SEE PROJECT REPORT

MULTIMODAL VIRTUAL ASSISTANT (NVIDIA AI WORKBENCH)

Name of the course: Foundation Of Generative AI

Name of the institution: RV UNIVERSITY

Names of the author(s):

Karmishtha Patnaik [1RVU22CSE077], Renu Bojja [1RVU22CSE129]

Date of submission: 10 November 2024

Supervisor/Instructor’s Name: Prof. Manjul Krishna Gupta

CONTENT

1. Executive Summary

This project involves the development of a virtual product assistant on the NVIDIA AI Workbench, leveraging a multimodal Retrieval-Augmented Generation (RAG) pipeline and fallback websearch to assist users with troubleshooting, informing, and answering questions about the NVIDIA AI Workbench software. The project provides two user-facing applications: the Control-Panel for adding content to a local vector database and the Public-Chat, a simplified, shareable chatbot interface. The assistant can perform inference with NVIDIA cloud endpoints, self-hosted endpoints via NVIDIA Inference Microservices (NIM), or third-party microservices, allowing flexible integration. This customizable setup is intended to improve access to AI Workbench documentation, user-generated content, and help resources, making it versatile and accessible for various user needs.

2. Introduction

2.1 Background and Context

The NVIDIA AI Workbench is a powerful platform designed for AI model deployment, management, and execution. However, its advanced features and multifaceted workflows can present challenges for users, particularly those requiring immediate guidance or technical support. This project addresses the need for an intelligent virtual assistant capable of guiding users through queries and troubleshooting related to the AI Workbench.

2.2 Objective and Scope

We aim to build an adaptable virtual product assistant that helps users navigate NVIDIA AI Workbench through a blend of multimodal RAG pipelines and websearch fallback. The project includes two distinct interfaces: a Control-Panel for content addition and a read-only Public-Chat for user engagement. While focused on the AI Workbench, this framework is adaptable to other NVIDIA products with similar technical assistance needs.

2.3 Problem Statement

Users frequently encounter complex technical issues or need detailed guidance within the NVIDIA AI Workbench, leading to delays and support inefficiencies. An intelligent virtual assistant with a RAG-based approach can enhance user experience by offering instant, context-aware responses.

2.4 Overview of Solution

By integrating a multimodal RAG pipeline with fallback websearch, the assistant can answer queries using local knowledge and external sources. Built on LangGraph, the pipeline directs queries through a LLM-based router to either RAG or websearch, providing accurate responses and maintaining user context.

3. Literature Review

3.1 Review of the Literature and Technologies

Past projects in virtual assistance and RAG-based systems have shown that integrating knowledge bases with generative models enhances support quality. Tools like OpenAI's ChatGPT and Google's BERT have advanced contextual assistance but often struggle with content-specificity for product-focused tasks.

NVIDIA NIM™ offers a solution by providing GPU-accelerated inferencing microservices for pretrained and custom AI models. These microservices, deployed with a single command, expose industry-standard APIs, simplifying integration into AI applications. Built on optimized engines like TensorRT™ and TensorRT-LLM, NIM ensures optimal latency and throughput, with autoscaling support on Kubernetes. This makes it easier for developers to self-host AI models and build powerful assistants at scale.

Mistral-7B-Instruct, a fine-tuned version of the Mistral-7B model, excels in following instructions and generating creative text. While not developed by NVIDIA, it enhances AI's ability to execute complex requests in conversational systems.

3.2 Distinguishing Project from Existing System

Unlike general-purpose assistants, this project is focused on the NVIDIA AI Workbench, adding tailored features for troubleshooting and software navigation. Existing tools lack integration of multimodal data and self-hosted inference options, both critical for product-specific virtual assistance. This project aims to provide a more targeted, flexible solution, enhanced by self-hosted NIM capabilities.

4. System Requirements And Specifications

4.1 Functional Requirements

1. Ingest documents (web pages, PDFs, images, videos) into a vector database.
2. Answer user questions based on the ingested documents.
3. Provide relevant documents alongside answers.
4. Allow users to clear the document database.
5. Offer a user interface for interaction.
6. Use an agentic workflow to evaluate and refine answers.

4.2 Non-Functional Requirements

1. Reasonable response times for queries. (Specific targets aren't defined in the code.)
2. Intuitive user interface for both chatting and knowledge base management.
3. Implied scalability through the use of LanceDB

4.3 Software and Hardware Requirements

A. **Software**
   a. Python 3 is the primary programming language.
   b. Specific library dependencies are listed in requirements.txt. These include libraries for machine learning (Torch, Torchvision, scikit-image), natural language processing (LlamaIndex, LangChain, ftfy, regex), vector databases (LanceDB), web scraping and PDF processing (Unstructured, PyTube, MoviePy), audio processing (PyDub, SpeechRecognition), image processing

(Pillow), OCR (Tesseract), user interface development (Gradio), web server development (FastAPI, Uvicorn), and other utilities (tqdm).

c. NVIDIA drivers are required for leveraging GPUs for embedding generation.

d. FFmpeg must be installed in the operating system and in the PATH.

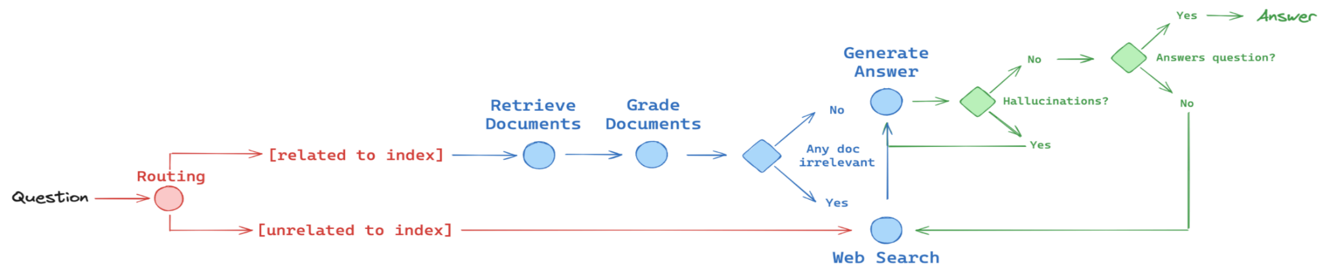
B. **Hardware**

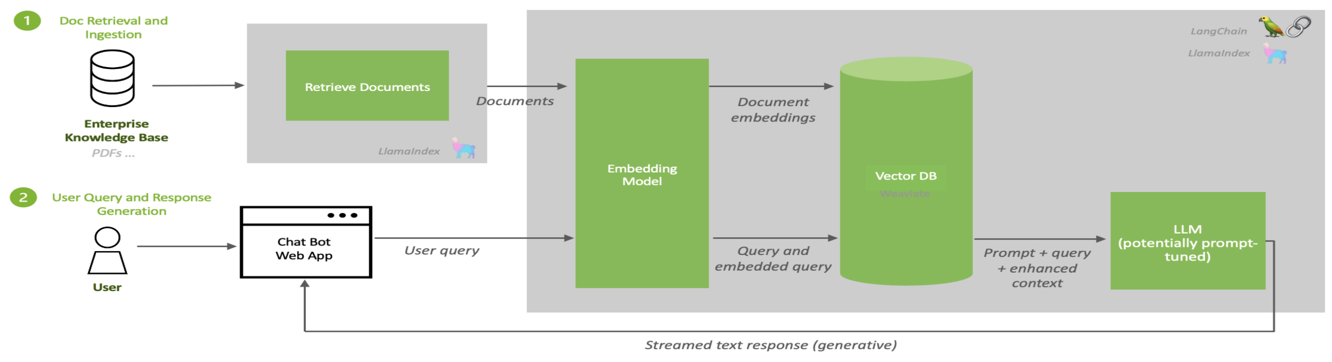
The system should meet the computational requirements of the specified software. A GPU is strongly recommended to accelerate embedding generation with NVIDIA Embeddings. The RAM and storage capacity should be sufficient to hold the vector database and ingested documents. The specific requirements depend on the size of the knowledge base and expected usage.

5. System Design

5.1 Architecture Diagram

The system follows a client-server architecture. The Gradio-based user interface acts as the client, communicating with a FastAPI backend server. The backend server handles query processing, document retrieval from LanceDB, interaction with LLMs (either through APIs like ChatNVIDIA or through NIMs for local inference), and response generation.




1 Doc Retrieval and Ingestion
Enterprise Knowledge Base
PDFs ...
Retrieve Documents
LlamaIndex
Documents
Embedding Model
Document embeddings
Vector DB
LangChain
LlamaIndex
2 User Query and Response Generation
User
Chat Bot Web App
User query
Query and embedded query
Prompt + query + enhanced context
LLM (potentially prompt-tuned)
Streamed text response (generative)

5.2 Detailed Design

A. Modules and Components

a. **chatui_public**: This module encompasses the client-side and server-side components of the application.
    i. pages/converse.py: Defines the Gradio interface for user interaction, including the chat window, knowledge base management controls, and model/prompt configuration options.
    ii. api.py: Implements the FastAPI backend server, handling API requests from the client, and orchestrating the RAG workflow.
    iii. chat_client.py: Provides a client class for communicating with the FastAPI backend, abstracting API calls for document search and answer generation.
    iv. configuration.py, configuration_wizard.py: Manage application configuration by loading parameters from files or environment variables.
    v. utils: Contains utility modules for different aspects of the system.

b. **utils**: This module contains utility submodules:
    i. database.py: Handles interactions with the LanceDB vector database, including document ingestion, retrieval, and database clearing.
    ii. compile.py: Compiles the LangGraph workflow defined in graph.py.
    iii. nim.py: Provides a custom LangChain class for interacting with LLMs hosted as NIMs.
    iv. logger.py: Logs script progress for the user to monitor progress.

c. **prompts**: Stores prompt templates used for different LLMs and different stages of the agentic workflow. prompts_llama3.py and prompts_mistral.py contain templates for Llama 2 and Mistral models respectively.

d. **assets**: Contains static assets, such as the Kaizen theme for the Gradio interface.

B. Database Design

LanceDB is used as a vector database. It implicitly handles vector storage and similarity search. Separate tables ("collections" in LanceDB terminology) are used for web pages (web_collection), PDFs (pdf_collection), and combined text/image data (text_img_collection). A table for image embeddings is also included (image_collection). The database schema is managed by LanceDB internally.

C. User Interface Design

The Gradio interface provides a user-friendly chat window for interacting with the bot. It also includes tabs for configuring models, prompts, and managing the knowledge base. Collapsible sections and clear labels improve usability.

5.3 Data Flow

a. **User Input:** The user enters a query in the Gradio chat interface.
b. **Client Request:** The Gradio client sends an API request to the FastAPI backend.
c. **Workflow Execution:** The backend server receives the query and initiates the compiled LangGraph workflow.
d. **Query Routing:** The workflow's router determines whether to use the vector database or web search based on the query.
e. **Document Retrieval:** If the router selects the vector database, the appropriate retriever (web page, PDF, or text/image) fetches relevant documents from LanceDB.
f. **Retrieval Grading:** Retrieved documents are graded by a separate LLM to assess their relevance to the query.
g. **Web Search (Optional):** If the router selects web search or if retrieval grading determines the retrieved documents are insufficient, a web search is performed using Tavily.
h. **Answer Generation:** An LLM generates an answer using the retrieved documents (and web search results, if applicable) as context.

i. **Hallucination Grading:** Another LLM checks whether the generated answer is grounded in the provided context to detect hallucinations.
j. **Answer Grading:** A final LLM assesses the usefulness of the generated answer in addressing the user's question.
k. **Response:** The final answer and the trace of the workflow execution are sent back to the Gradio client.
l. **Display:** The client displays the answer in the chat interface and the workflow trace in the "Monitor" tab.

6. Implementation

6.1 Technologies Used

a. The project is implemented in **Python 3**, leveraging the **FastAPI** framework for the backend API.
b. **Gradio** for the interactive user interface.
c. **LanceDB** serves as the vector database.
d. **LlamaIndex**, and **LangChain** facilitate interactions with LLMs and building the RAG pipeline.
e. **LangGraph** is used to define the agentic workflow.
f. **NVIDIA Embeddings**, powered by NVIDIA GPUs, are used for generating embeddings.
g. Several additional libraries handle multimedia processing, web scraping, PDF handling, OCR, and other functionalities.

6.2 Code Structure

a. **Start at __main__.py**: This script initiates server configurations, loads settings, initializes the ChatClient, and launches the Gradio interface. It also starts the FastAPI backend server.
b. **User Interface (pages/converse.py)**: The Gradio interface (chat window, input box, and configuration settings) is built here. User actions trigger event handlers to communicate with the backend API via ChatClient.

c. **Backend API (api.py)**: This FastAPI server has endpoints for interactions like query handling (/generate) and document uploads (/uploadDocument).
d. **Agentic Workflow (graph.py, compile.py)**: Using LangGraph, these files define and compile a query-handling workflow with steps like retrieval and response generation. This workflow is executed upon queries.
e. **Database Interaction (database.py)**: Manages document ingestion and retrieval, connecting with LanceDB for storing and searching documents.
f. **Utility Modules**: Files like nim.py handle NVIDIA Inference Microservices requests, and logger.py logs events for Gradio display.

6.3 Algorithms and Key Components

a. **Agentic Workflow:** The core logic is defined using LangGraph, a framework for defining workflows with conditional branching and different LLM agents. The utils/graph.py file defines the workflow graph, specifying the nodes (functions like retrieve, generate, grade_documents), edges (transitions between nodes), and conditional branching logic. The utils/compile.py module compiles this graph into an executable form.
b. **LanceDB:** This vector database efficiently stores and retrieves document embeddings. The utils/database.py module handles interactions with LanceDB.
c. **NVIDIA Embeddings:** These embeddings, optimized for GPUs, are used for generating vector representations of text and images.
d. **Multimedia Processing:** Functions within utils/database.py handle the processing of videos, audio, and images. Videos are downloaded, converted to frames, and their audio transcribed to text using SpeechRecognition. Image captioning is performed using a vision language model.

6.4 Challenges and Solutions

a. **Multimodal Data Integration:** Handling various data types (text, images, videos) presented a challenge. The implemented solution involves using separate retrievers and specialized processing for each data type. Text is extracted from multimedia sources.
b. **Agentic Workflow Management:** Implementing a complex agentic workflow requires managing state and context across different LLM calls. LangGraph provides a structure for defining and executing such workflows.
c. **External Dependencies:** The system relies on various external libraries, creating potential dependency management challenges. The code uses requirements.txt file and APT package list file to specify and install these dependencies, promoting reproducibility across environments.

7. Testing

7.1 Testing methodology

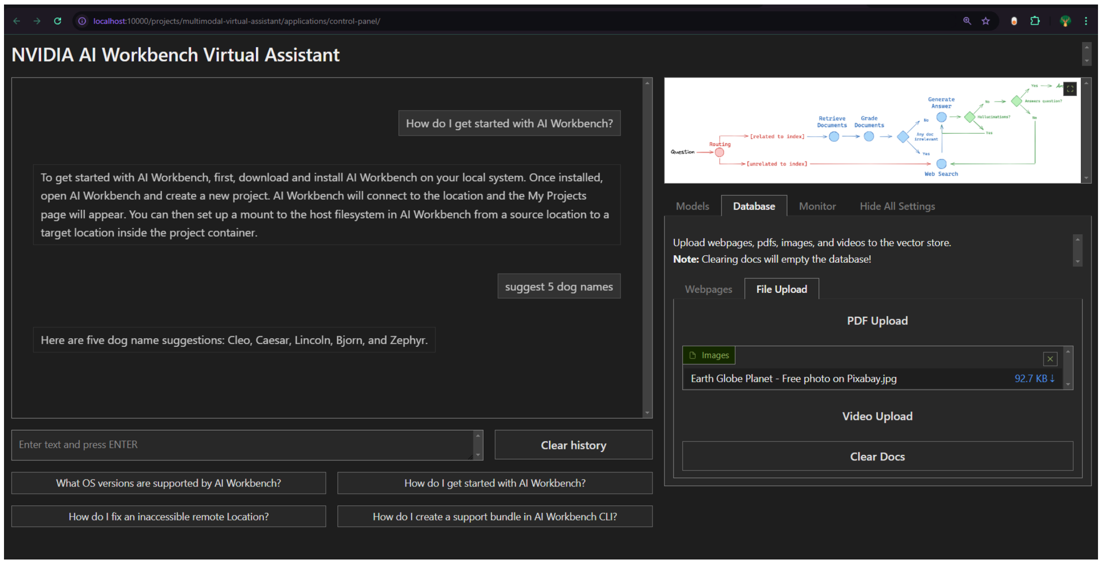
a. **Unit Testing**: Test individual components, like document ingestion and prompt formatting, to ensure each part behaves correctly.
b. **Integration Testing**: Test the RAG workflow by sending example queries, verifying the retrieval and response accuracy.
c. **End-to-End Testing**: Run the entire pipeline, from UI input to backend response, covering query types, document types, and error handling.
d. **Performance Testing**: Assess latency and speed, especially under load, to identify bottlenecks.

With NVIDIA AI Workbench:

a. Upload files, build, start the environment, and access the console through the local server.
b. Keep Docker open throughout this process.

7.2 Test Cases and Results

1. Text prompting : Passed



localhost:10000/projects/multimodal-virtual-assistant/applications/control-panel/

NVIDIA AI Workbench Virtual Assistant

How do I get started with AI Workbench?

To get started with AI Workbench, first, download and install AI Workbench on your local system. Once installed, open AI Workbench and create a new project. AI Workbench will connect to the location and the My Projects page will appear. You can then set up a mount to the host filesystem in AI Workbench from a source location to a target location inside the project container.

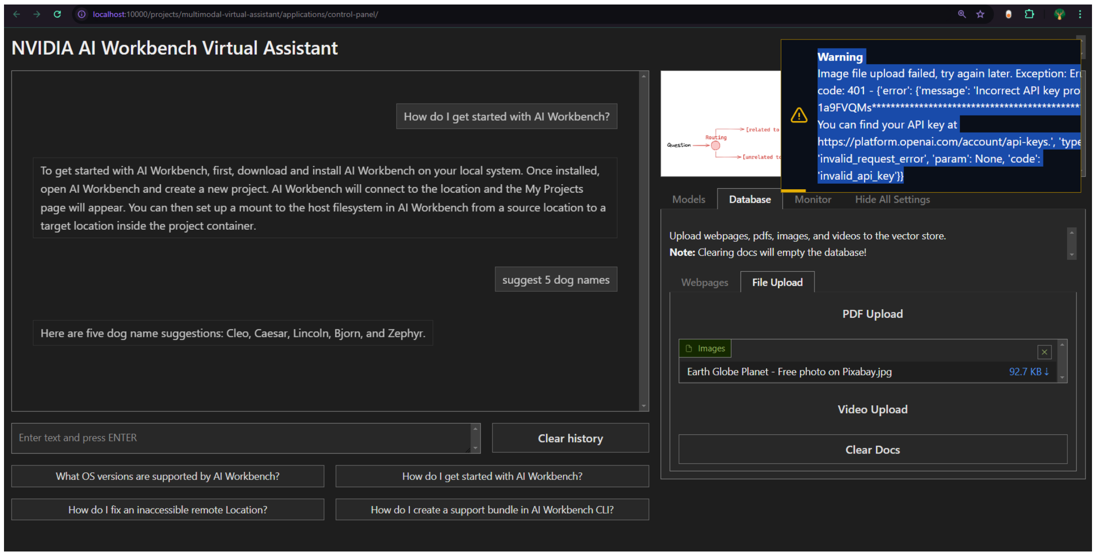
suggest 5 dog names

Here are five dog name suggestions: Cleo, Caesar, Lincoln, Bjorn, and Zephyr.

Upload webpages, pdfs, images, and videos to the vector store.
Note: Clearing docs will empty the database!

Earth Globe Planet - Free photo on Pixabay.jpg 92.7 KB ↓

2. Image upload : Failed



localhost:10000/projects/multimodal-virtual-assistant/applications/control-panel/

NVIDIA AI Workbench Virtual Assistant

How do I get started with AI Workbench?

To get started with AI Workbench, first, download and install AI Workbench on your local system. Once installed, open AI Workbench and create a new project. AI Workbench will connect to the location and the My Projects page will appear. You can then set up a mount to the host filesystem in AI Workbench from a source location to a target location inside the project container.

suggest 5 dog names

Here are five dog name suggestions: Cleo, Caesar, Lincoln, Bjorn, and Zephyr.

Warning
Image file upload failed, try again later. Exception: Er
code: 401 - {'error': {'message': 'Incorrect API key pro
1a9FVQMs**
You can find your API key at
https://platform.openai.com/account/api-keys.', 'type
'invalid_request_error', 'param': None, 'code':
'invalid_api_key'}}

Upload webpages, pdfs, images, and videos to the vector store.
Note: Clearing docs will empty the database!

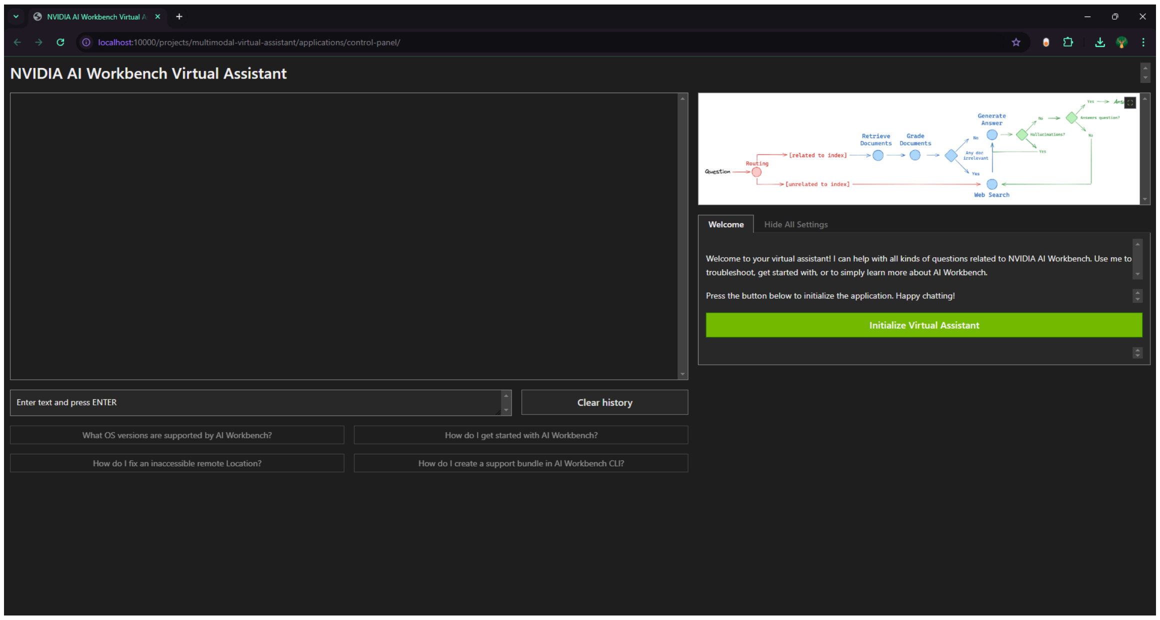
Earth Globe Planet - Free photo on Pixabay.jpg 92.7 KB ↓

Question
Routing
[related to
[unrelated to
Models
Database
Monitor
Hide All Settings
Webpages
File Upload
PDF Upload
Images
Video Upload
Clear Docs
Enter text and press ENTER
Clear history
What OS versions are supported by AI Workbench?
How do I get started with AI Workbench?
How do I fix an inaccessible remote Location?
How do I create a support bundle in AI Workbench CLI?

8. Results And Analysis

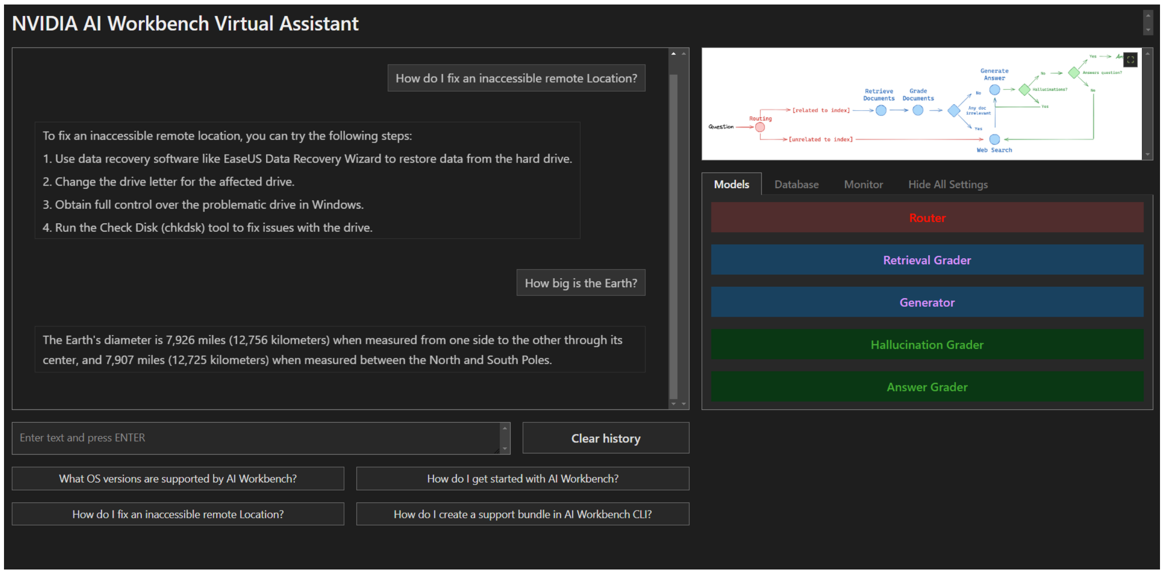
8.1 Key Outcomes

The project delivers a functional prototype of a multimodal Retrieval Augmented Generation (RAG) chatbot deployed and tested within the NVIDIA AI Workbench environment. The key outcomes demonstrate the feasibility and effectiveness of building a RAG system for specialized knowledge domains like NVIDIA AI Workbench itself. The chatbot's functionality encompasses the ingestion and processing of diverse document formats (web pages, PDFs, images, and videos) into a LanceDB vector database. Furthermore, the implementation successfully integrates an agentic workflow, employing multiple LLMs for query routing, answer generation, and quality assessment. The successful deployment and testing within the NVIDIA AI Workbench demonstrate compatibility with the platform's Dockerized environment and leverage its GPU resources for optimized performance.


NVIDIA AI Workbench Virtual A
localhost:10000/projects/multimodal-virtual-assistant/applications/control-panel/
NVIDIA AI Workbench Virtual Assistant
Question
Routing
[related to index]
[unrelated to index]
Retrieve Documents
Grade Documents
Any doc irrelevant
No
Yes
Generate Answer
Web Search
Hallucinations?
Answers question?
Welcome
Hide All Settings
Initialize Virtual Assistant
Enter text and press ENTER
Clear history
What OS versions are supported by AI Workbench?
How do I get started with AI Workbench?
How do I fix an inaccessible remote Location?
How do I create a support bundle in AI Workbench CLI?

Welcome to your virtual assistant! I can help with all kinds of questions related to NVIDIA AI Workbench. Use me to troubleshoot, get started with, or to simply learn more about AI Workbench.

Press the button below to initialize the application. Happy chatting!


NVIDIA AI Workbench Virtual Assistant
Question
Routing
[related to index]
[unrelated to index]
Retrieve Documents
Grade Documents
Generate Answer
Web Search
Models
Database
Monitor
Hide All Settings
Router
Retrieval Grader
Generator
Hallucination Grader
Answer Grader
Enter text and press ENTER
Clear history
What OS versions are supported by AI Workbench?
How do I get started with AI Workbench?
How do I fix an inaccessible remote Location?
How do I create a support bundle in AI Workbench CLI?

How do I fix an inaccessible remote Location?

To fix an inaccessible remote location, you can try the following steps:
1. Use data recovery software like EaseUS Data Recovery Wizard to restore data from the hard drive.
2. Change the drive letter for the affected drive.
3. Obtain full control over the problematic drive in Windows.
4. Run the Check Disk (chkdsk) tool to fix issues with the drive.

How big is the Earth?

The Earth's diameter is 7,926 miles (12,756 kilometers) when measured from one side to the other through its center, and 7,907 miles (12,725 kilometers) when measured between the North and South Poles.

8.2 Performance Metrics

To analyze the system's performance, metrics like query response time, document retrieval speed, and LLM inference latency should be collected and analyzed. Precision and recall metrics could be used to evaluate retrieval quality. Human evaluation is also important to assess the quality and helpfulness of generated answers. Evaluation should be done with benchmark datasets and real-world usage scenarios.

Metric	**Tools/Techniques**
Response Time (Latency)	Postman, nvidia-smi, NVIDIA Nsight Systems
Throughput	Apache JMeter, Locust
Accuracy/Success Rate	Custom logging, BLEU, ROUGE
Resource Usage	nvidia-smi, NVIDIA DCGM, Prometheus
Inference Latency	NVIDIA NIM metrics, TensorRT optimizations

9. Discussion

9.1 Limitations

The current assistant is limited to predefined inference endpoints and the NVIDIA AI Workbench product. Additionally, latency may occur when handling large volumes of multimodal data, and certain query topics might still require human oversight.

9.2 Future Enhancements

Future iterations could include more dynamic content update mechanisms, expanded product coverage, and improved integration with third-party APIs to extend functionality. Developing a feedback loop could also enable continuous learning and adaptation to emerging support issues.

10. Conclusion

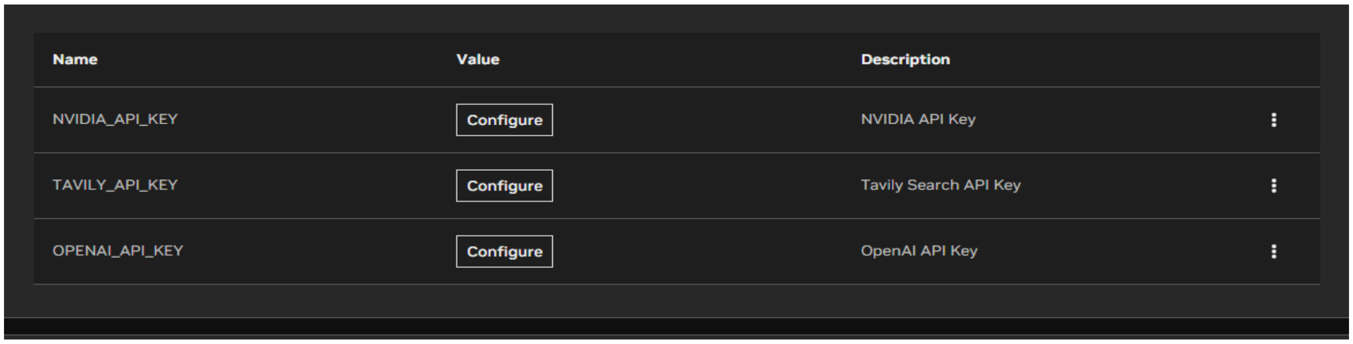
The NVIDIA AI Workbench virtual assistant project successfully demonstrates how multimodal RAG and websearch integration can enhance user experience by providing quick, contextually accurate responses to technical queries. This project sets a foundation for product-specific, intelligent virtual assistants in AI software, with potential for further expansion into other domains and more advanced knowledge management features.

11. References

1. *What is NVIDIA AI Workbench?* (n.d.). NVIDIA Docs. Retrieved November 9, 2024, from https://docs.nvidia.com/ai-workbench/user-guide/latest/overview/introduction.html#what-is-ai-workbench
2. (n.d.). Try NVIDIA NIM APIs. Retrieved November 9, 2024, from https://build.nvidia.com/explore/discover

12. Appendices

1. TAVILY_SEARCH_API : https://app.tavily.com/home
2. NVIDIA_API_KEY : https://build.nvidia.com/mistralai/mistral-7b-instruct-v2?integrate_nim=true&hosted_api=true
3. OPENAI_API_KEY : https://drive.google.com/file/d/1-MI43jc52ve3Ku9BYjmOiSBZj2a1hAu4/view



Name	Value	Description
NVIDIA_API_KEY	Configure	NVIDIA API Key
TAVILY_API_KEY	Configure	Tavily Search API Key
OPENAI_API_KEY	Configure	OpenAI API Key