MASTERING LLM PRESENTS

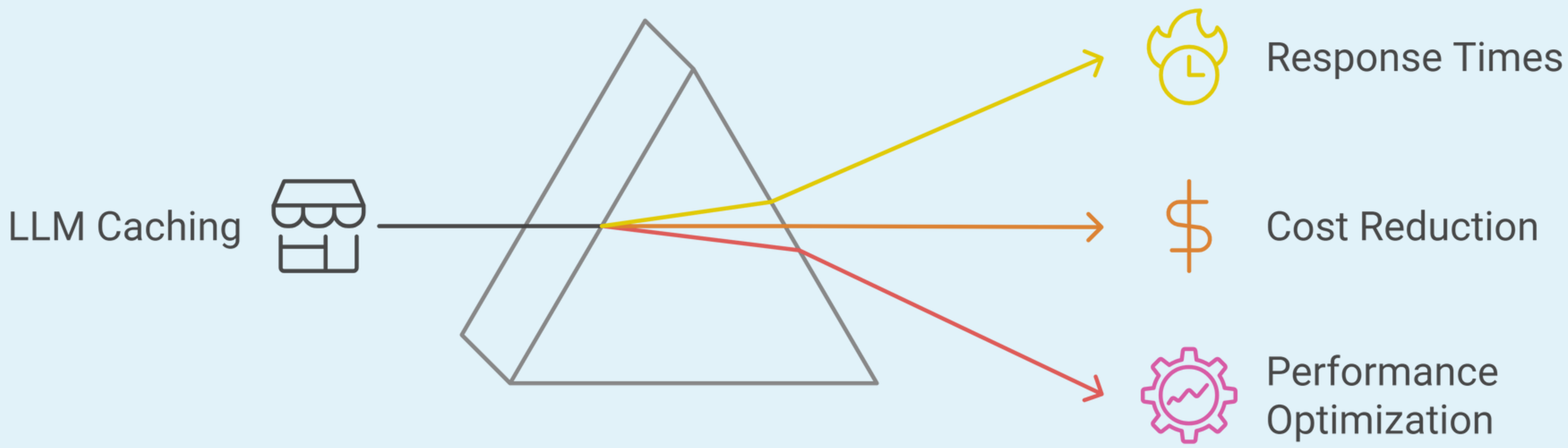
COFFEE BREAK CONCEPTS

Caching Methods in Large Language Models (LLMs)

Follow us on

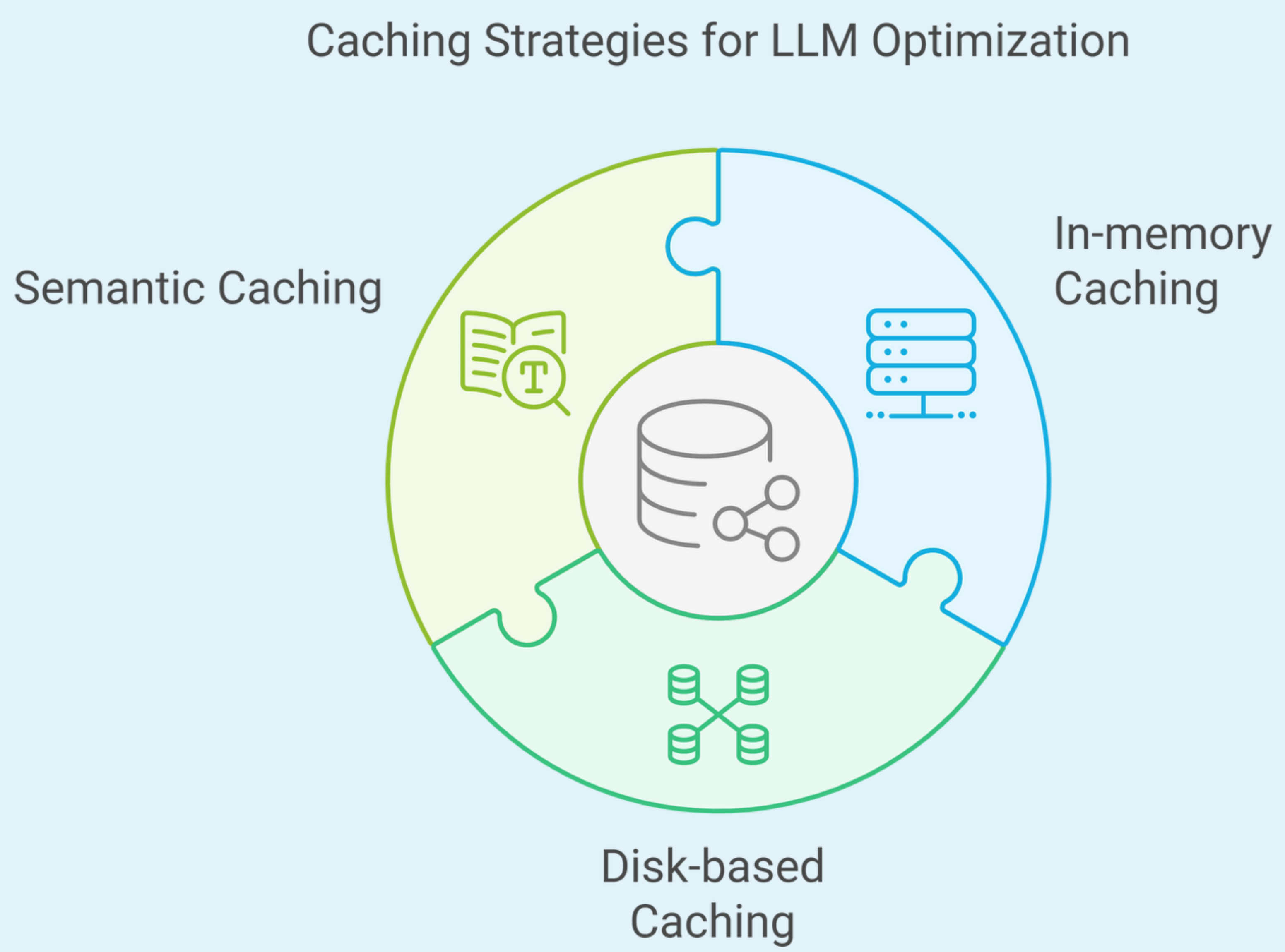
What is caching?

1. Store and reuse responses
2. Improve response time
3. Reduce cost



Types of Caching

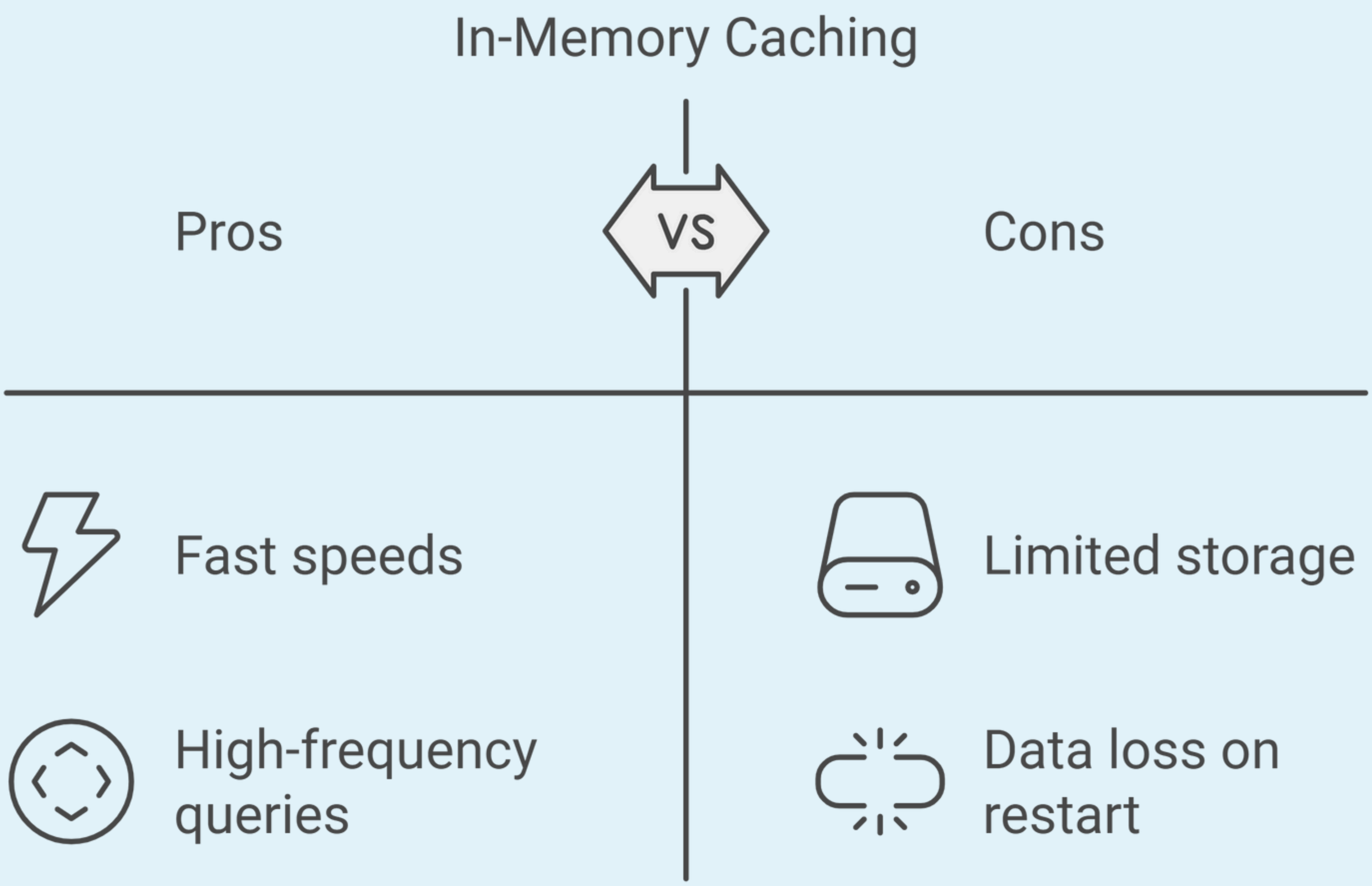
1. In-memory caching
2. Disk-based caching
3. Semantic caching



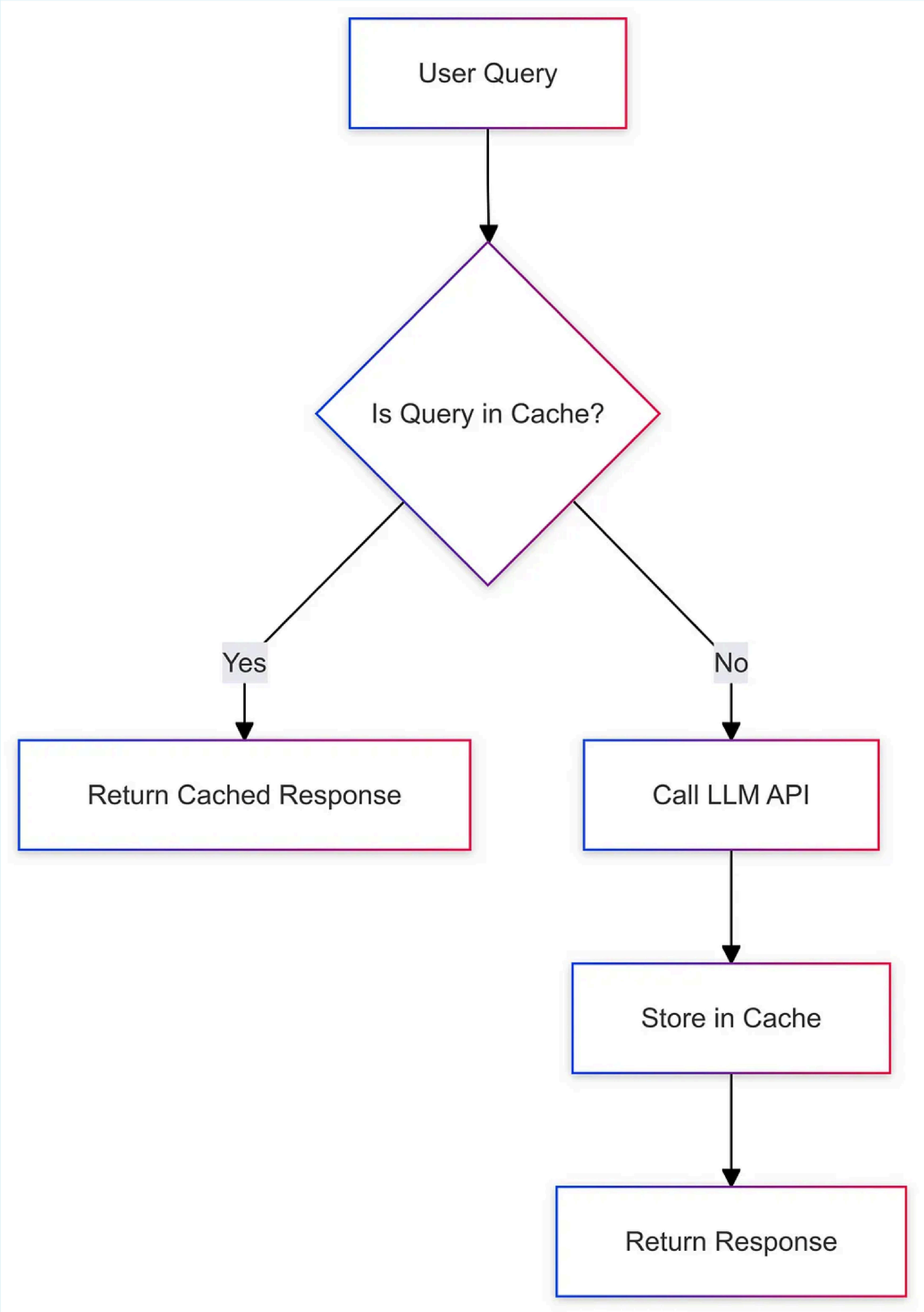
In-Memory Caching

1. Stores data in RAM for rapid access.

2. **Example**: Using a dictionary in Python or tools like Redis for quick data retrieval.



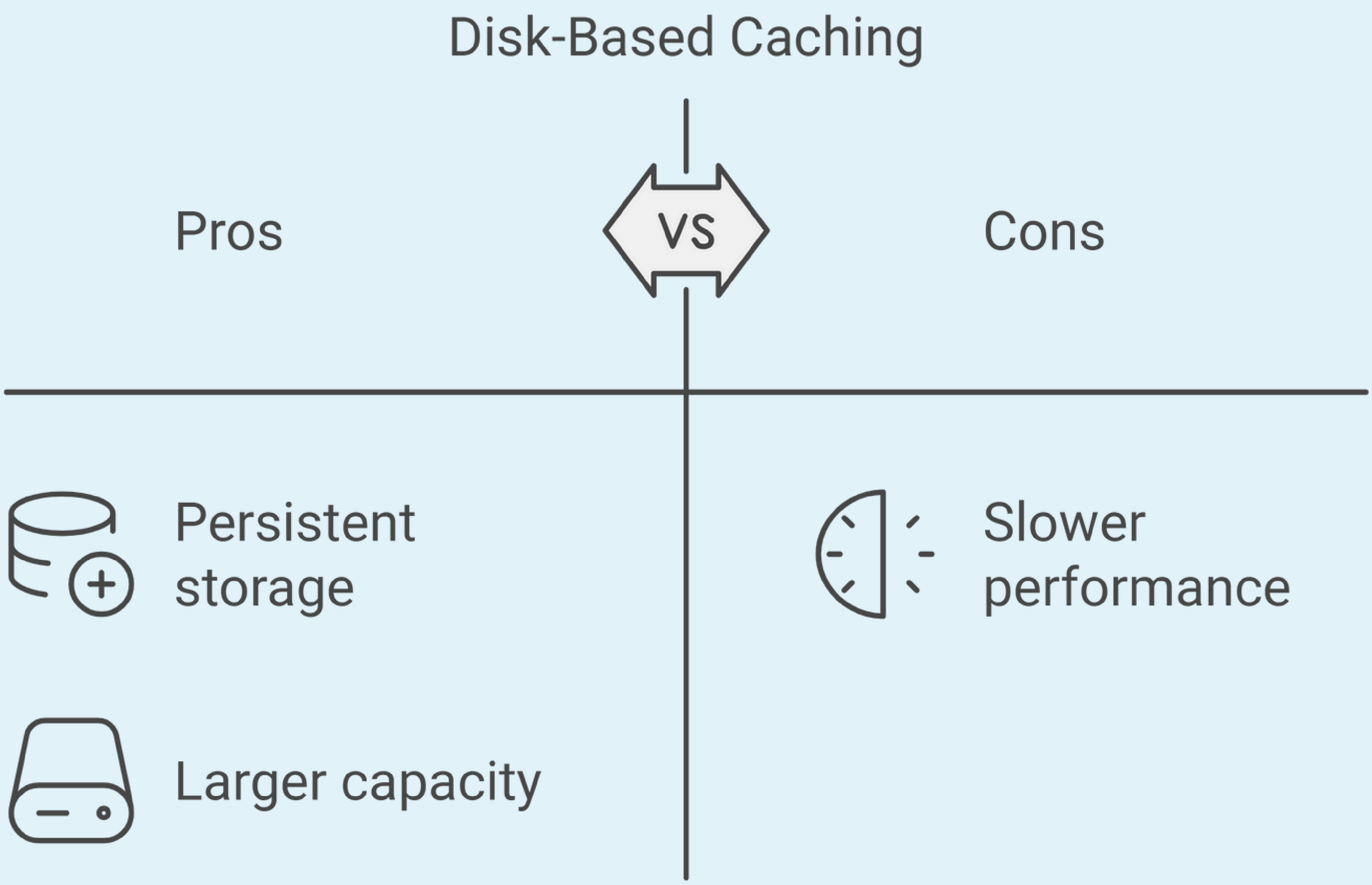
In-Memory Caching



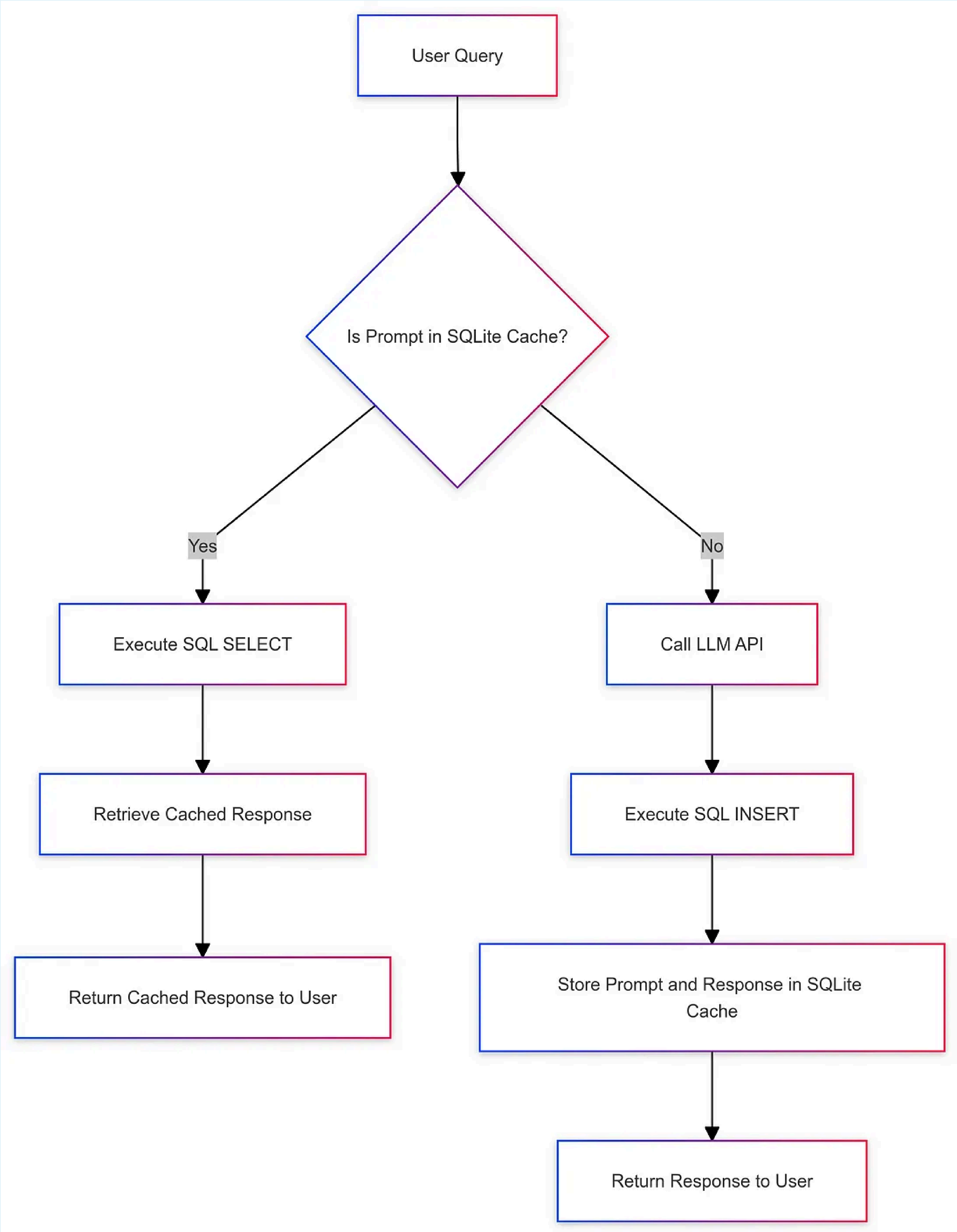
Disk-Based Caching

1. Stores data on disk using databases like SQLite.

2. **Example:** Utilizing SQLite or other disk-based databases to store cached responses.

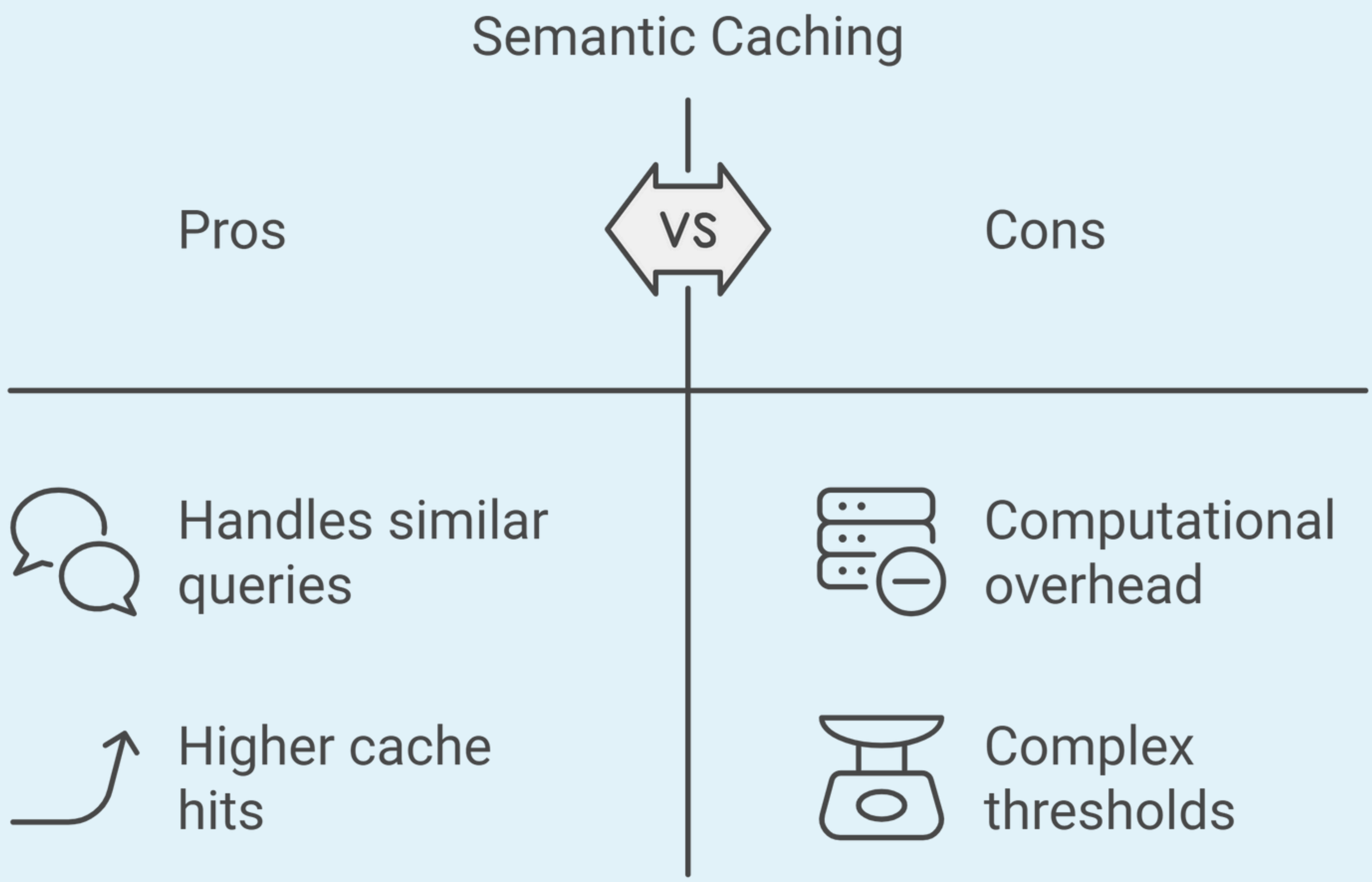


Disk-Based Caching

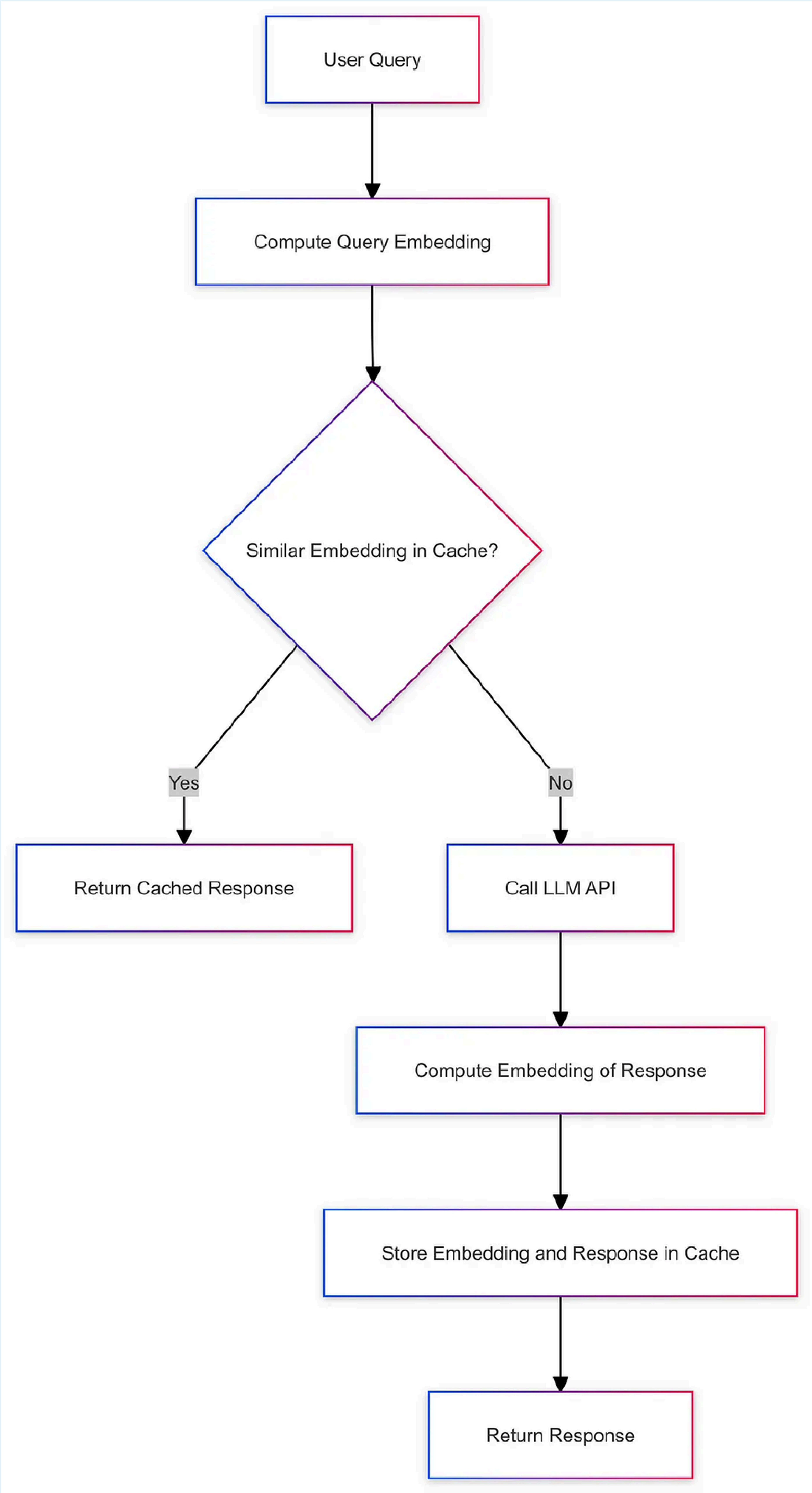


Semantic Caching

1 Stores responses based on the semantic meaning of queries using embeddings.



Semantic Caching



Without Caching

Without Caching.py

```python
import time

def call_llm_api(prompt):
    # Simulate LLM API call delay
    time.sleep(2)  # Simulate network latency and processing time
    return f"LLM response to: {prompt}"

def get_response_no_cache(prompt):
    start_time = time.time()
    response = call_llm_api(prompt)
    end_time = time.time()
    print(f"Time taken without caching: {end_time - start_time:.2f} seconds")
    return response

# Usage
response = get_response_no_cache("What is the capital of France?")
```

CodeImage

Output

```
Time taken without caching: 2.00 seconds
```

Note: We are not doing LLM call for simplicity.

Disk-Based Caching

```python
import sqlite3
import time

def initialize_cache():
    conn = sqlite3.connect('cache.db')
    cursor = conn.cursor()
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS cache (
            prompt TEXT PRIMARY KEY,
            response TEXT
        )
    ''')
    conn.commit()
    conn.close()

def call_llm_api(prompt):
    # Simulate LLM API call delay
    time.sleep(2)  # Simulate network latency and processing time
    return f"LLM response to: {prompt}"

def get_response_disk_cache(prompt):
    start_time = time.time()
    conn = sqlite3.connect('cache.db')
    cursor = conn.cursor()
    # Check if the prompt exists in the cache
    cursor.execute('SELECT response FROM cache WHERE prompt=?', (prompt,))
    result = cursor.fetchone()
    if result:
        # Cache hit
        conn.close()
        end_time = time.time()
        print("Cache hit - Retrieved from SQLite")
        print(f"Time taken with disk-based caching (cache hit): {end_time - start_time:.2f} seconds")
        return result[0]
    else:
        # Cache miss
        response = call_llm_api(prompt)
        # Store the new prompt and response in the cache
        cursor.execute('INSERT INTO cache (prompt, response) VALUES (?, ?)', (prompt, response))
        conn.commit()
        conn.close()
        end_time = time.time()
        print("Cache miss - Not found in SQLite")
        print(f"Time taken with disk-based caching (cache miss): {end_time - start_time:.2f} seconds")
        return response

# Initialize cache database
initialize_cache()

# Usage
# First call (cache miss)
response = get_response_disk_cache("What is the tallest mountain in the world?")
print(response)

# Second call (cache hit)
response = get_response_disk_cache("What is the tallest mountain in the world?")
print(response)
```

Output

```
Cache miss - Not found in SQLite
Time taken with disk-based caching (cache miss): 2.05 seconds
LLM response to: What is the tallest mountain in the world?

Cache hit - Retrieved from SQLite
Time taken with disk-based caching (cache hit): 0.01 seconds
LLM response to: What is the tallest mountain in the world?
```

With Semantic Caching

Semantic.py

```python
import time
from sentence_transformers import SentenceTransformer
import faiss
import numpy as np

# Initialize the embedding model and index
model = SentenceTransformer('all-MiniLM-L6-v2')
embedding_dim = 384  # Embedding size for 'all-MiniLM-L6-v2'
index = faiss.IndexFlatL2(embedding_dim)
embeddings = []
responses = []

def call_llm_api(prompt):
    # Simulate LLM API call delay
    time.sleep(2)  # Simulate network latency and processing time
    return f"LLM response to: {prompt}"

def get_response_semantic_cache(prompt, similarity_threshold=0.8):
    start_time = time.time()
    # Compute embedding for the prompt
    embedding = model.encode([prompt])[0]
    embedding = np.array([embedding]).astype('float32')

    if index.ntotal > 0:
        # Search for similar embeddings
        distances, indices = index.search(embedding, k=1)
        similarity = 1 - distances[0][0] / 4  # Normalize distance to similarity
        if similarity ≥ similarity_threshold:
            response = responses[indices[0][0]]
            print(f"Semantic cache hit (similarity: {similarity:.2f})")
        else:
            response = call_llm_api(prompt)
            index.add(embedding)
            responses.append(response)
            print("Semantic cache miss")
    else:
        # Cache is empty
        response = call_llm_api(prompt)
        index.add(embedding)
        responses.append(response)
        print("Semantic cache miss")
    end_time = time.time()
    print(f"Time taken with semantic caching: {end_time - start_time:.2f} seconds")
    return response

# First call (cache miss)
response = get_response_semantic_cache("Tell me about the Eiffel Tower.")

# Second call with a semantically similar prompt (cache hit)
response = get_response_semantic_cache("Give me information on the Eiffel Tower.")
```

CodeImage

Output

```
Semantic cache miss
Time taken with semantic caching: 2.35 seconds
Semantic cache hit (similarity: 0.89)
Time taken with semantic caching: 0.08 seconds
```

Tracking Performance

1. **Cache Hit Ratio:** Percentage of requests served from the cache.
2. **Average Latency:** Time taken to serve requests.
3. **Cost Savings:** Reduction in API calls.

Summary

1. **Choose Caching Type:** Select In-Memory (fast, limited size), Disk-Based (persistent, larger capacity), or Semantic Caching (handles similar queries).
2. **Optimize Cache Usage:** Set clear cache rules; adjust similarity thresholds to balance hit rate and accuracy.
3. **Leverage Tools:** Use libraries like GPTCache; integrate with frameworks like LangChain or OpenAI API.
4. **Monitor Performance:** Track cache hits, latency, cost savings; refine strategy based on metrics.

www.masteringllm.com

AgenticRAG with LlamaIndex

Want to learn why AgenticRAG is future of RAG?

- Master **RAG fundamentals** through practical case studies
- Understand how to overcome **limitations of RAG**
- Introduction to **AgenticRAG** & techniques like **Routing Agents, Query planning agents, Structure planning agents, and React agents with human in loop**.
- **5** real-time **case studies with code walkthroughs**