Generative AI Text Completion App

Objective
The goal of this project is to provide hands-on experience with Generative AI by building and experimenting with a simple text-completion application. You will:

- Understand how Generative AI processes input prompts to generate coherent, relevant text
- Set up and interact with a pre-trained AI model through an API
- Experiment with prompt design and model parameters to evaluate output quality
- Reflect on the capabilities and limitations of the AI model

---

Part 1: Building the Application

1. Environment Setup

• Ensure Python 3.8 or higher is installed
• Create and activate a virtual environment:
– Windows (PowerShell):
1. Run python -m venv venv
2. Then run .\venv\Scripts\Activate.ps1
– macOS/Linux:
1. Run python3 -m venv venv
2. Then run source venv/bin/activate
• Install required libraries by running:
pip install huggingface_hub python-dotenv

2. API Key Configuration

• Sign up at Hugging Face and create an access token with write permissions
• In the project root, create a file named .env containing:
HF_TOKEN=hf_your_api_token_here

3. Application Code

• Create a file called app.py that does the following:
– Loads environment variables (HF_TOKEN)
– Initializes the InferenceClient from huggingface_hub
– Enters a prompt loop to accept user text
– Parses and validates temperature and token-length inputs
– Calls text_generation on the model and prints the result
• To run the application, simply execute:
python app.py

Part 2: Debugging and Improving the Application

1. Error Handling

• Check for a missing or invalid HF_TOKEN and exit with a clear message
• Catch network, authentication or model-error exceptions and report them clearly
• Validate user inputs (empty prompt, non-numeric temperature or length) and fall back to defaults

2. API Usage

• Replaced the deprecated InferenceApi with the modern InferenceClient
• Use the text_generation method with two arguments:
– inputs: the user's prompt text
– parameters: a map containing max_new_tokens and temperature
• Ensure you parse both JSON-list and JSON-dict response shapes

3. Code Organization

• Encapsulate the core logic in a generate_text function with clear parameters
• Keep the REPL loop (prompt → generate → display) in main()
• Add comments and simple docstrings for maintainability

Part 3: Experimentation and Evaluation

1. Prompts Tested

- Story continuation: Once upon a time, there was a robot who…
- Explanation: Explain photosynthesis to a 10-year-old.
- Poetry: Write a haiku about the ocean.
- Instructional: Explain recursion like I'm five.

2. Parameter Variations

- Temperature values tested: 0.0 and 1.0
- max_new_tokens tested: 20, 50 and 60

3. Logging and Analysis

• Recorded each prompt, parameter set and model response in a spreadsheet called experimentation_log.xlsx
• Reviewed outputs for coherence, creativity and factual accuracy

4. Reflection on Limitations

• **Strengths:** Creative, coherent continuations and concise explanations for simple topics
• **Repetition & Drift:** Tends to repeat phrases or veer off-topic on longer generations
• **Factual Simplification:** Oversimplifies or may misstate facts when asked for detailed accuracy
• **Logical Reasoning:** Struggles with multi-step or conditional reasoning tasks
• **Hallucinations:** Generates plausible-sounding but incorrect statements without external grounding

---

Conclusion

This end-to-end project shows how to set up a Python environment, call a Generative AI model via API, handle errors and edge cases, experiment with prompt design and parameters, log results, and reflect on real-world limitations. Future work could integrate retrieval-augmented generation, output filtering, or fact-checking modules to improve reliability.