GenAI Unit 2 - Submission 1 - Handson Assignment 1 (Langchain + Prompt Engineering)

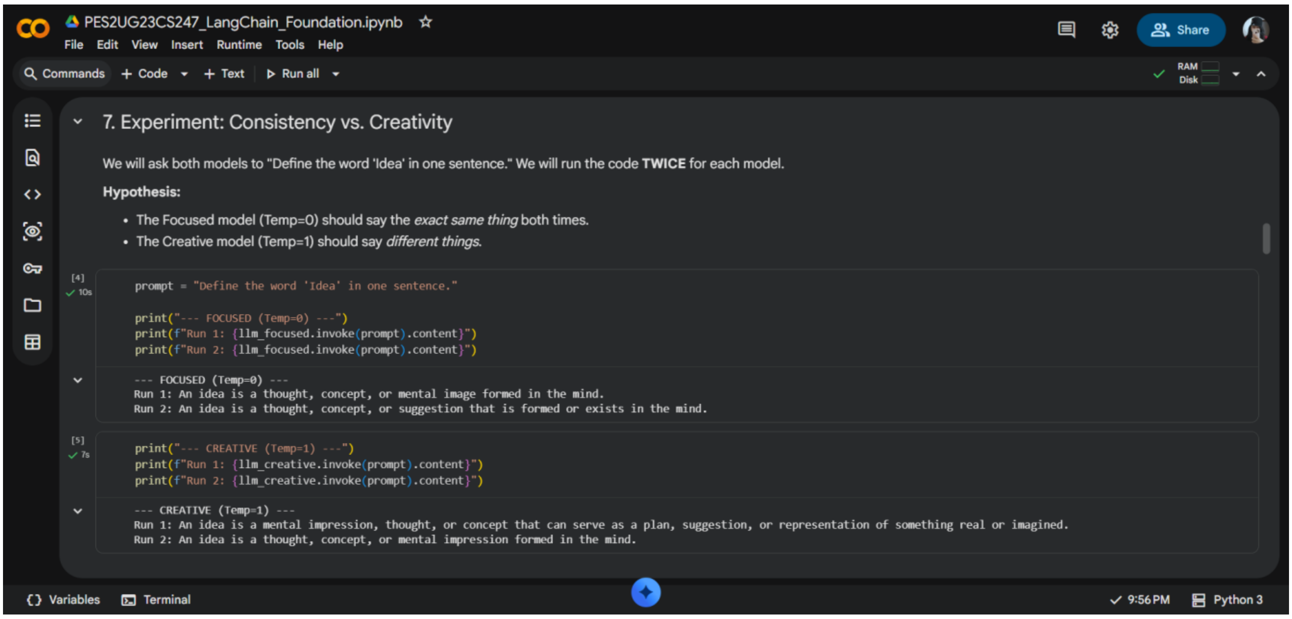
Name: K L Sonika

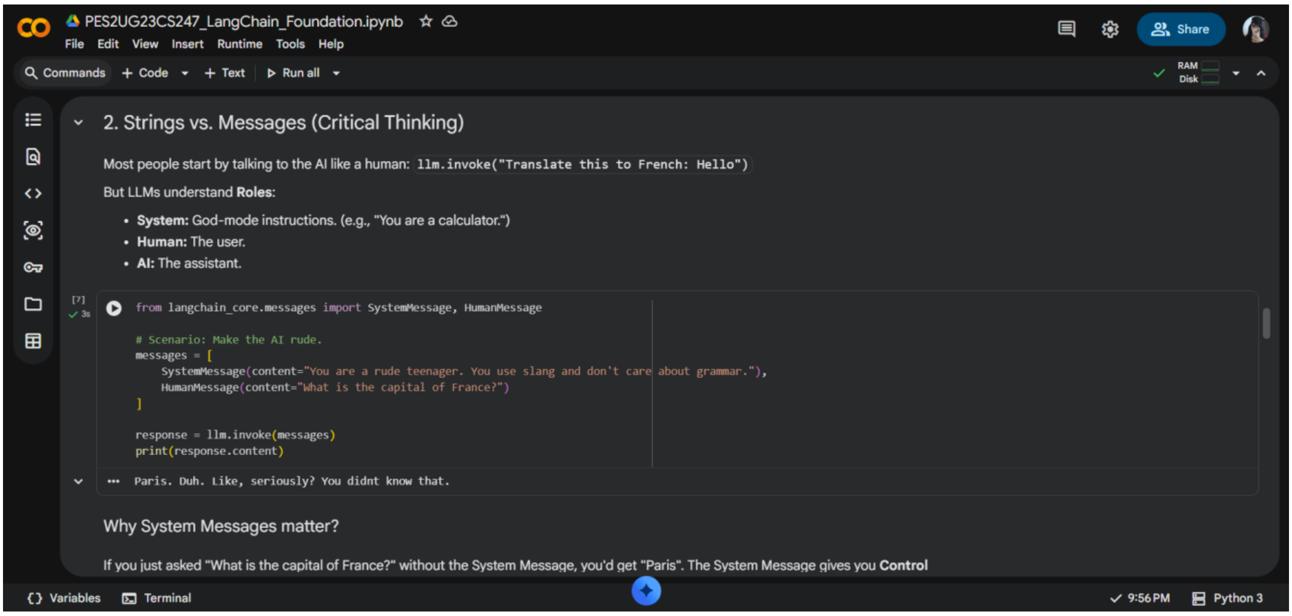
SRN: PES2UG23CS247

Section: D

Date: 13-02-2026

LangChain Foundation





PES2UG23CS247_LangChain_Foundation.ipynb

```python
# 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.

```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: Hello! How can I help you today?
```

PES2UG23CS247_LangChain_Foundation.ipynb

2. Method A: The Manual Way (Bad)

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

```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 one:

Crows are incredibly intelligent and have an amazing ability to **recognize and remember individual human faces!**

Scientists have done experiments where crows were harassed by people wearing specific masks. Years later, those same crows (and even their offspring!) would scold and mob any

So, if you're ever nice to a crow, it might just remember you as a friend (and tell its buddies!). But if you're mean... well, they might hold a grudge!
```

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.

PES2UG23CS247_LangChain_Foundation.ipynb

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.

```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 the third circulates blood to the rest of their body. And because their blood is copper-based (not ir
```

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

Imagine you want to swap the Model.

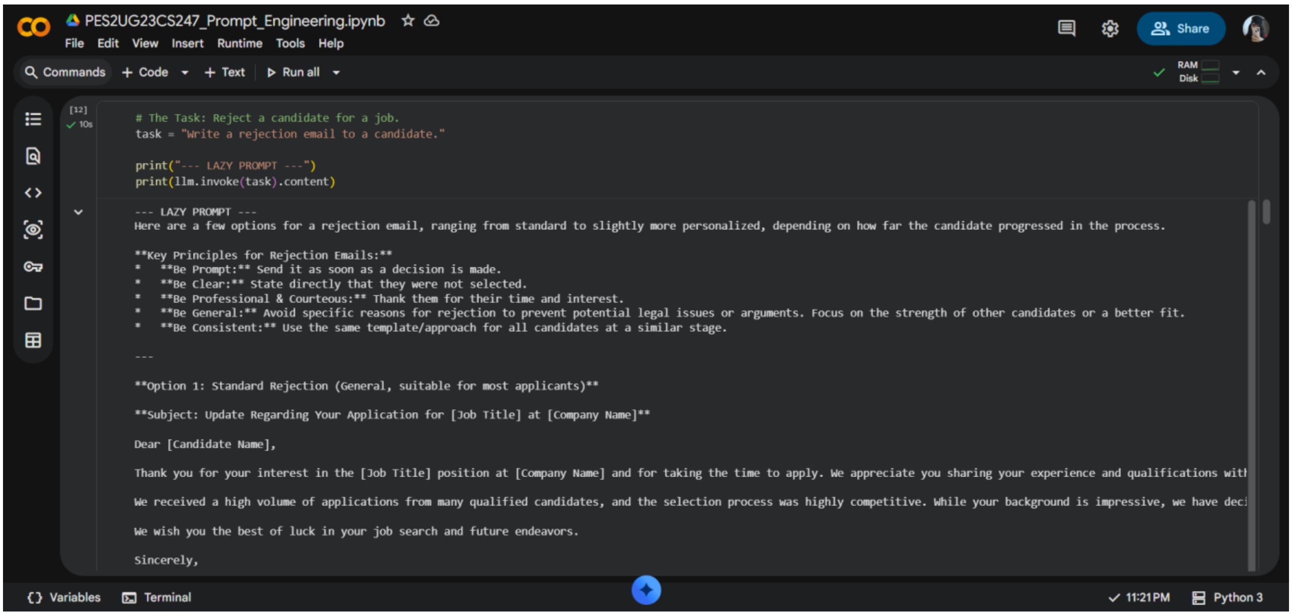
- **Manual:** You hunt for the line where `llm.invoke` happens.
- **LCEL:** You just change the `llm` variable in the chain definition.

Imagine you want to add a step (e.g., a spellchecker) between the prompt and the model.

- **LCEL:** `chain = template | spellchecker | llm | parser`

It makes your AI logic **Composable**.

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 standard to slightly more personalized, depending on how far the candidate progressed in the process.

**Key Principles for Rejection Emails:**
*   **Be Prompt:** Send it as soon as a decision is made.
*   **Be Clear:** State directly that they were not selected.
*   **Be Professional & Courteous:** Thank them for their time and interest.
*   **Be General:** Avoid specific reasons for rejection to prevent potential legal issues or arguments. Focus on the strength of other candidates or a better fit.
*   **Be Consistent:** Use the same template/approach for all candidates at a similar stage.

---

**Option 1: Standard Rejection (General, suitable for most applicants)**

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

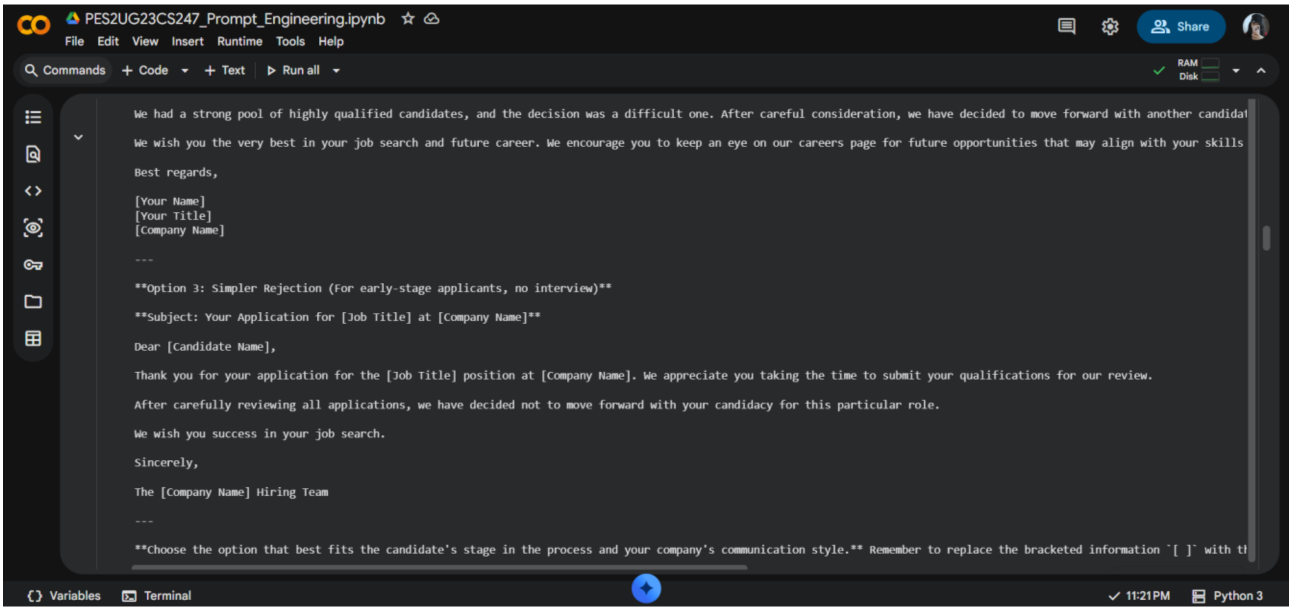
Dear [Candidate Name],

Thank you for your interest in the [Job Title] position at [Company Name] and for taking the time to apply. We appreciate you sharing your experience and qualifications with

We received a high volume of applications from many qualified candidates, and the selection process was highly competitive. While your background is impressive, we have deci

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

Sincerely,
```




```
We had a strong pool of highly qualified candidates, and the decision was a difficult one. After careful consideration, we have decided to move forward with another candidat

We wish you the very best in your job search and future career. We encourage you to keep an eye on our careers page for future opportunities that may align with your skills

Best regards,

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

---

**Option 3: Simpler Rejection (For early-stage applicants, no interview)**

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

Dear [Candidate Name],

Thank you for your application for the [Job Title] position at [Company Name]. We appreciate you taking the time to submit your qualifications for our review.

After carefully reviewing all applications, we have decided not to move forward with your candidacy for this particular role.

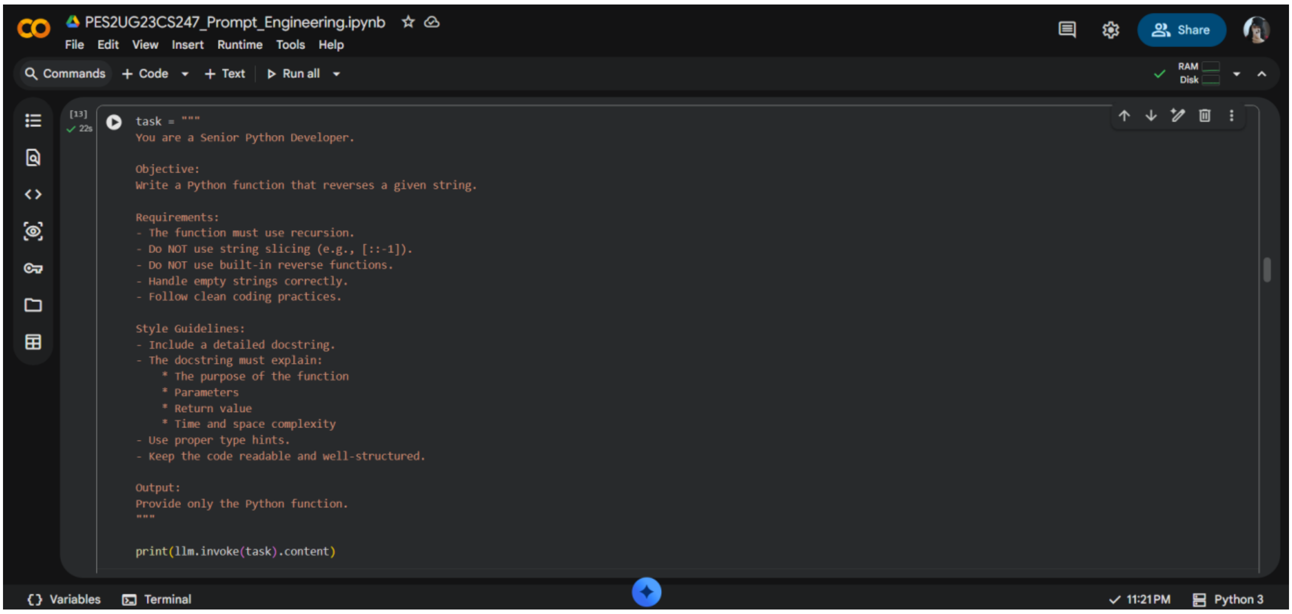
We wish you success in your job search.

Sincerely,

The [Company Name] Hiring Team

---

**Choose the option that best fits the candidate's stage in the process and your company's communication style.** Remember to replace the bracketed information `[ ]` with th
```




```
task = """
You are a Senior Python Developer.

Objective:
Write a Python function that reverses a given string.

Requirements:
- The function must use recursion.
- Do NOT use string slicing (e.g., [::-1]).
- Do NOT use built-in reverse functions.
- Handle empty strings correctly.
- Follow clean coding practices.

Style Guidelines:
- Include a detailed docstring.
- The docstring must explain:
    * The purpose of the function
    * Parameters
    * Return value
    * Time and space complexity
- Use proper type hints.
- Keep the code readable and well-structured.

Output:
Provide only the Python function.
"""

print(llm.invoke(task).content)
```

PES2UG23CS247_Prompt_Engineering.ipynb

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

    This function takes a string and returns a new string with its characters
    in reverse order. It implements a recursive approach where the first
    character of the string is appended to the end of the recursively
    reversed remainder of the string.

    Args:
        s (str): The input string to be reversed.

    Returns:
        str: The reversed string.

    Time Complexity:
        O(n^2), where n is the length of the input string.
        This complexity arises from two main factors:
        1. Substring creation: In Python, `s[1:]` creates a new string object
           for each recursive call. Creating a substring of length k takes O(k) time.
           Across n recursive calls (for string lengths n, n-1, ..., 1), the
           total time for substring creation sums up to O(n + (n-1) + ... + 1) = O(n^2).
        2. String concatenation: The `+` operator for strings in Python creates
           a new string. Concatenating a character to a string of length k takes
           O(k+1) time. As the recursive calls return and concatenate, the
           lengths of the strings being concatenated grow, leading to a total
           cost of O(1 + 2 + ... + n) = O(n^2) for concatenations.

    Space Complexity:
        O(n), where n is the length of the input string.
        This is primarily due to the recursion call stack. For each character in
```

PES2UG23CS247_Prompt_Engineering.ipynb

```
           a new string. Concatenating a character to a string of length k takes
           O(k+1) time. As the recursive calls return and concatenate, the
           lengths of the strings being concatenated grow, leading to a total
           cost of O(1 + 2 + ... + n) = O(n^2) for concatenations.

    Space Complexity:
        O(n), where n is the length of the input string.
        This is primarily due to the recursion call stack. For each character in
        the string, a new stack frame is added. For a string of length n, there
        will be n stack frames. Each frame stores local variables and references
        to the substrings created.
        While temporary string objects created during substring slicing and
        concatenation also consume memory, the maximum *concurrent* space used
        (excluding the final result string) is dominated by the call stack depth.
    """
    # Base case:
    # If the string is empty or contains only one character,
    # it is already reversed, so return it as is.
    if len(s) <= 1:
        return s

    # Recursive step:
    # Take the first character of the string (s[0]) and append it to the
    # result of recursively reversing the rest of the string (s[1:]).
    # For example, to reverse "abc":
    # 1. Calls reverse_string_recursive("bc") and adds 'a' to its result.
    # 2. reverse_string_recursive("bc") calls reverse_string_recursive("c") and adds 'b' to its result.
    # 3. reverse_string_recursive("c") hits the base case and returns "c".
    # 4. "c" + 'b' returns "cb".
    # 5. "cb" + 'a' returns "cba".
    return reverse_string_recursive(s[1:]) + s[0]
```
```

PES2UG23CS247_Prompt_Engineering.ipynb

```python
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 this time.

Keep flying,
RocketBoots HR
```

PES2UG23CS247_Prompt_Engineering.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

[7]

```python
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 combining "Angry" and "Hungry" is:

**Hangry**
```

3. Few-Shot (Pattern Matching)

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

[8]

```python
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: Hangry (The severe medical condition that makes you believe everyone else is personally responsible for the fact you haven't had a snack in the last hour.)
```

Variables Terminal 11:21PM Python 3

PES2UG23CS247_Prompt_Engineering.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

[10]

```python
    {"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)
```

```
We've identified several areas for improvement within this application.
```

Variables Terminal 11:21PM Python 3

RAG

RAG.ipynb

```python
import numpy as np

def cosine_similarity(a, b):
    return np.dot(a, b) / (np.linalg.norm(a) * np.linalg.norm(b))

vec_cat = embeddings.embed_query("Cat")
vec_dog = embeddings.embed_query("Dog")
vec_car = embeddings.embed_query("Car")

print(f"Cat vs Dog: {cosine_similarity(vec_cat, vec_dog):.4f}")
print(f"Cat vs Car: {cosine_similarity(vec_cat, vec_car):.4f}")
```

```
Cat vs Dog: 0.6606
Cat vs Car: 0.4633
```

Analysis You should see that Cat & Dog score higher (e.g., ~0.8) than Cat & Car (e.g., ~0.3). This Mathematical Distance is the foundation of all Search engines and RAG systems. This is arguably the most important concept in modern AI.

Unit 2 - Part 4b: Naive RAG Pipeline

1. Introduction: The Open-Book Test RAG (Retrieval-Augmented Generation) is just an Open-Book Test architecture.

Retrieval: Find the right page in the textbook. Generation: Write the answer using that page. The Pipeline (Flowchart) graph TD User[User Question] --> Retriever[Retriever System] Retriever -->|Search Database| Docs[Relevant Documents] Docs --> Combiner[Prompt Template] User --> Combiner Combiner -->|Full Prompt w/ Context| LLM[Gemini Model] LLM --> Answer[Final Answer]

PES2UG23CS247_RAG.ipynb

```python
nlist = 100 # How many 'zip codes' (clusters) we want
quantizer = faiss.IndexFlatL2(d) # The calculator for distance
index_ivf = faiss.IndexIVFFlat(quantizer, d, nlist)

# We MUST train it first so it learns where the clusters are
index_ivf.train(xb)
index_ivf.add(xb)
```

```python
print("Is index trained?", index_ivf.is_trained)
print("Total vectors in index:", index_ivf.ntotal)
print("Number of clusters (nlist):", index_ivf.nlist)
```

```
Is index trained? True
Total vectors in index: 10000
Number of clusters (nlist): 100
```

```python
index_ivf.nprobe = 5   # search in 5 clusters

xq = np.random.random((1, d)).astype('float32')
D, I = index_ivf.search(xq, 5)

print("Nearest indices:", I)
print("Distances:", D)
```

```
Nearest indices: [[5806 3980 3784 5004 8457]]
Distances: [[15.095278 15.32453  15.595385 16.149902 16.357386]]
```

PES2UG23CS247_RAG.ipynb

4. The RAG Chain We use LCEL to stitch it together.

Step 1: The retriever takes the question, converts it to numbers, and finds the closest document. Step 2: RunnablePassthrough holds the question. Step 3: The prompt combines them.

```python
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'.
```

PES2UG23CS247_RAG.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text ▷ Run all

2. Flat Index (Brute Force) Concept: Check every single item.

Algo: IndexFlatL2 Pros: 100% Accuracy (Gold Standard). Cons: Slow (O(N)). Unusable at 1M+ vectors.

[11] 0s

```
index = faiss.IndexFlatL2(d)
index.add(xb)
print(f"Flat Index contains {index.ntotal} vectors")
xq = np.random.random((1, d)).astype('float32')
k = 5
D, I = index.search(xq, k)

print("Nearest vector indices:", I)
print("Distances:", D)

vector_0 = index.reconstruct(0)
print("First 10 numbers of vector_0:", vector_0[:10])
```

```
Flat Index contains 10000 vectors
Nearest vector indices: [[9025 6483 8863 7016 7236]]
Distances: [[13.471307  13.6593275 14.268453  14.365671  14.367761 ]]
First 10 numbers of vector_0: [0.8173798  0.0738485  0.6384204  0.4677386  0.0191156  0.40989366
 0.36507857 0.3649886  0.13669589 0.8743454 ]
```

IVF (Inverted File Index) Concept: Clustering / Partitioning.

Imagine looking for a book. Instead of checking every shelf, you go to the "Sci-Fi" section. Then you only search books in that section.

How it works (Flowchart) graph TD Data[All 1M Vectors] -->|Train| Clusters[1000 Cluster Centers (Centroids)] Query[User Query] -->|Step 1| FindClosest[Find Closest Centroid] FindClosest -->|Step 2| Search[Search ONLY vectors in that Cluster] Analogy: Voronoi Cells (Zip

Variables Terminal ✓ 11:42 PM Python 3

PES2UG23CS247_RAG.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text ▷ Run all

0s

```
Is index trained? True
Total vectors in index: 10000
Number of clusters (nlist): 100
```

[17] 0s

```
index_ivf.nprobe = 5   # search in 5 clusters

xq = np.random.random((1, d)).astype('float32')
D, I = index_ivf.search(xq, 5)

print("Nearest indices:", I)
print("Distances:", D)
```

```
Nearest indices: [[3505 8411 4056 5342 2176]]
Distances: [[15.932281 16.050947 16.056328 16.420364 16.654198]]
```

[18] 0s

```
index_ivf.nprobe = 5   # search in 5 clusters

xq = np.random.random((1, d)).astype('float32')
D, I = index_ivf.search(xq, 5)

print("Nearest indices:", I)
print("Distances:", D)
```

```
Nearest indices: [[2217 3333 7822 7855 3814]]
Distances: [[14.369042 14.54064  15.089977 15.147356 15.177164]]
```

Variables Terminal ✓ 11:42 PM Python 3

PES2UG23CS247_RAG.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text ▷ Run all

[22] 0s

```
xq = np.random.random((1, d)).astype('float32')

D, I = index_hnsw.search(xq, 5)

print("Nearest indices:", I)
print("Distances:", D)
```

```
Nearest indices: [[8955 5858 9991 2137 1086]]
Distances: [[13.987291 14.367203 14.814301 14.909238 15.030158]]
```

[23] 1s

```
M = 16 # Number of connections per node (The 'Hub' factor)
index_hnsw = faiss.IndexHNSWFlat(d, M)
index_hnsw.add(xb)
```

[24] 0s

```
xq = np.random.random((1, d)).astype('float32')

D, I = index_hnsw.search(xq, 5)

print("Nearest indices:", I)
print("Distances:", D)
```

```
Nearest indices: [[8665 9512  271 1032 4862]]
Distances: [[13.851658 14.833803 14.84812  14.998225 15.139857]]
```

5. PQ (Product Quantization) Concept: Compression (Lossy).

Do we need 32-bit float precision (0.123456789)? No. 0.12 is fine. PQ breaks the vector into chunks and approximates them.

Variables Terminal ✓ 11:42 PM Python 3

PES2UG23CS247_RAG.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

RAM Disk

Do we need 32-bit float precision (0.123456789)? No. 0.12 is fine. PQ breaks the vector into chunks and approximates them.

Analogy: 4K Video vs 480p Video.

480p is blurry, but it's 10x smaller and faster to stream. Use PQ when you are RAM constrained (e.g., storing 1 Billion vectors)

[25] 0s

```
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.
```

Index	Speed	Accuracy	Memory
Flat	Slow	100%	High
IVF	Fast	High	Medium
HNSW	Very Fast	Very High	High
PQ	Very Fast	Medium	Very Low

Method	Think as
Flat	Check All
IVF	Go to Section
HNSW	Travel via Hubs
PQ	Compress Data

Flat → Exact but heavy IVF → Clustered search HNSW → Graph navigation PQ → Compressed storage

Variables Terminal ✓ 11:42 PM Python 3