Project Documentation: Using Google's Gemini AI in Jupyter Notebook

Learn how to integrate and leverage Gemini AI for creative and technical tasks directly within a Jupyter environment.

Overview

This Jupyter Notebook demonstrates a step-by-step workflow to integrate Google's Gemini AI model (gemini-1.5-flash-001) for generating text-based outputs. The example focuses on crafting a rhythmic poem, but the methodology can be adapted for diverse applications like content generation, data analysis, or brainstorming.

Key Components

Setup & Configuration

- Dependencies: Install google-generativeai and google-api-core for API connectivity.
- API Key Management: Securely retrieves the Google API key (best practice: store in environment variables).
- Model Initialization: Configures the Gemini client with a custom endpoint (optional) and initializes the gemini-1.5-flash-001 model.

Code Breakdown

Install & Import Libraries

In [1]:
```
# Install dependencies (uncomment if needed)
# !pip install google-generativeai google-api-core
import os
import google.generativeai as genai
from google.api_core import client_options as client_options_lib
```

Configure Gemini API

In [2]:
```
# Load API key from environment variables (ensure security)
os.environ["GOOGLE_API_KEY"] = "your_api_key" # Replace with your API key
genai.configure(
    api_key=os.getenv("GOOGLE_API_KEY"),
    transport="rest",
    client_options=client_options_lib.ClientOptions()
)

# Initialize the Gemini model
model_flash = genai.GenerativeModel('gemini-1.5-flash-001')
```

Define Generation Functions

In [3]:
```
def flash(prompt, model=model_flash, temperature=0.8):
    """Generates a response using Gemini AI model."""
    try:
        return model.generate_content(prompt, generation_config={'temperature': temperature})
    except Exception as e:
        print("Error generating response:", e)
        return None

def prompt(feed, temp = 0.8):
    """Passes user input to Gemini and prints the response."""
    response = flash(feed, model=model_flash, temperature= temp)
    if response:
        print(response.text)
```

Execute Example

In [4]:
```
# Test case: Generating a rhythmic poem
prompt("Write a rhythmic poem on 'Hello World'")
```

Hello World

Hi-lo, World, a sound so bright,
Spark of code, a shining light.
The first words spoken, clear and bold,
A story now, a tale untold.

From lines of code, a message flows,
A symphony of bits, a gentle prose.
The screen awakes, a canvas wide,
Where dreams take flight, and thoughts reside.

Hi-lo, World, a greeting warm,
A welcome to a digital storm.
A universe of data, vast and deep,
Where knowledge grows, and secrets keep.

So let us sing, with joy and pride,
Hi-lo, World, our digital guide.
May every line, with grace and ease,
Bring forth a world, where all can seize.

Best Practices and Examples

1. Experimentation with Temperature

To balance creativity and precision, adjust the temperature parameter:

In [5]:
```
# Low temperature for deterministic output
prompt("Briefly explain Newton's laws of motion.", temp =0.1)
print("-----------------------------------------------------------------------------")
# High temperature for creative storytelling
prompt("Write a fantasy story about a time-traveling cat.", temp=0.9)
```

Newton's Laws of Motion in a Nutshell:

1. Law of Inertia: An object at rest stays at rest, and an object in motion stays in motion at a constant velocity, unless acted upon by a net force. Think of a ball sitting still, or a car moving at a steady speed on a straight road.

2. Law of Acceleration: The acceleration of an object is directly proportional to the net force acting on it and inversely proportional to its mass. This means a bigger force makes things speed up or slow down faster, and heavier objects are harder to move.

3. Law of Action-Reaction: For every action, there is an equal and opposite reaction. Think of jumping: you push down on the ground, and the ground pushes back up on you, propelling you into the air.

These three laws form the foundation of classical mechanics, explaining how objects move and interact with each other.

Whiskers twitched, a faint glow emanating from his emerald eyes, Midnight the tabby prepared for his latest adventure. He wasn't just any cat; he was a time traveler, a guardian of history, with a magic collar that shimmered with celestial energy. His latest mission: prevent the theft of the legendary Sunstone, a crystal said to hold the power of a thousand suns.

He materialized in a bustling marketplace, the air thick with the scent of spices and the cacophony of hawkers. The year was 1485, the place, the sprawling city of Alexandria. He recognized the thief, a slithering shadow known as Scarab, notorious for his cunning and swiftness. Scarab, with eyes like molten gold, was already making his way towards the Temple of Ra, where the Sunstone lay guarded.

Midnight, a blur of ginger fur, slipped through the crowded streets, his senses alert. He reached the temple just as Scarab, cloaked in darkness, was about to break through the temple's ornate doors. He hissed, a sound that echoed with surprising power, startling the thief.

"Who dares disturb Scarab's plans?" Scarab hissed back, his voice a chilling whisper.

"A guardian of time," Midnight retorted, his fur bristling. "The Sunstone will not be stolen on my watch."

Scarab, dismissive, lunged. But Midnight was too quick, dodging the attack with agility born of centuries of experience. He leapt onto a pillar, a mischievous glint in his eyes.

"You may be fast, Scarab, but I am faster," he meowed, his voice echoing with the ancient wisdom of ages.

The ensuing chase was a whirlwind of feline grace and shadowy cunning. Midnight, utilizing the architectural nuances of the temple and his feline agility, kept Scarab at bay. He had the thief on a dizzying pursuit, through intricate corridors and hidden chambers, finally cornering him in the sanctum where the Sunstone rested.

Scarab, cornered and desperate, unleashed a burst of dark magic. The chamber shook, and the air crackled with sinister energy. But Midnight, undeterred, faced him with a newfound intensity. He closed his eyes, focusing on the collar's magic, its celestial energy surging through him.

With a powerful leap, he unleashed a wave of pure light, dispelling the dark magic and throwing Scarab back. The thief, weakened and defeated, fled into the shadows.

Midnight, his chest heaving, approached the Sunstone, a radiant orb pulsing with golden light. He felt its warmth, its power, and understood the responsibility entrusted to him. He knew he would return to his own time, but the memory of this adventure, of protecting the Sunstone and safeguarding history, would forever be etched in his feline heart.

As the sun rose, casting a golden glow over Alexandria, Midnight, the time-traveling tabby, vanished into the swirling mists of time, leaving behind a legacy of bravery and the whispered whisper of a purr, a testament to his unwavering commitment to protecting the threads of time. He knew, as he disappeared into the ethereal vortex, that his journey, and his duty, were far from over. He was Midnight, the guardian of time, and he would always be there, a silent protector in the shadows of history.

2. Scalability for Other Use Cases

This framework can be extended beyond simple text generation:

- Code Generation

In [6]:
```
def generate_code(prompt, model=model_flash, temperature=0.2):
    """Generates high-quality code snippets using Gemini AI."""
    code_prompt = f"Write a well-structured and efficient code snippet for: {prompt}"
    response = flash(code_prompt, model, temperature)
    if response:
        print(response.text)
    else:
        print("Error: Unable to generate code. Please try again.")

# Usage
generate_code("Write a Python function to sort a list using merge sort.")
```

```python
def merge_sort(arr):
    """
    Sorts a list using the merge sort algorithm.

    Args:
        arr: The list to be sorted.

    Returns:
        The sorted list.
    """
    if len(arr) <= 1:
        return arr

    mid = len(arr) // 2
    left_half = arr[:mid]
    right_half = arr[mid:]

    left_half = merge_sort(left_half)
    right_half = merge_sort(right_half)

    return merge(left_half, right_half)

def merge(left, right):
    """
    Merges two sorted lists into a single sorted list.

    Args:
        left: The first sorted list.
        right: The second sorted list.

    Returns:
        The merged sorted list.
    """
    merged = []
    i = 0
    j = 0

    while i < len(left) and j < len(right):
        if left[i] <= right[j]:
            merged.append(left[i])
            i += 1
        else:
            merged.append(right[j])
            j += 1

    while i < len(left):
        merged.append(left[i])
        i += 1

    while j < len(right):
        merged.append(right[j])
        j += 1

    return merged

# Example usage
my_list = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]
sorted_list = merge_sort(my_list)
print(sorted_list)  # Output: [1, 1, 2, 3, 3, 4, 5, 5, 5, 6, 9]
```

Explanation:

1. **`merge_sort(arr)` function:**
    - **Base case:** If the list has 0 or 1 element, it's already sorted, so return it.
    - **Divide:** Split the list into two halves (left and right).
    - **Conquer:** Recursively call `merge_sort` on both halves to sort them.
    - **Combine:** Call the `merge` function to merge the sorted halves into a single sorted list.

2. **`merge(left, right)` function:**
    - Creates an empty list `merged` to store the results.
    - Uses two pointers `i` and `j` to iterate through the `left` and `right` lists, respectively.
    - Compares elements from both lists and appends the smaller element to `merged`.
    - After one list is exhausted, appends the remaining elements from the other list to `merged`.
    - Returns the `merged` list.

Key points:

- **Recursion:** The `merge_sort` function uses recursion to divide the list into smaller subproblems.
- **Divide and Conquer:** The algorithm follows the divide-and-conquer paradigm.
- **Stable:** Merge sort is a stable sorting algorithm, meaning that elements with equal values maintain their relative order in the sorted list.
- **Time Complexity:** O(n log n) in all cases.
- **Space Complexity:** O(n) due to the extra space used for merging.

In [7]:
```
def enhanced_generate_code(prompt, model=model_flash, temperature=0.2):
    """Generates structured code with additional context."""
    code_prompt = (
        f"You are a professional AI assistant that writes optimized and well-commented code. "
        f"Please write an efficient solution for: {prompt}. "
        f"Ensure best practices such as modularity, readability, and performance."
    )
    response = flash(code_prompt, model, temperature)
    if response:
        print(response.text)
    else:
        print("Error: Code generation failed. Try refining your prompt.")

enhanced_generate_code("Write a Python function to find the nth Fibonacci number using recursion.")
```

```python
def fibonacci(n: int) -> int:
    """
    Calculates the nth Fibonacci number using recursion.

    Args:
        n: The index of the Fibonacci number to calculate (starting from 0).

    Returns:
        The nth Fibonacci number.

    Examples:
        fibonacci(0) == 0
        fibonacci(1) == 1
        fibonacci(5) == 5
    """
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)

# Example usage
print(fibonacci(5))  # Output: 5
```

Explanation:

1. **Function Definition:**
    - `def fibonacci(n: int) -> int:` defines a function named `fibonacci` that takes an integer `n` as input and returns an integer. The type hints (`: int`) improve code readability and help with static analysis.

2. **Base Cases:**
    - `if n == 1:` checks for the base cases where the Fibonacci sequence starts:
        - `return 0` returns 0 for `n = 0` and 1 for `n = 1`.

3. **Recursive Step:**
    - `else:` handles the recursive case for `n > 1`:
        - `return fibonacci(n-1) + fibonacci(n-2)` recursively calls the `fibonacci` function with `n-1` and `n-2` to calculate the previous two Fibonacci numbers and adds them together.

Modularity:
- The function is self-contained and performs a single, well-defined task: calculating the nth Fibonacci number.

Readability:
- Clear function name (`fibonacci`) and docstring explaining the function's purpose, arguments, return value, and examples.
- Type hints (`: int`) improve code readability and help with static analysis.
- Indentation and spacing enhance code readability.

Performance:
- While recursive solutions are elegant, they can be inefficient for large values of `n` due to repeated calculations. For performance-critical applications, consider using an iterative approach or memoization to avoid redundant computations.

Note: This recursive solution is primarily for educational purposes. For practical applications, consider using an iterative approach or memoization to improve performance.

- Data Summarization

In [8]:
```
def summarize_text(prompt, model=model_flash, temperature=0.3):
    """Summarizes the given text using Gemini AI."""
    summary_prompt = f"Summarize the following text concisely: {prompt}"
    response = flash(summary_prompt, model, temperature)
    if response:
        print(response.text)
    else:
        print("Failed to generate summary. Try again.")

summarize_text("Summarize the key takeaways from the latest AI research paper.")
```

Please provide me with the text of the AI research paper you'd like me to summarize. I need the content of the paper to be able to extract the key takeaways.

- AI-powered Q&A System

In [9]:
```
def generate_answer(prompt, model=model_flash, temperature=0.3):
    """Generates an answer for the given question using Gemini AI."""
    qa_prompt = f"Provide a detailed and accurate answer for: {prompt}"
    response = flash(qa_prompt, model, temperature)
    if response:
        print(response.text)
    else:
        print("Failed to generate an answer. Try again.")

generate_answer("What are the benefits of deep learning in AI?")
```

Benefits of Deep Learning in AI

Deep learning, a subset of machine learning, has revolutionized AI by offering significant advantages over traditional methods. Here's a breakdown of its key benefits:

1. Superior Performance in Complex Tasks:

* **Image Recognition:** Deep learning excels in tasks like object detection, image classification, and facial recognition, surpassing traditional methods in accuracy.
* **Natural Language Processing (NLP):** Deep learning models like BERT and GPT-3 have significantly improved machine translation, text summarization, sentiment analysis, and chatbot performance.
* **Speech Recognition:** Deep learning models can accurately transcribe speech, enabling voice assistants and speech-to-text applications.
* **Predictive Modeling:** Deep learning can analyze vast datasets to predict future trends, enabling better decision-making in areas like finance, healthcare, and marketing.

2. Automation of Feature Engineering:

* Deep learning models automatically learn relevant features from raw data, eliminating the need for manual feature engineering. This saves time and effort, particularly in complex datasets.

3. Handling High-Dimensional Data:

* Deep learning can effectively handle large datasets with numerous features, making it suitable for analyzing complex real-world data.

4. Adaptability and Generalization:

* Deep learning models are highly adaptable and can be trained on diverse datasets, enabling them to generalize well to new, unseen data.

5. Continuous Learning and Improvement:

* Deep learning models can continuously learn and improve their performance by being exposed to new data through techniques like transfer learning and fine-tuning.

6. End-to-End Learning:

* Deep learning allows for end-to-end learning, where the model learns all aspects of a task from raw input to output, eliminating the need for separate modules.

7. Reduced Human Intervention:

* Deep learning automates many tasks, reducing the need for human intervention and enabling faster and more efficient solutions.

8. Enhanced User Experience:

* Deep learning powers personalized experiences in various applications, like recommendation systems, personalized search results, and targeted advertising.

9. Innovation and New Possibilities:

* Deep learning has opened up new possibilities in AI, leading to advancements in areas like autonomous vehicles, medical diagnosis, and drug discovery.

However, it's important to note that deep learning also has limitations:

* **Data Requirements:** Deep learning models require massive amounts of data for training, which can be a challenge in some domains.
* **Computational Resources:** Training and deploying deep learning models can be computationally expensive, requiring specialized hardware and infrastructure.
* **Black Box Nature:** Deep learning models can be difficult to interpret, making it challenging to understand their decision-making process.
* **Bias and Fairness:** Deep learning models can inherit biases from the training data, leading to unfair or discriminatory outcomes.

Despite these limitations, the benefits of deep learning in AI are undeniable. Its ability to solve complex problems, automate tasks, and drive innovation makes it a powerful tool for advancing the field of artificial intelligence.

Best Practices:

- Provide specific and detailed prompts for better code quality.
- Adjust the temperature to control randomness in responses.
- Use structured prompts to enforce modularity and optimization in the generated code.

Conclusion

This project demonstrates how to integrate Google's Gemini AI into a Jupyter Notebook for text generation. Users can tailor responses for both creative and technical purposes by experimenting with prompt engineering and model settings. The framework is flexible, allowing for expansion to support a wide range of AI-driven applications, making it a valuable tool for both developers and content creators.