 = 1 - cosine_evaluator.evaluate_strings(
    prediction=case1_gemini_response1.content,
    reference=case1_ground_truth1
```

```
    )['score']


# Step 4: Store results in a DataFrame for easy visualization
case1_cosine_similarities1 = pd.DataFrame({
    "Model": ["GPT-4o", "Gemini 2.0 Flash"],
    "Cosine Similarity": [gpt_case1_cosine_similarity1, gemini_case1_cosi
})

# Step 5: Display the results
display(case1_cosine_similarities1)
```

	Model	Cosine Similarity
0	GPT-4o	0.442333
1	Gemini 2.0 Flash	0.510733

Case 1 Question 2

In [ ]:
```
gpt_case1_cosine_similarity2 = 1 - cosine_evaluator.evaluate_strings(
    prediction=case1_gpt_response2.content,
    reference=case1_ground_truth2
    )['score']

gemini_case1_cosine_similarity2 = 1 - cosine_evaluator.evaluate_strings(
    prediction=case1_gemini_response2.content,
    reference=case1_ground_truth2
    )['score']

# Display the cosine similarity
case1_cosine_similarities2 = pd.DataFrame({
    "Model": ["GPT-4o", "Gemini 2.0 Flash"],
    "Cosine Similarity": [gpt_case1_cosine_similarity2, gemini_case1_cosi
})

display(case1_cosine_similarities2)
```

	Model	Cosine Similarity
0	GPT-4o	0.701854
1	Gemini 2.0 Flash	0.697518

Case 2 Question 1

In [ ]:
```
gpt_case2_cosine_similarity1 = 1 - cosine_evaluator.evaluate_strings(
    prediction=case2_gpt_response1.content,
    reference=case2_ground_truth1
    )['score']

gemini_case2_cosine_similarity1 = 1 - cosine_evaluator.evaluate_strings(
    prediction=case2_gemini_response1.content,
    reference=case2_ground_truth1
    )['score']

# Display the cosine similarity
```

```
case2_cosine_similarities1 = pd.DataFrame({
    "Model": ["GPT-4o", "Gemini 2.0 Flash"],
    "Cosine Similarity": [gpt_case2_cosine_similarity1, gemini_case2_cosi
})

display(case2_cosine_similarities1)
```

	Model	Cosine Similarity
0	GPT-4o	0.781913
1	Gemini 2.0 Flash	0.763158

Case 2 Question 2

In [ ]:
```
gpt_case2_cosine_similarity2 = 1 - cosine_evaluator.evaluate_strings(
    prediction=case2_gpt_response2.content,
    reference=case2_ground_truth2
    )['score']

gemini_case2_cosine_similarity2 = 1 - cosine_evaluator.evaluate_strings(
    prediction=case2_gemini_response2.content,
    reference=case2_ground_truth2
    )['score']

case2_cosine_similarities2 = pd.DataFrame({
    "Model": ["GPT-4o", "Gemini 2.0 Flash"],
    "Cosine Similarity": [gpt_case2_cosine_similarity2, gemini_case2_cosi
})

display(case2_cosine_similarities2)
```

6 AI as a judge

This section leverage AI as a judge to evaluate responses based on predefined criteria in LangChain.

How to prompt an AI judge is similar to how to prompt any AI application. We will experiment the following four criteria based on LangChain's built-in prompt:

- **Correctness**: Is the submission correct, accurate, and factual?
- **Relevance**: Is the submission referring to a real quote from the text?
- **Coherence**: Is the submission coherent, well-structured, and organized?
- **Conciseness**: Is the submission concise and to the point?.

How to prompt an AI judge is similar to how to prompt any AI application.

For a complete list of evaluation criteria, refer to the LangChain Criteria class and click "Source Code" to view each criterion's promot.

In [ ]:
```
# Step 1: Combine the system prompt (scenario) and question into a single
# `strip() function removes any leading/trailing spaces from `case1_syste
input_query = f"{case1_system_prompt.strip()}\n\nQuestion:\n{case1_questi
```

```
# Step 2: Define custom evaluation criteria based on LangChain's predefin

custom_criteria = {
"CORRECTNESS": "Is the submission correct, accurate, and factual?",
"RELEVANCE": "Does the response directly address the question based on th
"COHERENCE": "Is the submission coherent, well-structured, and organized?
"CONCISENESS": "Is the submission concise and to the point?"
}
```

Case 1 Question 1

In [ ]:
```
# GPT4o

# Step 1: Initialize a list to store evaluation results
gpt_case1_evaluation_results1 = []


# Step 2: Loop through each evaluation criterion and assess the response
for criterion, description in custom_criteria.items():
    """
    - `criterion`: the name of the evaluation metric (e.g., CORRECTNESS).
    - `description`: A brief definition of the criterion, explaining what
    """

    # Load an evaluator for the given criterion
    evaluator = load_evaluator(
        "labeled_criteria",  # Use LangChain's labeled criteria evaluator
        criteria={criterion: description}, # Pass the evaluation metric a
        llm=llm_deepseek # Use the DeepSeek model for evaluation
        )

    # Step 3: Evaluate the response against the ground truth
    result = evaluator.evaluate_strings(
        prediction=case1_gpt_response1.content, # Model-generated respons
        input=input_query,  # The full prompt including scenario and ques
        reference=case1_ground_truth1 # The ground-truth answer
    )

    # Step 4: Append the results to the list
    gpt_case1_evaluation_results1.append({
        "Criterion": criterion,
        "Score": result["score"], # Numeric score assigned by the evaluat
        "Reasoning": result.get("reasoning", "No reasoning provided") # E
    })

# Step 5: Convert the results into a pandas DataFrame
gpt_case1_evaluation_results1 = pd.DataFrame(gpt_case1_evaluation_results
```

In [ ]:
```
print(tabulate(gpt_case1_evaluation_results1,
               headers='keys',
               tablefmt='fancy_grid'))
```

	Criterion	Score	Reasoning
0	CORRECTNESS	1	1. **Understanding the Scenario**: Mrs. Johnson is taking Lisinopril 10 mg once daily for hypertension. She is compliant with her medication and has no other medications listed. Her blood pressure is slightly elevated at 142/88 mmHg. 2. **Identifying the Question**: The question asks about potential drug-drug interactions in this scenario. 3. **Analyzing the Submission**: The submission correctly states that there are no drug-drug interactions in this specific scenario because Mrs. Johnson is only taking Lisinopril. It also provides additional information about potential interactions if other medications were introduced, such as NSAIDs or potassium supplements. 4. **Assessing Correctness**: The submission is accurate and factual. It correctly identifies that there are no current drug-drug interactions based on the information provided. The additional information about potential future interactions is also correct and relevant. 5. **Comparing with Reference**: The reference confirms that there are no drug-drug interactions in this scenario, which aligns with the submission. 6. **Conclusion**: The submission meets the criteria for correctness. Y

1	RELEVANCE	1	1. **Understanding the Question**: The question asks about drug-drug interactions in the context of Mrs. Johnson's current medication, Lisinopril 10 mg once daily. The scenario does not mention any other medications or supplements she is taking. 2. **Analyzing the Submission**: The submission correctly identifies that there are no drug-drug interactions in the current scenario since Mrs. Johnson is only taking Lisinopril. It also provides additional educational information about potential interactions if other medications were introduced in the future, such as NSAIDs or potassium supplements. 3. **Assessing Relevance**: The response directly addresses the question by stating that there are no drug-drug interactions based on the provided scenario. The additional information about potential future interactions, while useful, does not detract from the relevance of the response to the specific question asked. 4. **Conclusion**: The submission meets the relevance criterion as it directly answers the question based on the given scenario. Y
2	COHERENCE	1	1. **Coherence**: The submission is coherent and well-structured. It begins by directly addressing the question about drug-drug interactions, stating that there are none based on the information provided. It then provides additional relevant information about potential interactions with Lisinopril if other medications are introduced in the future. This structure ensures that the response is clear and logically organized. 2. **Well-structured**: The response is well-structured, starting with a clear answer to the question and then expanding on related considerations. The use of specific examples (e.g., NSAIDs, potassium supplements) enhances the clarity and usefulness of the response.

| | | | 3. **Organized**: The submission is organized in a logical flow. It first addresses the immediate question, then provides additional context and advice, which is helpful for understanding the broader implications of the scenario.
| | | | |
| | | | Based on this reasoning, the submission meets the criteria for coherence, well-structured, and organized.
| | | | |
			Y
3	CONCISENESS	0	1. **Understanding the Criteria**: The criterion is "CONCISENESS," which means the submission should be brief and to the point, avoiding unnecessary details or elaboration.
			2. **Analyzing the Submission**: The submission starts by correctly stating that there are no drug-drug interactions in the given scenario, which is concise. However, it then proceeds to discuss potential future interactions with Lisinopril, including NSAIDs and potassium supplements, which are not relevant to the current scenario. This additional information, while accurate, is not necessary for answering the question based on the provided data.
			3. **Assessing Conciseness**: The submission could be more concise by omitting the discussion of potential future interactions, as they are not pertinent to the current case. The focus should remain solely on the absence of drug-drug interactions in the present scenario.
			4. **Conclusion**: The submission does not fully meet the criterion of conciseness because it includes extraneous information that is not required to answer the question based on the given data.
			N

In [ ]:

```
# Gemini

gemini_case1_evaluation_results1 = []

for criterion, description in custom_criteria.items():
    evaluator = load_evaluator(
        "labeled_criteria",
        criteria={criterion: description},
        llm=llm_deepseek
        )

    result = evaluator.evaluate_strings(
        prediction=case1_gemini_response1.content,
        input=input_query,
        reference=case1_ground_truth1
    )

    gemini_case1_evaluation_results1.append({
        "Criterion": criterion,
        "Score": result["score"],
        "Reasoning": result.get("reasoning", "No reasoning provided")
    })

gemini_case1_evaluation_results1 = pd.DataFrame(gemini_case1_evaluation_r

print(tabulate(gemini_case1_evaluation_results1,
               headers='keys',
               tablefmt='fancy_grid'))
```

	Criterion	Score	Reasoning
0	CORRECTNESS	1	1. **Understanding the Scenario**: Mrs. Johnson is a 65-year-old woman with hypertension, currently taking Lisinopril 10 mg once daily. Her blood pressure is 142/88 mmHg, and she reports feeling well and being compliant with her medication. The question asks about drug-drug interactions. 2. **Analyzing the Submission**: The submission states that there are no drug-drug interactions to assess because the patient is only taking Lisinopril. It also mentions the importance of considering potential interactions if new medications are added in the future. 3. **Assessing Correctness**: The submission correctly identifies that there are no drug-drug interactions in the current scenario since the patient is only on Lisinopril. This aligns with the reference, which also indicates no drug-drug interactions. The additional advice about future considerations is prudent but not directly relevant to the current question. 4. **Conclusion**: The submission is correct, accurate, and factual based on the provided information and reference. Y
1	RELEVANCE	1	Let's evaluate the submission step by step based on the RELEVANCE criterion: 1. **Understanding the Question**: The question asks about drug-drug interactions in the context of Mrs. Johnson's current medications and scenario. 2. **Analyzing the Submission**: The submis

sion correctly identifies that Mrs. Johnson is only taking Lisinopril and states that there are no drug-drug interactions to assess at this time. It also appropriately advises considering potential interactions if new medications are added in the future.

| | | | 3. **Relevance to the Scenario**: The response directly addresses the question by focusing on the absence of drug-drug interactions given the current medication regimen. It also provides a relevant and practical consideration for future scenarios.

| | | | 4. **Alignment with the Reference**: The reference confirms that there are no drug-drug interactions, which aligns with the submission's conclusion.

| | | |

| | | | Since the submission directly and accurately addresses the question based on the provided scenario, it meets the RELEVANCE criterion.

| | | |

| | | | Y

|---|---|---|---|

| 2 | COHERENCE | 1 | 1. **Coherence**: The submission is coherent as it directly addresses the question about drug-drug interactions. It clearly states that there are no interactions to assess since the patient is only taking Lisinopril. This is a logical and straightforward response to the given scenario.

| | | | 2. **Well-Structured**: The submission is well-structured. It begins by addressing the current situation (no interactions due to only one medication) and then provides a forward-looking statement about the importance of considering potential interactions if new medications are introduced. This structure is clear and easy to follow.

| | | | 3. **Organized**: The submission is organized in a logical manner. It first answers the immediate question and then adds a relevant consideration for the future, which helps to provide a complete response without unnecessary information.

| | | | Based on this step-by-step analysis, the submission meets the criteria for coherence, well-structured, and organized.

| | | | Y

```
├─────┼──────────────┼──────────┼───────────────────────────────────────────
────────────────────────────────────────────────────────────────────────────
────────────────────────────────────────────────────────────────────────────
────────────────────────────────────────────────────────────────────────────
────────────────────────────────────────────────────────────────────────────
─────────────────────────┤
│  3 │ CONCISENESS │        1 │ 1. **Understanding the Criteria**: The crit
erion to evaluate is "CONCISENESS," which means the submission should be b
rief and to the point without unnecessary details or repetition.
│
│    │             │          │
│
│    │             │          │ 2. **Analyzing the Submission**: The submis
sion states that there are no drug-drug interactions because the patient i
s only taking Lisinopril. It also mentions the importance of considering p
otential interactions if new medications are added in the future.
│
│    │             │          │
│
│    │             │          │ 3. **Evaluating Conciseness**: The submissi
on is concise. It directly addresses the question by stating there are no
current drug-drug interactions and briefly mentions a consideration for th
e future. There is no unnecessary elaboration or repetition.
│
│    │             │          │
│
│    │             │          │ 4. **Conclusion**: The submission meets the
criterion of conciseness.
│
│    │             │          │
│
│    │             │          │ Y
│
╘═════╧══════════════╧══════════╧═══════════════════════════════════════════
════════════════════════════════════════════════════════════════════════════
════════════════════════════════════════════════════════════════════════════
════════════════════════════════════════════════════════════════════════════
════════════════════════════════════════════════════════════════════════════
═════════════════════════╛
```

Case 1 Question 2

In [ ]:
```
input_query = f"{case1_system_prompt.strip()}\n\nQuestion:\n{case1_questi

# GPT4o
gpt_case1_evaluation_results2 = []

# Load Evaluators and Run Evaluations
for criterion, description in custom_criteria.items():
    evaluator = load_evaluator(
        "labeled_criteria",
        criteria={criterion: description},
        llm=llm_deepseek
        )

    result = evaluator.evaluate_strings(
        prediction=case1_gpt_response2.content,
        input=input_query,
        reference=case1_ground_truth2
```

Loading [MathJax]/extensions/Safe.js

```
    )

    gpt_case1_evaluation_results2.append({
        "Criterion": criterion,
        "Score": result["score"],
        "Reasoning": result.get("reasoning", "No reasoning provided")
    })

gpt_case1_evaluation_results2 = pd.DataFrame(gpt_case1_evaluation_results

# Gemini
gemini_case1_evaluation_results2 = []

# Load Evaluators and Run Evaluations
for criterion, description in custom_criteria.items():
    evaluator = load_evaluator(
        "labeled_criteria",
        criteria={criterion: description},
        llm=llm_deepseek)

    result = evaluator.evaluate_strings(
        prediction=case1_gemini_response2.content,
        input=input_query,
        reference=case1_ground_truth2
    )

    gemini_case1_evaluation_results2.append({
        "Criterion": criterion,
        "Score": result["score"],
        "Reasoning": result.get("reasoning", "No reasoning provided")
    })

gemini_case1_evaluation_results2 = pd.DataFrame(gemini_case1_evaluation_r
```

Case 2 Question 1

In [ ]:

```
input_query = f"{case2_system_prompt.strip()}\n\nQuestion:\n{case2_questi

# GPT4o
gpt_case2_evaluation_results1 = []

for criterion, description in custom_criteria.items():
    evaluator = load_evaluator(
        "labeled_criteria",
        criteria={criterion: description},
        llm=llm_deepseek
        )

    result = evaluator.evaluate_strings(
        prediction=case2_gpt_response1.content,
        input=input_query,
        reference=case2_ground_truth1
    )

    gpt_case2_evaluation_results1.append({
        "Criterion": criterion,
        "Score": result["score"],
        "Reasoning": result.get("reasoning", "No reasoning provided")
```

```
gpt_case2_evaluation_results1 = pd.DataFrame(gpt_case2_evaluation_results

# Gemini
gemini_case2_evaluation_results1 = []

# Load Evaluators and Run Evaluations
for criterion, description in custom_criteria.items():

    evaluator = load_evaluator(
        "labeled_criteria",
        criteria={criterion: description},
        llm=llm_deepseek
        )

    result = evaluator.evaluate_strings(
        prediction=case2_gemini_response1.content,
        input=input_query,
        reference=case2_ground_truth1
    )

    gemini_case2_evaluation_results1.append({
        "Criterion": criterion,
        "Score": result["score"],
        "Reasoning": result.get("reasoning", "No reasoning provided")
    })

gemini_case2_evaluation_results1 = pd.DataFrame(gemini_case2_evaluation_r
```

Case 2 Question 2

In [ ]:
```
input_query = f"{case2_system_prompt.strip()}\n\nQuestion:\n{case2_questi

# GPT4o
gpt_case2_evaluation_results2 = []

# Load Evaluators and Run Evaluations
for criterion, description in custom_criteria.items():
    evaluator = load_evaluator(
        "labeled_criteria",
        criteria={criterion: description},
        llm=llm_deepseek
        )

    result = evaluator.evaluate_strings(
        prediction=case2_gpt_response2.content,
        input=input_query,
        reference=case2_ground_truth2
    )

    gpt_case2_evaluation_results2.append({
        "Criterion": criterion,
        "Score": result["score"],
        "Reasoning": result.get("reasoning", "No reasoning provided")
    })

gpt_case2_evaluation_results2 = pd.DataFrame(gpt_case2_evaluation_results
```

```
gemini_case2_evaluation_results2 = []

for criterion, description in custom_criteria.items():
    evaluator = load_evaluator(
        "labeled_criteria",
        criteria={criterion: description},
        llm=llm_deepseek
        )

    result = evaluator.evaluate_strings(
        prediction=case2_gemini_response2.content,
        input=input_query,
        reference=case2_ground_truth2
    )

    gemini_case2_evaluation_results2.append({
        "Criterion": criterion,
        "Score": result["score"],
        "Reasoning": result.get("reasoning", "No reasoning provided")
    })

gemini_case2_evaluation_results2 = pd.DataFrame(gemini_case2_evaluation_r
```

7 NLP-based Metric

In this section, we explore three widely used NLP-based metrics for evaluation. These metrics provide objective, quantitative assessments of how closely a model's response aligns with the ground truth.

- **F1 Score**: A balance between precision and recall, measuring the exact token matches between the prediction and ground truth.
- **ROUGE-L**: Evaluates longest common subsequence overlap between the model's answer and the ground truth. Focuses on recall to assess how much of the ground truth is covered by the model's answer.
- **BLEU**: Measures N-gram overlap between the model's answer and the ground truth. Focuses on precision to determine how many n-grams in the model's answer match the ground truth.

In [ ]:
```
# Import necessary libraries
from sklearn.metrics import f1_score
from rouge_score import rouge_scorer
from nltk.translate.bleu_score import sentence_bleu, SmoothingFunction

# F1 Score
def compute_f1(pred, truth):
    """
    Computes the F1 Score between the model's prediction and the ground t

    Parameters:
    - pred (str): Model-generated response.
    - truth (str): Ground-truth answer.

    Returns:
    - F1 score (between 0 and 1).
    """
```

```
    # Step 1: Tokenize prediction and ground truth into word lists
    truth_tokens = truth.split()
    pred_tokens = pred.split()

    # Step 2: Create a set of all unique tokens from both lists
    all_tokens = list(set(truth_tokens + pred_tokens))

    # Step 3: Structure binary vectors based on token's presence
    truth_vec = [1 if token in truth_tokens else 0 for token in all_token
    pred_vec = [1 if token in pred_tokens else 0 for token in all_tokens]

    # Step 4: Compute F1 Score
    return f1_score(truth_vec, pred_vec, average="binary")

# ROUGE
def compute_rouge(pred, truth):
    """
    Parameters:
    - pred (str): Model-generated response.
    - truth (str): Ground-truth reference answer.

    Returns:
    - float: ROUGE-L F-measure score (between 0 and 1).
    """

    # Step 1: Initialize ROUGE scorer with stemming enabled
    scorer = rouge_scorer.RougeScorer(['rougeL'], use_stemmer=True)

    # Step 2: Compute ROUGE-L F1 score
    scores = scorer.score(truth, pred)
    return scores['rougeL'].fmeasure

# BLEU
def compute_bleu(pred, truth):
    """
    Parameters:
    - pred (str): Model-generated response.
    - truth (str): Ground-truth reference answer.

    Returns:
    - float: BLEU score (between 0 and 1).
    """

    # Step 1: Convert reference and candidate sentences into tokenized li
    reference = [truth.split()]
    candidate = pred.split()

    # Step 2: Apply smoothing for short sequences
    smooth = SmoothingFunction().method1  # Handles short answers better

    # Step 3: Compute BLEU score
    return sentence_bleu(reference, candidate, smoothing_function=smooth)
```

Case 1 Question 1

In [ ]:

```
# gpt4o
gpt_case1_nlp_evaluation_results1 = {
    "F1 Score": round(compute_f1(case1_gpt_response1.content, case1_groun
    "ROUGE-L": round(compute_rouge(case1_gpt_response1.content, case1_gro
```

```
    "BLEU": round(compute_bleu(case1_gpt_response1.content, case1_ground_
}



# Gemini
gemini_case1_nlp_evaluation_results1 = {
    "F1 Score": round(compute_f1(case1_gemini_response1.content, case1_gr
    "ROUGE-L": round(compute_rouge(case1_gemini_response1.content, case1_
    "BLEU": round(compute_bleu(case1_gemini_response1.content, case1_grou
}


print("GPT Evaluation Results:")
print(gpt_case1_nlp_evaluation_results1)

print("\nGemini Evaluation Results:")
print(gemini_case1_nlp_evaluation_results1)
```

```
GPT Evaluation Results:
{'F1 Score': 0.0476, 'ROUGE-L': 0.069, 'BLEU': 0.0035}

Gemini Evaluation Results:
{'F1 Score': 0.0909, 'ROUGE-L': 0.1455, 'BLEU': 0.0081}
```

Case 1 Question 2

In [ ]:
```
# gpt4o
gpt_case1_nlp_evaluation_results2 = {
    "F1 Score": round(compute_f1(case1_gpt_response2.content, case1_groun
    "ROUGE-L": round(compute_rouge(case1_gpt_response2.content, case1_gro
    "BLEU": round(compute_bleu(case1_gpt_response2.content, case1_ground_
}

# Gemini
gemini_case1_nlp_evaluation_results2 = {
    "F1 Score": round(compute_f1(case1_gemini_response2.content, case1_gr
    "ROUGE-L": round(compute_rouge(case1_gemini_response2.content, case1_
    "BLEU": round(compute_bleu(case1_gemini_response2.content, case1_grou
}

print("GPT Evaluation Results:")
print(gpt_case1_nlp_evaluation_results2)

print("\nGemini Evaluation Results:")
print(gemini_case1_nlp_evaluation_results2)
```

```
GPT Evaluation Results:
{'F1 Score': 0.2025, 'ROUGE-L': 0.1455, 'BLEU': 0.0107}

Gemini Evaluation Results:
{'F1 Score': 0.1486, 'ROUGE-L': 0.1478, 'BLEU': 0.0051}
```

Case 2 Question 1

In [ ]:
```
# gpt4o
gpt_case2_nlp_evaluation_results1 = {
    "F1 Score": round(compute_f1(case2_gpt_response1.content, case2_groun
    "ROUGE-L": round(compute_rouge(case2_gpt_response1.content, case2_gro
```

```
    "BLEU": round(compute_bleu(case2_gpt_response1.content, case2_ground_
}


# Gemini
gemini_case2_nlp_evaluation_results1 = {
    "F1 Score": round(compute_f1(case2_gemini_response1.content, case2_gr
    "ROUGE-L": round(compute_rouge(case2_gemini_response1.content, case2_
    "BLEU": round(compute_bleu(case2_gemini_response1.content, case2_grou
}


print("GPT Evaluation Results:")
print(gpt_case2_nlp_evaluation_results1)

print("\nGemini Evaluation Results:")
print(gemini_case2_nlp_evaluation_results1)
```

```
GPT Evaluation Results:
{'F1 Score': 0.2079, 'ROUGE-L': 0.1908, 'BLEU': 0.0047}

Gemini Evaluation Results:
{'F1 Score': 0.2073, 'ROUGE-L': 0.1725, 'BLEU': 0.0064}
```

Case 2 Question 2

In [ ]:
```
# gpt4o
gpt_case2_nlp_evaluation_results2 = {
    "F1 Score": round(compute_f1(case2_gpt_response2.content, case2_groun
    "ROUGE-L": round(compute_rouge(case2_gpt_response2.content, case2_gro
    "BLEU": round(compute_bleu(case2_gpt_response2.content, case2_ground_
}


# Gemini
gemini_case2_nlp_evaluation_results2 = {
    "F1 Score": round(compute_f1(case2_gemini_response2.content, case2_gr
    "ROUGE-L": round(compute_rouge(case2_gemini_response2.content, case2_
    "BLEU": round(compute_bleu(case2_gemini_response2.content, case2_grou
}
```

8. Comparative Analysis

In this section, we create summaries of the evaluation results across different approaches. You can use the summaries for comparative analysis and determine which foundation model to use for your application.

Note: this lab does not include manual evaluation by human. For high-stakes applications such as Medical AI or legal AI, you may want to have manual evaluation by experts, Comparing expert evaluaion with the automated approaches covered in this lab can help create a gold standard for quality control. However, it is often time-consuming and may introduce subjectivity.Therefore, a hybrid approach, combining automated metrics with expert review, is recommended for critical applications.

Case 1 Question 1

In [ ]:
```
# Extract Total Cost & Response Time for GPT-4o
gpt_case1_total_cost1 = gpt_case1_usages1[0]["Total Cost (USD)"]
gpt_case1_response_time1 = gpt_case1_usages1[0]["Response Time (s)"]

# Extract Total Cost and Response Time for Gemini 2.0 Flash
gemini_case1_total_cost1 = gemini_case1_usages1[0]["Total Cost (USD)"]
gemini_case1_response_time1 = gemini_case1_usages1[0]["Response Time (s)"

# Extract Cosine Similarity Scores
gpt_case1_cosine_similarity1 = case1_cosine_similarities1[case1_cosine_si
gemini_case1_cosine_similarity1 = case1_cosine_similarities1[case1_cosine

# Extract LLM-based Criterion Scores GPT
gpt_case1_criterion_scores1 = {}
for _, row in gpt_case1_evaluation_results1.iterrows():
    gpt_case1_criterion_scores1[row["Criterion"]] = row["Score"]

# Extract Criterion Scores for Gemini
gemini_case1_criterion_scores1 = {}
for _, row in gemini_case1_evaluation_results1.iterrows():
    gemini_case1_criterion_scores1[row["Criterion"]] = row["Score"]


# Combine all metrics into a summary table
case1_table1_of_metrics = pd.DataFrame({
    "Model": ["GPT-4o", "Gemini 2.0 Flash"],
    "Cosine Similarity": [gpt_case1_cosine_similarity1, gemini_case1_cosi

    # Dynamically add LLM-based criterion scores
    **{f"{criterion} Score": [gpt_case1_criterion_scores1.get(criterion,
       for criterion in gpt_case1_criterion_scores1.keys()},

    # NLP-based evaluation metrics
    "F1 Score": [gpt_case1_nlp_evaluation_results1["F1 Score"], gemini_ca
    "ROUGE-L": [gpt_case1_nlp_evaluation_results1["ROUGE-L"], gemini_case
    "BLEU": [gpt_case1_nlp_evaluation_results1["BLEU"], gemini_case1_nlp_

     # Efficiency metrics
    "Total Cost (USD)": [gpt_case1_total_cost1, gemini_case1_total_cost1]
    "Response Time (s)": [gpt_case1_response_time1, gemini_case1_response
})
```

In [ ]:
```
case1_table1_of_metrics
```

Out[ ]:

	Model	Cosine Similarity	CORRECTNESS Score	RELEVANCE Score	COHERENCE Score	CONCISENESS Score	S
0	GPT-4o	0.443316	1	1	1	0	0.
1	Gemini 2.0 Flash	0.510786	1	1	1	1	0.

Case 1 Question 2

In [ ]:
```
gpt_case1_total_cost2 = gpt_case1_usages2[0]["Total Cost (USD)"]
gpt_case1_response_time2 = gpt_case1_usages2[0]["Response Time (s)"]

gemini_case1_total_cost2 = gemini_case1_usages2[0]["Total Cost (USD)"]
gemini_case1_response_time2 = gemini_case1_usages2[0]["Response Time (s)"

gpt_case1_cosine_similarity2 = case1_cosine_similarities2[case1_cosine_si
gemini_case1_cosine_similarity2 = case1_cosine_similarities2[case1_cosine

gpt_case1_criterion_scores2 = {}
for _, row in gpt_case1_evaluation_results2.iterrows():
    gpt_case1_criterion_scores2[row["Criterion"]] = row["Score"]

gemini_case1_criterion_scores2 = {}
for _, row in gemini_case1_evaluation_results2.iterrows():
    gemini_case1_criterion_scores2[row["Criterion"]] = row["Score"]

case1_table2_of_metrics = pd.DataFrame({
    "Model": ["GPT-4o", "Gemini 2.0 Flash"],
    "Cosine Similarity": [gpt_case1_cosine_similarity2, gemini_case1_cosi

    **{f"{criterion} Score": [gpt_case1_criterion_scores2.get(criterion,
       for criterion in gpt_case1_criterion_scores2.keys()},

    "F1 Score": [gpt_case1_nlp_evaluation_results2["F1 Score"], gemini_ca
    "ROUGE-L": [gpt_case1_nlp_evaluation_results2["ROUGE-L"], gemini_case
    "BLEU": [gpt_case1_nlp_evaluation_results2["BLEU"], gemini_case1_nlp_

    "Total Cost (USD)": [gpt_case1_total_cost2, gemini_case1_total_cost2]
    "Response Time (s)": [gpt_case1_response_time2, gemini_case1_response
})

case1_table2_of_metrics
```

Out[ ]:

	Model	Cosine Similarity	CORRECTNESS Score	RELEVANCE Score	COHERENCE Score	CONCISENESS Score	S
0	GPT-4o	0.701854	1	1	1	0	0.
1	Gemini 2.0 Flash	0.697518	1	1	1	0	0.

Case 2 Question 1

In [ ]:
```
gpt_case2_total_cost1 = gpt_case2_usages1[0]["Total Cost (USD)"]
gpt_case2_response_time1 = gpt_case2_usages1[0]["Response Time (s)"]

gemini_case2_total_cost1 = gemini_case2_usages1[0]["Total Cost (USD)"]
gemini_case2_response_time1 = gemini_case2_usages1[0]["Response Time (s)"

gpt_case2_cosine_similarity1 = case2_cosine_similarities1[case2_cosine_si
gemini_case2_cosine_similarity1 = case2_cosine_similarities1[case2_cosine
```

```
gpt_case2_criterion_scores1 = {}
for _, row in gpt_case2_evaluation_results1.iterrows():
    gpt_case2_criterion_scores1[row["Criterion"]] = row["Score"]

gemini_case2_criterion_scores1 = {}
for _, row in gemini_case2_evaluation_results1.iterrows():
    gemini_case2_criterion_scores1[row["Criterion"]] = row["Score"]

case2_table1_of_metrics = pd.DataFrame({
    "Model": ["GPT-4o", "Gemini 2.0 Flash"],
    "Cosine Similarity": [gpt_case2_cosine_similarity1, gemini_case2_cosi

    **{f"{criterion} Score": [gpt_case2_criterion_scores1.get(criterion,
       for criterion in gpt_case2_criterion_scores1.keys()},

    "F1 Score": [gpt_case2_nlp_evaluation_results1["F1 Score"], gemini_ca
    "ROUGE-L": [gpt_case2_nlp_evaluation_results1["ROUGE-L"], gemini_case
    "BLEU": [gpt_case2_nlp_evaluation_results1["BLEU"], gemini_case2_nlp_

    "Total Cost (USD)": [gpt_case2_total_cost1, gemini_case2_total_cost1]
    "Response Time (s)": [gpt_case2_response_time1, gemini_case2_response

})

case2_table1_of_metrics
```

Out[ ]:

	Model	Cosine Similarity	CORRECTNESS Score	RELEVANCE Score	COHERENCE Score	CONCISENESS Score	S
0	GPT-4o	0.781913	1	1	1	0	0.
1	Gemini 2.0 Flash	0.763158	1	1	1	1	0.

Case 2 Question 2

In [ ]:

```
gpt_case2_total_cost2 = gpt_case2_usages2[0]["Total Cost (USD)"]
gpt_case2_response_time2 = gpt_case2_usages2[0]["Response Time (s)"]

gemini_case2_total_cost2 = gemini_case2_usages2[0]["Total Cost (USD)"]
gemini_case2_response_time2 = gemini_case2_usages2[0]["Response Time (s)"

gpt_case2_cosine_similarity2 = case2_cosine_similarities2[case2_cosine_si
gemini_case2_cosine_similarity2 = case2_cosine_similarities2[case2_cosine

gpt_case2_criterion_scores2 = {}
for _, row in gpt_case2_evaluation_results2.iterrows():
    gpt_case2_criterion_scores2[row["Criterion"]] = row["Score"]

gemini_case2_criterion_scores2 = {}
for _, row in gemini_case2_evaluation_results2.iterrows():
    gemini_case2_criterion_scores2[row["Criterion"]] = row["Score"]

case2_table2_of_metrics = pd.DataFrame({
    "Model": ["GPT-4o", "Gemini 2.0 Flash"],
```

```
    "Cosine Similarity": [gpt_case2_cosine_similarity2, gemini_case2_cosi

    **{f"{criterion} Score": [gpt_case2_criterion_scores2.get(criterion,
       for criterion in gpt_case2_criterion_scores2.keys()},

    "F1 Score": [gpt_case2_nlp_evaluation_results2["F1 Score"], gemini_ca
    "ROUGE-L": [gpt_case2_nlp_evaluation_results2["ROUGE-L"], gemini_case
    "BLEU": [gpt_case2_nlp_evaluation_results2["BLEU"], gemini_case2_nlp_

    "Total Cost (USD)": [gpt_case2_total_cost2, gemini_case2_total_cost2]
    "Response Time (s)": [gpt_case2_response_time2, gemini_case2_response

})

case2_table2_of_metrics
```

Out[ ]:

	Model	Cosine Similarity	CORRECTNESS Score	RELEVANCE Score	COHERENCE Score	CONCISENESS Score	S
0	GPT-4o	0.705067	1	1	1	0	0.
1	Gemini 2.0 Flash	0.653491	0	1	1	1	0.

Final summary

Let's aggregate evaluation results across multiple cases for GPT-4o and Gemini 2.0 Flash and create a final summary of averaged metrics.

In [ ]:

```
def compute_case_summary(tables, model_name):

    # Step 1: Filter data for the specified model across all tables
    model_df = pd.concat([df[df["Model"] == model_name] for df in tables],

    # Step 2: Define the list of metrics to be averaged
    metrics = ["Cosine Similarity", "CORRECTNESS Score", "RELEVANCE Score"
               "COHERENCE Score", "CONCISENESS Score", "F1 Score", "ROUGE-

    # Compute average values across all cases for each metric
    averaging_metrics = {f"Average {m}": model_df[m].mean() for m in metri

    # Step 3: Define the list of additive metrics for Total Cost and Respo
    additive_metrics = {
        "Case-wise Average Cost (USD)": model_df["Total Cost (USD)"].sum()
        "Case-wise Average Time (s)": model_df["Response Time (s)"].sum()
    }

    # Merge both types of metrics into a single summary
    return averaging_metrics | additive_metrics

# Prepare evaluation tables for each case
case1_tables = [case1_table1_of_metrics, case1_table2_of_metrics]
case2_tables = [case2_table1_of_metrics, case2_table2_of_metrics]

# Compute summary for each model acrss cases
```

```
summaries = {
   "case1": {m: compute_case_summary(case1_tables, m) for m in ["GPT-4o",
   "case2": {m: compute_case_summary(case2_tables, m) for m in ["GPT-4o",
}

# Compute the final average summary across all cases
final_summary_df = pd.DataFrame.from_dict({
   model: {k: (summaries["case1"][model][k] + summaries["case2"][model][k
          for k in summaries["case1"][model].keys()
          }
   for model in ["GPT-4o", "Gemini 2.0 Flash"]
}, orient="index")
```

In [ ]:
```
final_summary_df
```

Out[ ]:

	Average Cosine Similarity	Average CORRECTNESS Score	Average RELEVANCE Score	Average COHERENCE Score	Average CONCISENESS Score	Aver F1 Sc
GPT-4o	0.658038	1.00	1.0	1.0	0.00	0.169
Gemini 2.0 Flash	0.656238	0.75	1.0	1.0	0.75	0.153