Agentic AI Support – Documentation

Created by – Prashant Saxena

Github – https://github.com/p3rcyshots

# 1. Introduction

This document provides details about the **Agentic AI Support** system, which connects to an AI model supported by **Ollama** to analyze files in a specified folder and answer user queries based on the content. It supports multiple file formats and handles large files efficiently.

# 2. Features

* Reads and processes text from multiple file formats: **DOCX, PDF, TXT, XLSX, CSV**.
* Loads and analyzes large files (100,000+ words).
* Provides real-time user interaction with AI via the command line.
* Handles errors gracefully, including missing files and unsupported formats.
* Displays file statistics (number of files, character count, word count per file).
* Allows users to exit by typing **"exit" or "bye"** or pressing **Ctrl+C** (graceful termination).

# 3. Installation

## 3.1 Prerequisites

Ensure you have the following installed:

* Python 3.7 or later
* Pip package manager

## 3.2 Required Python Libraries

Install the required libraries using:

* pip install ollama pypdf2 python-docx pandas openpyxl

# 4. Usage

## 4.1 Running the AI Assistant

To start the assistant, use:

python supportai.py -l <ollama\_model>

Replace <ollama\_model> with the name of the supported Ollama model.

## 4.2 Interacting with the Assistant

After running the script, you can enter queries as:

* AI Assistant: Summarize the content of all files
* Type exit or bye to quit.

## 4.3 Folder Path Configuration

The script reads files from the folder specified in **folder.txt**. Ensure folder.txt contains the absolute path of the folder with files.

# 5. File Handling and Processing

The AI assistant loads files from the specified folder and reads content based on file format:

* **TXT**: Reads the entire text.
* **DOCX**: Extracts text from paragraphs and tables.
* **PDF**: Extracts text from all pages.
* **CSV/XLSX**: Converts data into a readable text format.

# 6. Error Handling

The program includes robust error handling:

* **Missing folder.txt** → Displays an error and exits.
* **Invalid folder path** → Alerts the user and exits.
* **Unsupported file formats** → Skips them and proceeds.
* Ctrl+C interruption → Gracefully exits with the message "Aborted by the user."

# 7. Test Cases

## 7.1 Basic Functionality

|  |  |  |
| --- | --- | --- |
| Test Case | Input | Expected Output |
| Load TXT File | sample.txt | Extracts text |
| Load DOCX File | sample.docx | Extracts paragraphs and tables |
| Load PDF File | sample.pdf | Extracts text from pages |
| Load Excel File | sample.xlsx | Converts tables to text |
| Exit Command | exit | Terminates program |

## 7.2 Edge Cases

|  |  |  |
| --- | --- | --- |
| Test Case | Input | Expected Output |
| No Files in Folder | Empty directory | "No readable files found." |
| Large File (100,000+ words) | large.docx | Loads in chunks and processes |
| Unsupported File | sample.xyz | Skips file, no crash |
| Ctrl+C | KeyboardInterrupt | "Aborted by the user." |

# 8. Conclusion

The **Agentic AI Support** provides an efficient way to analyze large documents and interact with an AI model seamlessly. It ensures reliable error handling and smooth performance for handling multiple large files simultaneously.

For any further enhancements, feel free to update the script accordingly.

# Complete Code

## Code Start

import os

import sys

import signal

import argparse

import ollama

import PyPDF2

import docx

import pandas as pd

from pathlib import Path

# Graceful exit on Ctrl+C

def signal\_handler(sig, frame):

print("\nAborted by the user.")

sys.exit(0)

signal.signal(signal.SIGINT, signal\_handler)

# Read folder path from folder.txt

def get\_folder\_path():

try:

with open("folder.txt", "r") as f:

return f.read().strip()

except FileNotFoundError:

print("Error: folder.txt not found.")

sys.exit(1)

except Exception as e:

print(f"Error reading folder.txt: {e}")

sys.exit(1)

# Read text from different file formats efficiently

def read\_file\_content(file\_path):

ext = file\_path.suffix.lower()

try:

if ext == ".txt":

with open(file\_path, "r", encoding="utf-8") as f:

return f.read()

elif ext == ".docx":

doc = docx.Document(file\_path)

text = "\n".join([para.text for para in doc.paragraphs])

for table in doc.tables:

for row in table.rows:

text += "\n" + " | ".join(cell.text for cell in row.cells)

return text

elif ext in [".xlsx", ".csv"]:

df = pd.read\_excel(file\_path) if ext == ".xlsx" else pd.read\_csv(file\_path)

return df.to\_string()

elif ext == ".pdf":

with open(file\_path, "rb") as f:

reader = PyPDF2.PdfReader(f)

text = "\n".join([page.extract\_text() or "" for page in reader.pages])

return text

else:

return None

except Exception as e:

print(f"Error reading {file\_path.name}: {e}")

return None

# Load all file contents efficiently

def load\_files(folder):

folder\_path = Path(folder)

if not folder\_path.exists() or not folder\_path.is\_dir():

print("Error: Specified folder does not exist.")

sys.exit(1)

content = []

file\_stats = []

for file in folder\_path.iterdir():

file\_text = read\_file\_content(file)

if file\_text:

char\_count = len(file\_text)

word\_count = len(file\_text.split())

file\_stats.append((file.name, char\_count, word\_count))

content.append(f"\n\n--- {file.name} ---\n{file\_text}")

if not content:

print("⚠️ No readable content found in files.")

return "No readable files found."

full\_content = "\n".join(content)

print(f"✅ Loaded {len(full\_content)} characters from {len(file\_stats)} files") # Debugging output

print("\nFile Statistics:")

for file\_name, char\_count, word\_count in file\_stats:

print(f"{file\_name}: {char\_count} characters, {word\_count} words")

return full\_content

# Main AI interaction loop

def main(ollama\_model):

folder = get\_folder\_path()

file\_data = load\_files(folder)

print("\nAI Assistant is ready. Type your query or 'exit' to quit.")

while True:

try:

user\_input = input("AI Assistant: ")

if user\_input.lower() in ["exit", "bye"]:

print("Goodbye!")

break

# Send file content in chunks to avoid overloading the model

chunk\_size = 5000 # Adjust based on model capabilities

chunks = [file\_data[i:i+chunk\_size] for i in range(0, len(file\_data), chunk\_size)]

responses = []

for chunk in chunks:

response = ollama.chat(model=ollama\_model, messages=[

{"role": "system", "content": "Analyze the given documents and respond based on user queries."},

{"role": "user", "content": chunk},

{"role": "user", "content": user\_input}

])

responses.append(response["message"]["content"])

print("\nResponse:\n", "\n".join(responses), "\n")

except Exception as e:

print(f"Error: {e}")

if \_\_name\_\_ == "\_\_main\_\_":

parser = argparse.ArgumentParser()

parser.add\_argument("-l", required=True, help="Specify Ollama AI model")

args = parser.parse\_args()

main(args.l)

## Code End

Code Ends Above