Technical Interview Question: Building a Multi-Functional Console Application with AI Integration

Objective:

Create a complex console application that connects to a free language model API to perform multiple text-based tasks: text summarization, translation, text expansion, and fact-checking.

Description:

Your task is to build a console application that can perform several text-based tasks using a free language model API. The application should allow the user to choose between summarizing text, translating text to another language, expanding text, and fact-checking text.

Requirements:

- The application should be written in a language of your choice (Python, JavaScript, etc.).
- The application should prompt the user to select a task: summarization, translation, expansion, or fact-checking.
- Use a free language model API (such as Cohere, Hugging Face Inference API, etc.) to perform the selected task.
- Display the result of the selected task to the user in the console.

Steps to Complete the Task:

1. **Set up the project:**
    a. Initialize a new project and set up any necessary dependencies.
    b. Ensure you have access to an API key for the chosen free language model API.
2. **Task Selection:**
    a. Implement functionality to allow the user to select a task from the following options:
        i. Summarize text
        ii. Translate text to another language
        iii. Expand text
        iv. Fact-check text
3. **Input Handling:**

    a. For summarization, expansion, and fact-checking, capture a block of text input from the user through the console.
    b. For translation, capture a block of text input and the target language.
4. **API Integration:**
    a. Connect to the language model API using the appropriate client library or HTTP requests.
    b. Send the user's input to the API and request the appropriate operation (summarization, translation, expansion, or fact-checking).
5. **Output Handling:**
    a. Capture the response from the API.
    b. Display the result to the user in the console.
6. **Error Handling:**
    a. Ensure the application gracefully handles any errors, such as network issues or invalid API responses.

Example:

Here's an example of how the application should work:

1. User runs the console application.
2. Application prompts the user to select a task:

```vbnet
Please select a task:
1. Summarize text
2. Translate text to another language
3. Expand text
4. Fact-check text
```

3. User selects a task (e.g., Summarize text).
4. Application prompts the user to enter a block of text:

```vbnet
Please enter the text you want to summarize:
```

5. User inputs the text:
   "Artificial intelligence (AI) is intelligence demonstrated by machines, in contrast to the natural intelligence displayed by humans and animals. Leading AI textbooks define the field as the study of 'intelligent agents': any device that

perceives its environment and takes actions that maximize its chance of successfully achieving its goals."

6. The application sends this text to the language model API for summarization.
7. The API returns the summarized text.
8. The application displays the summarized text in the console:
   "AI is the intelligence shown by machines, contrasting with human and animal intelligence. It involves intelligent agents that perceive their environment and take actions to achieve their goals."

Additional Scenarios:

Translation:

- Application prompts the user to enter the text and the target language.
- User inputs the text and the target language (e.g., ***"Translate to Spanish: 'Hello, how are you?'"***).
- The application sends this data to the API for translation.
- The API returns the translated text.
- The application displays the translated text in the console (e.g., "Hola, ¿cómo estás?").

Text Expansion:

- Application prompts the user to enter a block of text.
- User inputs the text (e.g., ***"The future of AI is promising."***).
- The application sends this text to the API for expansion.
- The API returns the expanded text.
- The application displays the expanded text in the console (e.g., ***"The future of AI is promising, with advancements in machine learning and neural networks driving significant progress in various fields such as healthcare, finance, and transportation."***).

Fact-Checking:

- Application prompts the user to enter a statement to fact-check.
- User inputs the statement (e.g., ***"The Earth is flat."***).
- The application sends this statement to the API for fact-checking.
- The API returns the fact-checked information.
- The application displays the fact-checked information in the console (e.g., ***"False: The Earth is an oblate spheroid, meaning it is mostly spherical but slightly flattened at the poles and bulging at the equator."***).

Resources:

- Cohere API documentation: (https://docs.cohere.ai/)
- Hugging Face Inference API documentation: (https://huggingface.co/docs/api-inference/)
- Example client libraries for making HTTP requests: `requests` for Python, `axios` for JavaScript, etc.

Evaluation Criteria:

- **Correctness**: Does the application correctly capture input, interact with the API, and display the output for each task?
- **Code Quality**: Is the code clean, well-organized, and easy to understand?
- **Error Handling**: Does the application handle errors gracefully and inform the user of any issues?
- **User Experience**: Is the application easy to use and does it provide clear instructions and feedback?

Bonus:

- Add functionality to allow the user to input multiple paragraphs and summarize each one separately.
- Implement a feature to save the output (summarized text, translated text, expanded text, or fact-checked information) to a file.