

Local AI Assistant Documentation

Authors:

- Akarsh Raj M Hulihalli (akarshraj2510@gmail.com)
- Chandan G D (chandangd27@Gmail.Com)

1. Project Overview

JnyaAI is a sophisticated, command-line based AI assistant that runs entirely on a local machine. It leverages the power of local Large Language Models (LLMs) through the [Ollama](#) framework, ensuring that all interactions are private, secure, and do not require an internet connection.

The name "JnyaAI" is inspired by the Sanskrit word *jñāna* (ज्ञान), which means knowledge, wisdom, and insight. The assistant is designed to be a "wise" and helpful companion, capable of answering questions, generating creative text, summarizing information, and more.

The user interface is built using the rich library, providing a modern, colourful, and user-friendly experience directly in the terminal.

2. Key Features

- **100% Local and Private:** All processing is done on the user's machine. No data is sent to external servers, guaranteeing absolute privacy.
- **Offline Capability:** Since no internet is required after initial setup, JnyaAI can be used anywhere, anytime.
- **Conversational Context:** The assistant remembers the history of the current conversation, allowing for natural, follow-up questions and contextual understanding.
- **Streaming Responses:** JnyaAI streams its responses token-by-token, creating a "live typing" effect that enhances user experience and provides immediate feedback.
- **Rich CLI Interface:** Utilizes the rich library for colored text, spinners (e.g., "JnyaAI is thinking..."), and clean formatting, making the interaction more engaging than a standard terminal program.
- **Customizable Persona:** A system prompt is used to define JnyaAI's personality as a "highly intelligent and wise AI assistant," which guides the tone and style of its responses.
- **Graceful Error Handling:** The application handles common issues like user interruptions (Ctrl+C) and other potential errors gracefully.
- **Standalone Executable:** Includes a command to package the application into a single executable file using PyInstaller for easy distribution and use without needing a Python environment.

3. Dependencies

To run JnyaAI, you need the following software installed:

1. **Ollama Application:** The core framework that serves the local LLM.

- Download from: <https://ollama.com/>
- 2. **A Local LLM (e.g., tinylama):** A model file that Ollama will run. The script is configured to use tinylama by default due to its small size and fast performance.
- 3. **Python 3:** The programming language the script is written in.
- 4. **Python Libraries:**
 - ollama: The official Python client for interacting with the Ollama API.
 - rich: For creating beautiful and readable terminal user interfaces.

4. Setup and Execution

Follow these steps to get JnyaAI running on your machine.

Step 1: Install and Run Ollama

Download and install the Ollama application for your operating system (macOS, Windows, or Linux). After installation, ensure the Ollama application is running in the background.

Step 2: Pull the Language Model

Open your terminal and run the following command to download the tinylama model. This only needs to be done once.

Generated sh

```
ollama pull tinylama
```

Step 3: Install Required Python Libraries

In your terminal, install the ollama and rich libraries using pip.

Generated sh

```
pip install ollama rich
```

Step 4: Run the JnyaAI Script

Save the provided code as JnyaAI.py. Navigate to the directory where you saved the file in your terminal and run the script:

Generated sh

```
python JnyaAI.py
```

You should now see the welcome message and be able to chat with JnyaAI. Type exit to end the session.

5. Building a Standalone Executable

You can package JnyaAI into a single executable file that can be run on other computers without needing to install Python or any libraries.

Step 1: Install PyInstaller

Generated sh

```
pip install pyinstaller
```

Step 2: Build the Executable

Run the following command in the directory containing JnyaAI.py. The --onefile flag bundles everything into a single file.

Generated sh

```
pyinstaller --onefile JnyaAI.py
```

After the process completes, you will find the executable file inside a new dist folder.

6. Code Breakdown

This section provides a detailed explanation of the JnyaAI.py script.

Imports

Generated python

```
import ollama
```

```
from rich.console import Console
```

```
from rich.markdown import Markdown
```

- `ollama`: The client library to communicate with the running Ollama service.
- `rich.console.Console`: The main object for printing rich text, spinners, and handling input.
- `rich.markdown.Markdown`: Although not used in the final script, it's available for rendering Markdown-formatted text.

initialize() function

```
def initialize():  
    console = Console()  
    console.print("[bold green]JnyaAI: your Assistant (Local Edition) [/bold green]")  
    # ... welcome messages ...  
    return console, []
```

This function sets up the application environment.

- It creates a `Console` object for all UI interactions.
- It prints the initial welcome banner with styling.
- It returns the console object and an empty list `[]`, which will be used to store the conversation history.

main() function

This is the core logic of the application.

1. Initialization and System Prompt:

```
console, messages = initialize()  
messages.append({  
    'role': 'system',
```

```

        'content': (
            "You are JnyaAI, a highly intelligent and wise AI assistant..."
        ),
    })

```

- It calls `initialize()` to get the console and the message list.
- Crucially, it adds a **system prompt** to the messages list. This message is not displayed to the user but is sent to the LLM with every request. It instructs the AI on its persona and behaviour, shaping its responses to be "wise" and "thoughtful."

2. The Main Loop:

```

while True:
    # ... conversation logic ...

```

An infinite loop keeps the chat session active until the user decides to exit.

3. User Input and Exit Condition:

```

user_input = console.input("[bold blue]You: [/bold blue]")
if user_input.lower() in ['exit', 'quit']:
    # ... exit message ...
    break
messages.append({'role': 'user', 'content': user_input})

```

- It prompts the user for input with styled text.
- It checks for "exit" or "quit" commands to break the loop.
- The user's input is appended to the messages list with the role 'user'. This adds the user's turn to the conversation history.

4. Calling the LLM and Streaming the Response:

```

full_response = ""
with console.status("[bold green]JnyaAI is thinking...[/bold green]", spinner="dots"):
    stream = ollama.chat(
        model='tinyllama',
        messages=messages,
        stream=True,
    )
console.print("\n[bold green]JnyaAI:[/bold green]", end="")
for chunk in stream:
    part = chunk['message']['content']
    console.print(part, end="")
    full_response += part

```

- The `with console.status(...)` block displays a spinner while the model is processing the request.
- `ollama.chat()` is called to send the request to the LLM.
 - `model='tinylama'`: Specifies which model to use.
 - `messages=messages`: Sends the entire conversation history (including the system prompt) to provide context.
 - `stream=True`: This is key. It tells Ollama to send the response back in small chunks as it's being generated.
- The code then iterates through each chunk in the stream, printing it to the console immediately. This creates the real-time typing effect. The complete response is also accumulated in the `full_response` variable.

5. Updating Conversation History:

```
messages.append({'role': 'assistant', 'content': full_response})
```

After the full response has been received and printed, it is appended to the `messages` list with the role 'assistant'. This ensures that the AI's own previous response is part of the context for the next turn.

6. Exception Handling:

```
except KeyboardInterrupt:
    # ... handle Ctrl+C ...

except Exception as e:
    # ... handle other errors ...
```

This `try...except` block ensures the program doesn't crash. It catches `KeyboardInterrupt` (when the user presses Ctrl+C) and any other unexpected errors, printing a friendly message before exiting.

Script Execution Block

```
if __name__ == "__main__":
    main()
```

This is a standard Python construct that ensures the `main()` function is called only when the script is executed directly, not when it's imported as a module into another script.

Use examples:

JnyaAI: your Assistant (Local Edition)

Powered by a local model via Ollama(tinyllama). No internet required.

Ask me anything, or type 'exit' to end the session.

You: What is the capital of India?

JnyaAI:The capital of India is New Delhi.

You: summarise the text: "Weighing 2.392 kg, the Earth observation will be launched aboard the Geosynchronous Satellite Launch Vehicle Mark II, or GSLV Mk I I rocket for short, with liftoff scheduled for Wednesday, July 30 at 5:40 PM IST. You can watch the live launch of NISAR using the link below, or you can head over to NASA's website to tune in to the launch.

JnyaAI:NASA's Geosynchronous Satellite Launch Vehicle Mark II (GSLV Mk II) will be launched on July 30 with a payload named NISAR, which stands for Near-Infrared Imaging Spectrograph for Advanced Research. The launch window runs from 5:40 PM to 6:10 PM Indian Standard Time (IST).

The spacecraft, weighing 2.392 kg and measuring 3.5 meters in length, is expected to orbit Earth at an altitude of about 770 km above the planet's surface. This will allow NASA to study Earth in unprecedented detail through detailed observations using the NISAR instrument suite. The launch will be broadcast live on NASA's website.

You: give me a suggestion on how I can become more efficient in my daily task.

JnyaAI:Certainly! Here are some suggestions to help you become more efficient in your daily tasks:

1. Plan - Create a schedule or to-do list for each day and plan out the tasks to avoid getting overwhelmed.
2. Prioritize - Prioritize the most important tasks according to their importance, urgency, and availability.
3. Eliminate distractions - Disable notifications on your phone or computers, minimize unnecessary apps, and close all unnecessary tabs or windows.
4. Automate - Use automation tools to perform repetitive tasks such as scheduling emails, setting reminders for appointments, or automatically updating social media accounts.
5. Time-blocking - Schedule time blocks for each task and work on the most important tasks first, allowing for more time for less critical tasks later in the day.
6. Minimize wasted time - Reduce unnecessary interruptions and stay focused on your main task.
7. Delegate - Look for opportunities to delegate tasks that you are not able to complete or that require a specific skill set, such as organizing or taking notes.
8. Automate repetitive tasks - Use automation tools for tasks like cleaning, cooking, or laundry, freeing up more time for other important activities.
9. Take breaks - Take regular breaks throughout your day to recharge and refocus on your main task.
10. Stick with the plan - Follow through with your plan even if it means making changes based on new information or unexpected events. Remember, efficiency is not a goal in and of itself, but rather an outcome that can be achieved through proper planning and execution.

You: write a small poem in English about gardening.

JnyaAI:In the garden's verdant care,
Where blooms bloom and nurture grows,
A world of beauty unfolds,
In gardens of wonder never seen.

From bright green flowers that whisper,
To tall stalks of bluebells that stand,
A symphony of colors and fragrance,
In a garden's symphony of song.

Their fragrant scent enchants the air,
Bringing peace and quiet to mind,
As they bloom in a garden's bounty,
Our gardens bring us closer, oh so near.

Fresh breezes blow through the air,
A gentle whisper of nature's song,
Roses in bloom, peonies in full bloom,
In a garden's beauty, nothing else can be found.

Oh, how gardens fill our hearts with grace,
While blooms and blooming all around,
Our world of nature in its glory,
An endless tapestry of life so rare.
