Comprehensive Development Documentation for Blitz Chatbot

Introduction

The development of Blitz, the AI chatbot for the Bibliotheca Alexandrina, involved extensive research, testing, and implementation of various tools and technologies. This document provides a detailed account of the tools used, the challenges encountered, and the decisions made during the project. The goal was to create a robust, multilingual chatbot capable of handling customer support queries with high accuracy and efficiency.

Tools and Technologies

H2oGPT

Overview: H2oGPT was the initial platform used for developing Blitz. It offered a range of features, including integration with the Gradio and OpenAI servers.
Challenges:

- **Lack of Retrieval-Augmented Generation (RAG)**: H2oGPT did not support RAG, which was essential for improving the chatbot's contextual accuracy and information retrieval capabilities.
- **Prompt Injection Vulnerabilities**: The platform lacked built-in protection against prompt injection attacks, posing a security risk.
- **Server Issues**: The Gradio server had performance issues, and while the OpenAI server provided some functionality, it did not allow full access to the desired features.

Decision: Due to these limitations, H2oGPT was eventually discontinued in favor of more advanced platforms.

LM Studio

Overview: LM Studio was chosen for its superior performance and more refined API endpoints, which facilitated easier experimentation with different models.
Key Features:

- **Model Flexibility**: The ability to test and switch between models like LLaMA 3.1 and Mistral 7B provided greater flexibility.
- **RAG Limitation**: Despite its strengths, LM Studio lacked support for RAG, which was crucial for the project.

Decision: While LM Studio offered improved performance, the lack of RAG functionality led to further exploration of other tools.

Custom RAG APIs

Overview: Several versions of custom RAG APIs were developed using Ollama, Langchain, LlamaIndex, and vector databases like ChromaDB.
Challenges:

- **Performance**: The custom APIs delivered average performance and lacked advanced features necessary for the project.
- **Complexity**: Developing and maintaining these custom solutions proved to be time-consuming and complex.

Decision: The custom RAG APIs were eventually replaced by the Ollama-Dify integration, which provided a more robust and feature-rich solution.

AnythingLLM and OpenWebUI

Overview: Both AnythingLLM and OpenWebUI were explored for their RAG capabilities and rich feature sets.
Challenges:

- **Lack of Developer Support**: Both tools suffered from poor developer support, with APIs that either did not work or were not available.
- **Feature Richness**: Despite their potential, the tools could not be utilized to their full capabilities due to these limitations.

Decision: Neither AnythingLLM nor OpenWebUI was adopted due to their lack of functional APIs and support.

Ollama and Dify

Overview: Ollama and Dify were integrated to leverage the strengths of both platforms. Ollama was used for running the LLMs locally, while Dify was employed for customizing the models and adding necessary features to enhance the chatbot’s capabilities.
Key Features:

- **RAG Capabilities**: Ollama enabled the testing and implementation of RAG, which was pivotal in improving the chatbot's ability to retrieve and generate accurate responses based on context.
- **Model Customization**: Dify allowed for extensive customization of the models, refining the system prompt and integrating additional features that enhanced performance and security.
- **Prompt Injection Protection**: Dify integrated seamlessly with prompt injection APIs, significantly enhancing the security of the chatbot.

Decision: The integration of Ollama and Dify proved to be the most effective solution for developing Blitz, providing the necessary performance, customization, and security features required for the project.

Tool	Overview	Key Features	Challenges	Decision
H2oGPT	Initial platform used for Blitz development, offering integration with Gradio and OpenAI servers.	- Integration with Gradio and OpenAI servers	- Lack of RAG functionality - Vulnerable to prompt injection attacks - Performance issues with Gradio server - Limited access to OpenAI server features	Discontinued due to limitations in RAG and security.
LM Studio	Chosen for its superior performance and refined API endpoints, facilitating easier model experimentation.	- Model flexibility (LLaMA 3.1, Mistral 7B) - Superior performance	- Lack of RAG functionality	Further exploration of other tools due to the absence of RAG support.
Custom RAG APIs	Developed using Ollama, Langchain, LlamaIndex, and ChromaDB to experiment with RAG implementation.	- Custom RAG implementations using various technologies	- Average performance- Lacked advanced features - Complexity in development and maintenance	-Replaced by the Ollama -Dify integration for better performance and feature richness.
AnythingLLM & OpenWebUI	Explored for their RAG capabilities and rich feature sets.	- Rich feature sets in RAG	- Poor developer support - Non-functional or unavailable APIs	Not adopted due to lack of functional APIs and support.
Ollama & Dify	Integrated to combine Ollama's local LLM capabilities with Dify's model customization and feature enhancement.	- RAG capabilities- Extensive model customization - Prompt injection protection - Superior multilingual support	- N/A	Proven to be the most effective solution, providing necessary performance, customization, and security.

Language Models & Embedding Models Used

Initially, I started with the Mistral 7B model, but later transitioned to LLaMA 3.1 due to its superior overall performance. However, for better handling of Arabic language queries, I eventually adopted the Aya model, which has proven to be more effective in this domain.

Regarding embedding models, I experimented with both Nomic Embed and Mxbai-Embed-Large. Although I initially used Mxbai, I eventually chose Nomic Embed due to its lighter footprint. However, I haven't fully explored the potential of Mxbai yet.

As of now, I am primarily using the Aya model for language processing and Nomic Embed for embedding tasks, balancing efficiency and performance.

Ollama

Installing Ollama

macOS

To set up Ollama on macOS, follow the steps below:

1. **Download** the latest version from the official Ollama website.
2. Install the application by following the on-screen instructions.

Windows (Preview)

For Windows users, the Ollama preview version is available:

1. **Download** the preview version from the official website.
2. Follow the installation guide provided during the setup process.

Linux

Linux users can install Ollama via the terminal:

1. **Download and install** Ollama using the following command:

   ```
   curl -fsSL https://ollama.com/install.sh | sh
   ```

2. Alternatively, refer to the **manual install instructions** on the Ollama website for detailed steps.

Docker

For those preferring to use Docker, Ollama offers an **official Docker image**:

1. Pull the Docker image from Docker Hub by running:

   ```
   docker pull ollama/ollama
   ```

2. Follow the Docker-specific instructions on Docker Hub to complete the setup.

Using Ollama

Once Ollama is installed, you can begin pulling the models you need. Use the following command to download a specific model:

```
ollama pull model-name
```

REST API

Ollama provides a REST API for running and managing models, allowing you to generate responses or interact with models via HTTP requests.

Generate a Response

To generate a response from a model, use the following curl command:

```
curl http://localhost:11434/api/generate -d '{
  "model": "llama3.1",
  "prompt":"Why is the sky blue?"
}'
```

This command sends a request to the API, specifying the model (llama3.1) and the prompt ("Why is the sky blue?"). The model will return a generated response based on the input.

Chat with a Model

To engage in a chat with a model, use the following curl command:

```
curl http://localhost:11434/api/chat -d '{
  "model": "llama3.1",
  "messages": [
    { "role": "user", "content": "why is the sky blue?" }
  ]
}'
```

This command sends a chat message to the specified model. The "messages" array allows for multiple interactions, with the "role" parameter distinguishing between user and model inputs.

See the API documentation for all endpoints.

Models:

1. Aya
2. Llama 3.1
3. Nomic-embed-text
4. Mxbai-embed-large

Dify

Dify is an open-source platform that streamlines the development of AI applications by integrating Backend-as-a-Service with LLMOps. It is designed to make generative AI accessible to a wide range of users, from developers to non-technical innovators.

Core Features

- **Mainstream LLM Support:** Dify integrates seamlessly with leading language models, ensuring compatibility and ease of use.
- **Prompt Orchestration Interface:** An intuitive interface allows for the effective creation, management, and deployment of AI prompts.
- **High-Quality RAG Engines:** Dify includes robust Retrieval-Augmented Generation (RAG) engines, enhancing the quality of AI-generated content.
- **Flexible AI Agent Framework:** The platform supports the development and deployment of customizable AI agents.
- **Low-Code Workflow:** Dify offers a user-friendly, low-code environment, enabling rapid development and deployment of AI solutions.
- **Easy-to-Use Interfaces and APIs:** Simplifies interactions with the platform, reducing the complexity of AI development.

The Dify Advantage

Unlike many AI development tools that provide only individual components, Dify offers a comprehensive, production-ready solution. It serves as a well-designed scaffolding system rather than just a toolbox, ensuring that all aspects of AI application development are seamlessly integrated.

As an open-source platform, Dify benefits from the contributions of a dedicated professional team and an active community. This collaborative approach drives rapid iteration, robust features, and a user-friendly interface.

With Dify, we can:

- Deploy AI capabilities similar to Assistants API and GPTs using any model.
- Maintain full control over your data with flexible security options.
- Leverage an intuitive interface for easy management and deployment of AI applications.

Running Dify Locally

Prerequisites

Operating System	**Software**	**Explanation**
macOS 10.14 or later	Docker Desktop	Set the Docker virtual machine (VM) to use a minimum of 2 virtual CPUs (vCPUs) and 8 GB of initial memory. Otherwise, the installation may fail. For more information, please refer to the Docker Desktop installation guide for Mac.
Linux platforms	Docker 19.03 or later Docker Compose 1.25.1 or later	Please refer to the Docker installation guide and the Docker Compose installation guide for more information on how to install Docker and Docker Compose, respectively.
Windows with WSL 2 enabled	Docker Desktop	We recommend storing the source code and other data that is bound to Linux containers in the Linux file system rather than the Windows file system. For more information, please refer to the Docker Desktop installation guide for using the WSL 2 backend on Windows.

Clone Dify

Clone the Dify source code to your local machine:

```
git clone https://github.com/langgenius/dify.git
```

Start Dify

Navigate to the docker directory in the Dify source code and execute the following command to start Dify:

```
cd dify/docker
cp .env.example .env
docker compose up -d
```

Access Dify

Finally, access `http://localhost/install` to use the deployed Dify.

Adding Model Provider(Ollama)

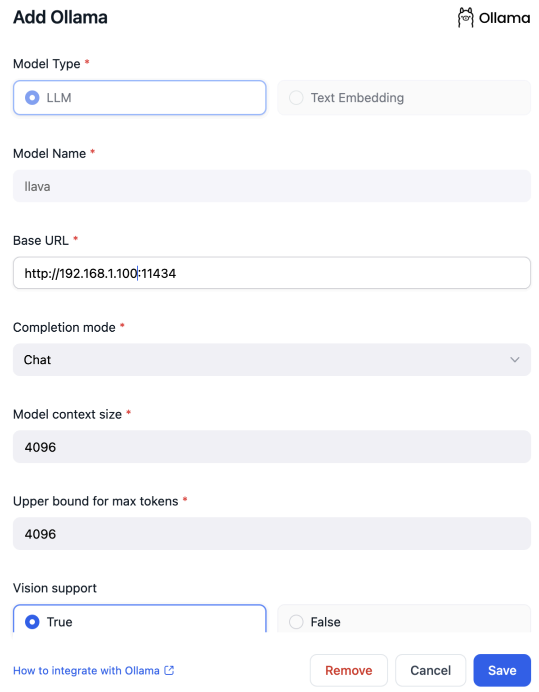
1. Pull model using Ollama

   ```
   ollama pull llava
   ```

   After successful launch, Ollama starts an API service on local port 11434, which can be accessed at http://localhost:11434.

2. Integrate Ollama in Dify

   In `Settings > Model Providers > Ollama`, fill in:



Add Ollama

Model Type *

● LLM ○ Text Embedding

Model Name *

llava

Base URL *

http://192.168.1.100:11434

Completion mode *

Chat

Model context size *

4096

Upper bound for max tokens *

4096

Vision support

● True ○ False

- Model Name: `llava`
- Base URL: `http://<your-ollama-endpoint-domain>:11434`
- Enter the base URL where the Ollama service is accessible.
- If Dify is deployed using docker, consider using the local network IP address, e.g., `http://192.168.1.100:11434` or the docker host machine IP address, e.g., `http://172.17.0.1:11434`.
- For local source code deployment, use `http://localhost:11434`.
- Model Type: `Chat`
- Model Context Length: `4096`
- The maximum context length of the model. If unsure, use the default value of 4096.
- Maximum Token Limit: `4096`
- The maximum number of tokens returned by the model. If there are no specific requirements for the model, this can be consistent with the model context length.
- Support for Vision: `Yes`
- Check this option if the model supports image understanding (multimodal), like `llava`.

Click "Save" to use the model in the application after verifying that there are no errors.
The integration method for Embedding models is similar to LLM, just change the model type to Text Embedding.

If you are using docker to deploy Dify and Ollama, you may encounter the following error:

```
httpconnectionpool(host=127.0.0.1, port=11434): max retries exceeded with
url:/cpi/chat (Caused by
NewConnectionError('<urllib3.connection.HTTPConnection object at
0x7f8562812c20>: fail to establish a new connection:[Errno 111] Connection
refused'))
```

```
httpconnectionpool(host=localhost, port=11434): max retries exceeded with
url:/cpi/chat (Caused by
NewConnectionError('<urllib3.connection.HTTPConnection object at
0x7f8562812c20>: fail to establish a new connection:[Errno 111] Connection
refused'))
```

This error occurs because the Ollama service is not accessible from the docker container.

`Localhost` usually refers to the container itself, not the host machine or other containers.

To resolve this issue, you need to expose the Ollama service to the network.

Setting environment variables on Mac

If Ollama is run as a macOS application, environment variables should be set using `launchctl`:

1. For each environment variable, call `launchctl setenv`.

   ```
   launchctl setenv OLLAMA_HOST "0.0.0.0"
   ```

2. Restart Ollama application.
3. If the above steps are ineffective, you can use the following method:
4. The issue lies within Docker itself, and to access the Docker host. you should connect to `host.docker.internal`. Therefore, replacing `localhost` with `host.docker.internal` in the service will make it work effectively.

   ```
   http://host.docker.internal:11434
   ```

Setting environment variables on Linux

If Ollama is run as a systemd service, environment variables should be set using `systemctl`:

1. Edit the systemd service by calling `systemctl edit ollama.service`. This will open an editor.
2. For each environment variable, add a line `Environment` under section `[Service]`:

   ```
   Environment="OLLAMA_HOST=0.0.0.0"
   ```

3. Save and exit.
4. Reload `systemd` and restart Ollama:

   ```
   systemctl daemon-reload
   systemctl restart ollama
   ```

Setting environment variables on Windows

On windows, Ollama inherits your user and system environment variables.

1. First Quit Ollama by clicking on it in the task bar
2. Edit system environment variables from the control panel
3. Edit or create New variable(s) for your user account for `OLLAMA_HOST`, `OLLAMA_MODELS`, etc.
4. Click OK/Apply to save
5. Run `ollama` from a new terminal window

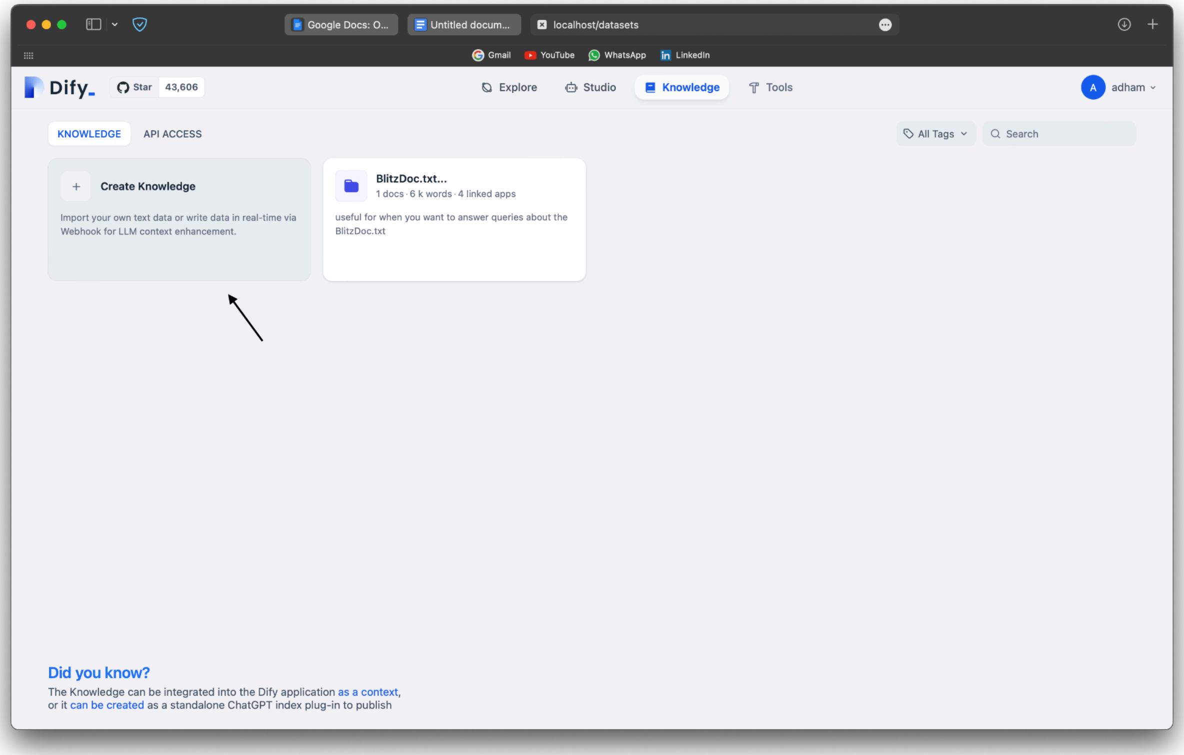
How can I expose Ollama on my network?

Ollama binds 127.0.0.1 port 11434 by default. Change the bind address with the `OLLAMA_HOST` environment variable.

For more information visit [Dify Documentation]

Adding Knowledge Base

1. Go to `localhost/databases` and select `Create Knowledge`



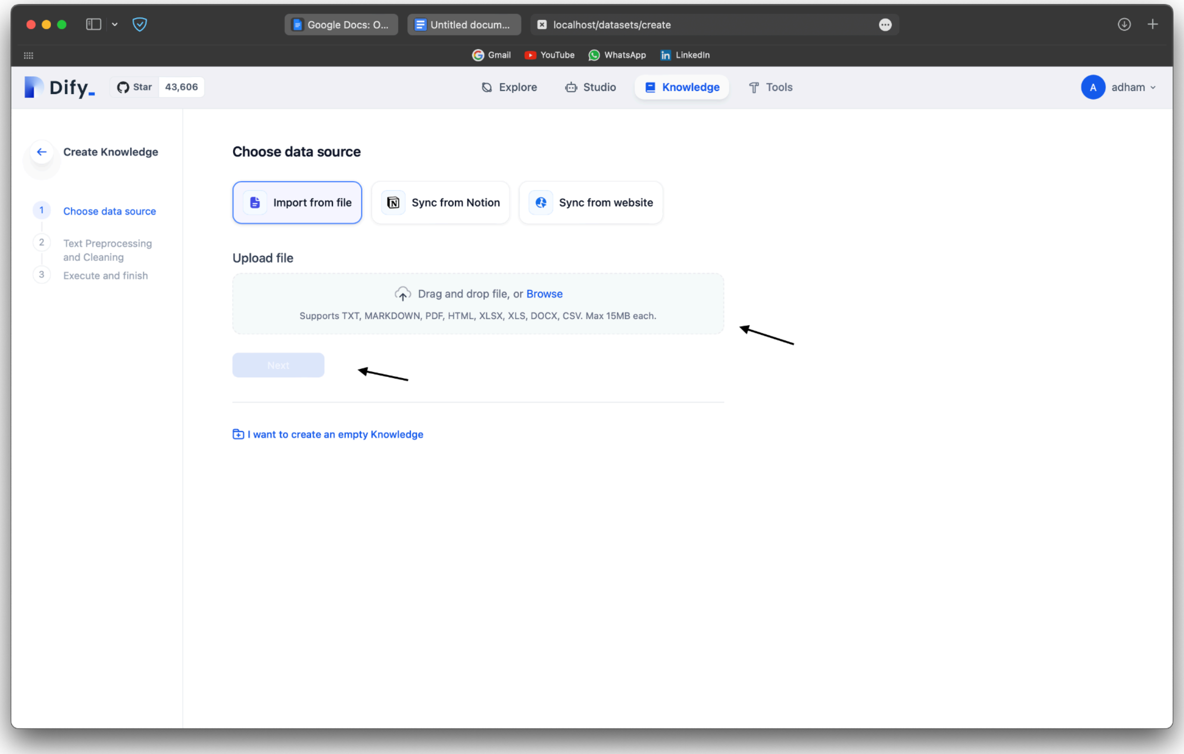
Import your own text data or write data in real-time via Webhook for LLM context enhancement.

useful for when you want to answer queries about the BlitzDoc.txt

Did you know?

The Knowledge can be integrated into the Dify application as a context, or it can be created as a standalone ChatGPT index plug-in to publish

2. Upload File and Click Next



Create Knowledge

1 Choose data source
2 Text Preprocessing and Cleaning
3 Execute and finish

Choose data source

Import from file | Sync from Notion | Sync from website

Upload file

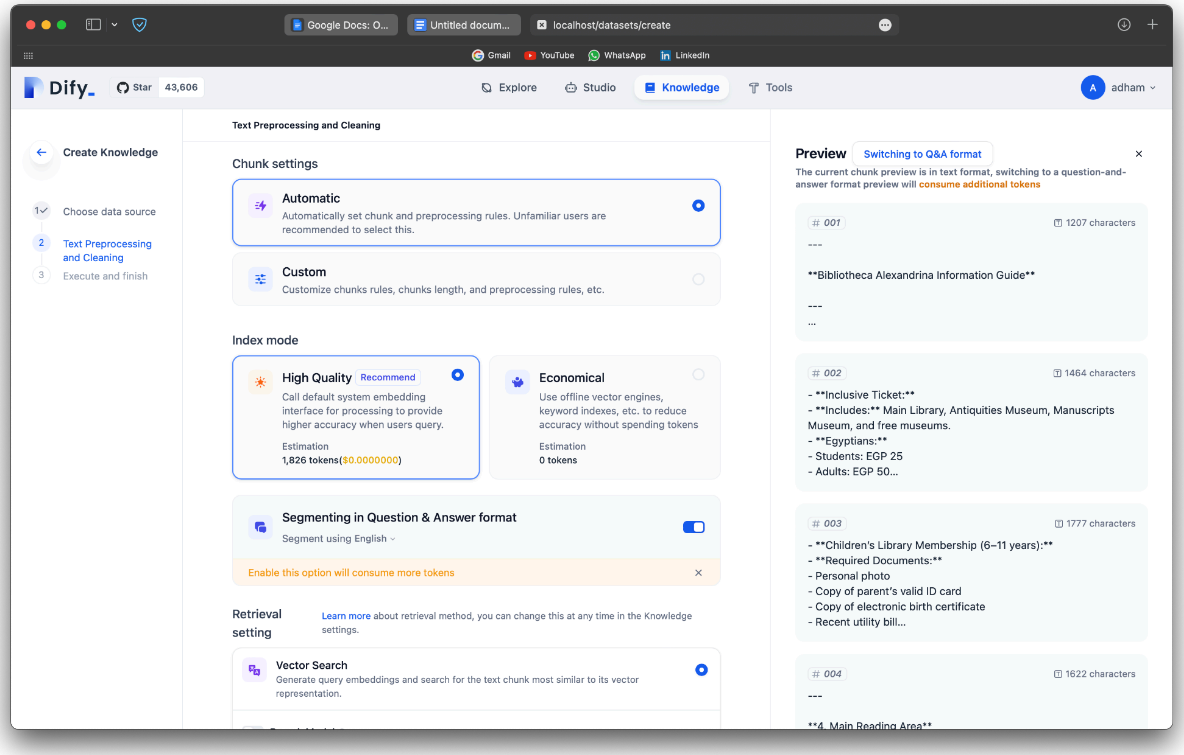
Drag and drop file, or Browse

Supports TXT, MARKDOWN, PDF, HTML, XLSX, XLS, DOCX, CSV. Max 15MB each.

Next

I want to create an empty Knowledge

3. Set the Chunking settings to automatic and Choose high quality indexing and choose the embedding model running local on ollama enable segmenting in Q&A format



Create Knowledge

1 Choose data source
2 Text Preprocessing and Cleaning
3 Execute and finish

Text Preprocessing and Cleaning

Chunk settings

● Automatic — Automatically set chunk and preprocessing rules. Unfamiliar users are recommended to select this.

○ Custom — Customize chunks rules, chunks length, and preprocessing rules, etc.

Index mode

● High Quality (Recommend) — Call default system embedding interface for processing to provide higher accuracy when users query.
Estimation
1,826 tokens($0.0000000)

○ Economical — Use offline vector engines, keyword indexes, etc. to reduce accuracy without spending tokens
Estimation
0 tokens

Segmenting in Question & Answer format
Segment using English

Enable this option will consume more tokens ×

Retrieval setting

Learn more about retrieval method, you can change this at any time in the Knowledge settings.

● Vector Search — Generate query embeddings and search for the text chunk most similar to its vector representation.

Preview | Switching to Q&A format ×

The current chunk preview is in text format, switching to a question-and-answer format preview will consume additional tokens

001 — 1207 characters

Bibliotheca Alexandrina Information Guide

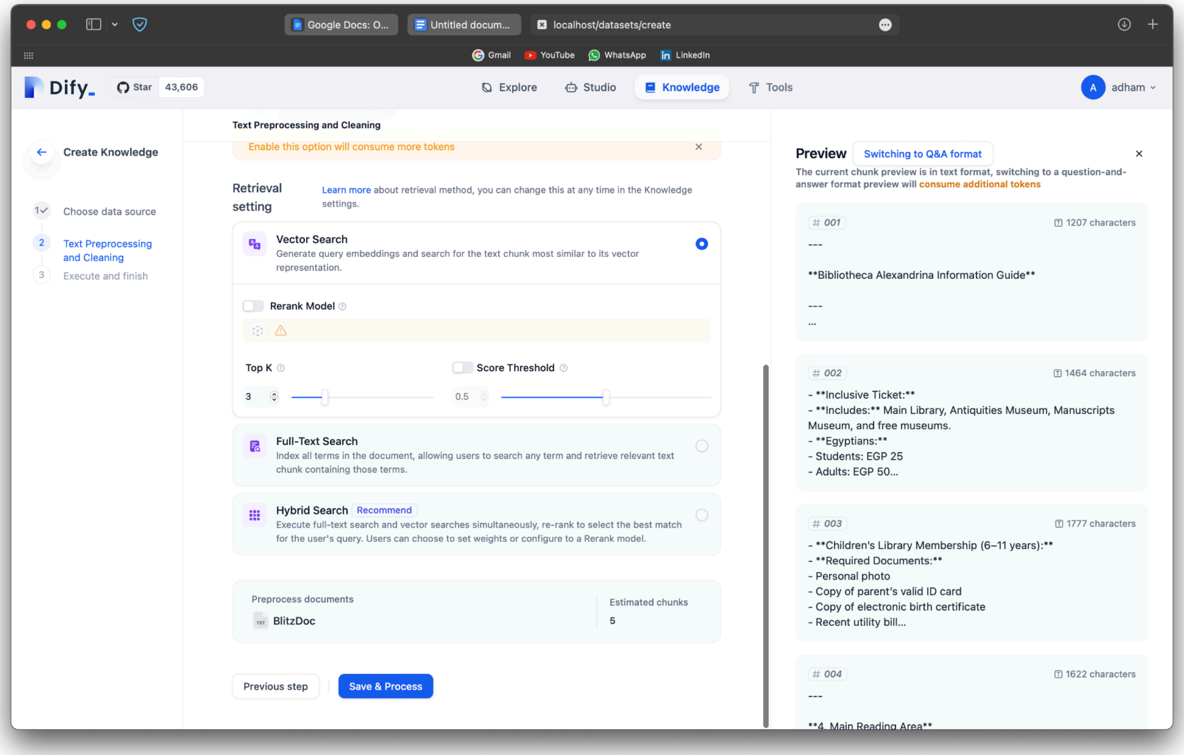
...

002 — 1464 characters
- **Inclusive Ticket:**
- **Includes:** Main Library, Antiquities Museum, Manuscripts Museum, and free museums.
- **Egyptians:**
- Students: EGP 25
- Adults: EGP 50...

003 — 1777 characters
- **Children's Library Membership (6–11 years):**
- **Required Documents:**
- Personal photo
- Copy of parent's valid ID card
- Copy of electronic birth certificate
- Recent utility bill...

004 — 1622 characters
---
4. Main Reading Area

4. I initially used **Vector Search** for retrieval in Blitz due to its efficiency. However, I'm exploring **Hybrid Search** for potentially better accuracy. I also recommend integrating a **reranking model** to enhance the relevance of the final responses.



Text Preprocessing and Cleaning

Enable this option will consume more tokens

Retrieval setting — Learn more about retrieval method, you can change this at any time in the Knowledge settings.

● Vector Search
Generate query embeddings and search for the text chunk most similar to its vector representation.

Rerank Model

Top K: 3

Score Threshold: 0.5

○ Full-Text Search
Index all terms in the document, allowing users to search any term and retrieve relevant text chunk containing those terms.

○ Hybrid Search — Recommend
Execute full-text search and vector searches simultaneously, re-rank to select the best match for the user's query. Users can choose to set weights or configure to a Rerank model.

Preprocess documents: BlitzDoc

Estimated chunks: 5

Preview — Switching to Q&A format

The current chunk preview is in text format, switching to a question-and-answer format preview will consume additional tokens

001 — 1207 characters

```
---

**Bibliotheca Alexandrina Information Guide**

---
...
```

002 — 1464 characters

```
- **Inclusive Ticket:**
- **Includes:** Main Library, Antiquities Museum, Manuscripts Museum, and free museums.
- **Egyptians:**
- Students: EGP 25
- Adults: EGP 50...
```

003 — 1777 characters

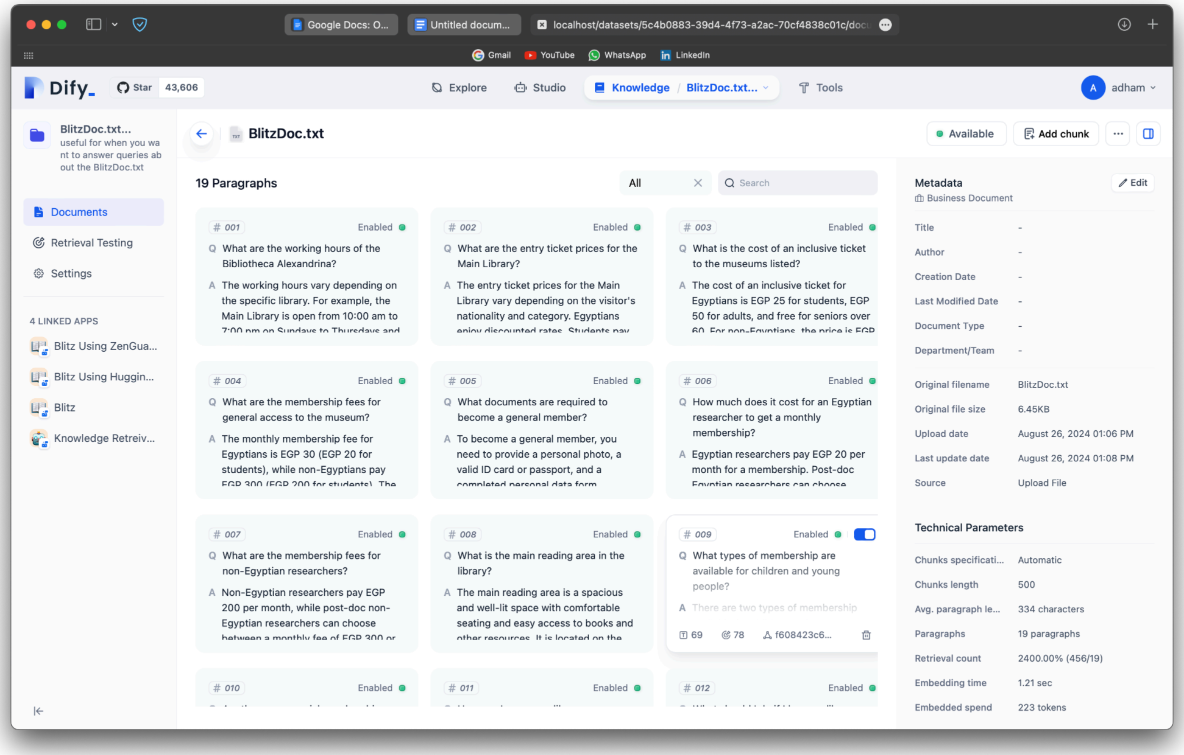
```
- **Children's Library Membership (6–11 years):**
- **Required Documents:**
- Personal photo
- Copy of parent's valid ID card
- Copy of electronic birth certificate
- Recent utility bill...
```

004 — 1622 characters

```
---

**4. Main Reading Area**
```

5. The Questions and answers are displayed after chunking and segmentation



BlitzDoc.txt... — useful for when you want to answer queries about the BlitzDoc.txt

BlitzDoc.txt

19 Paragraphs

001 — Enabled
Q What are the working hours of the Bibliotheca Alexandrina?
A The working hours vary depending on the specific library. For example, the Main Library is open from 10:00 am to 7:00 pm on Sundays to Thursdays and

002 — Enabled
Q What are the entry ticket prices for the Main Library?
A The entry ticket prices for the Main Library vary depending on the visitor's nationality and category. Egyptians enjoy discounted rates. Students pay

003 — Enabled
Q What is the cost of an inclusive ticket to the museums listed?
A The cost of an inclusive ticket for Egyptians is EGP 25 for students, EGP 50 for adults, and free for seniors over 60. For non-Egyptians, the price is EGP

004 — Enabled
Q What are the membership fees for general access to the museum?
A The monthly membership fee for Egyptians is EGP 30 (EGP 20 for students), while non-Egyptians pay EGP 300 (EGP 200 for students). The

005 — Enabled
Q What documents are required to become a general member?
A To become a general member, you need to provide a personal photo, a valid ID card or passport, and a completed personal data form

006 — Enabled
Q How much does it cost for an Egyptian researcher to get a monthly membership?
A Egyptian researchers pay EGP 20 per month for a membership. Post-doc Egyptian researchers can choose

007 — Enabled
Q What are the membership fees for non-Egyptian researchers?
A Non-Egyptian researchers pay EGP 200 per month, while post-doc non-Egyptian researchers can choose between a monthly fee of EGP 300 or

008 — Enabled
Q What is the main reading area in the library?
A The main reading area is a spacious and well-lit space with comfortable seating and easy access to books and other resources. It is located on the

009 — Enabled
Q What types of membership are available for children and young people?
A There are two types of membership
69 · 78 · f608423c6...

010 — Enabled

011 — Enabled

012 — Enabled

Metadata
Business Document

Title	-
Author	-
Creation Date	-
Last Modified Date	-
Document Type	-
Department/Team	-
Original filename	BlitzDoc.txt
Original file size	6.45KB
Upload date	August 26, 2024 01:06 PM
Last update date	August 26, 2024 01:08 PM
Source	Upload File

Technical Parameters

Chunks specificati...	Automatic
Chunks length	500
Avg. paragraph le...	334 characters
Paragraphs	19 paragraphs
Retrieval count	2400.00% (456/19)
Embedding time	1.21 sec
Embedded spend	223 tokens

File Used: BlitzDoc.txt

Monitoring and Logging Features

- **Total Messages**:
  - **Definition**: Represents the total number of interactions the AI handles each day. Each user query answered counts as a single message. Note that prompt orchestration and debugging sessions are excluded from this count.

- **Active Users**:
  - **Definition**: Indicates the number of unique users who have engaged with the AI through more than one question-and-answer exchange. This metric excludes prompt orchestration and debugging sessions.

- **Average Session Interactions**:
  - **Definition**: Measures the number of continuous interactions within a session. For instance, if a user engages in a 10-round Q&A with the AI, it counts as 10 interactions. This metric provides insight into user engagement and is available only for conversational applications.

- **Token Output Speed**:
  - **Definition**: Tracks the number of tokens generated per second. This metric indirectly reflects the model's generation rate and the application's usage frequency.

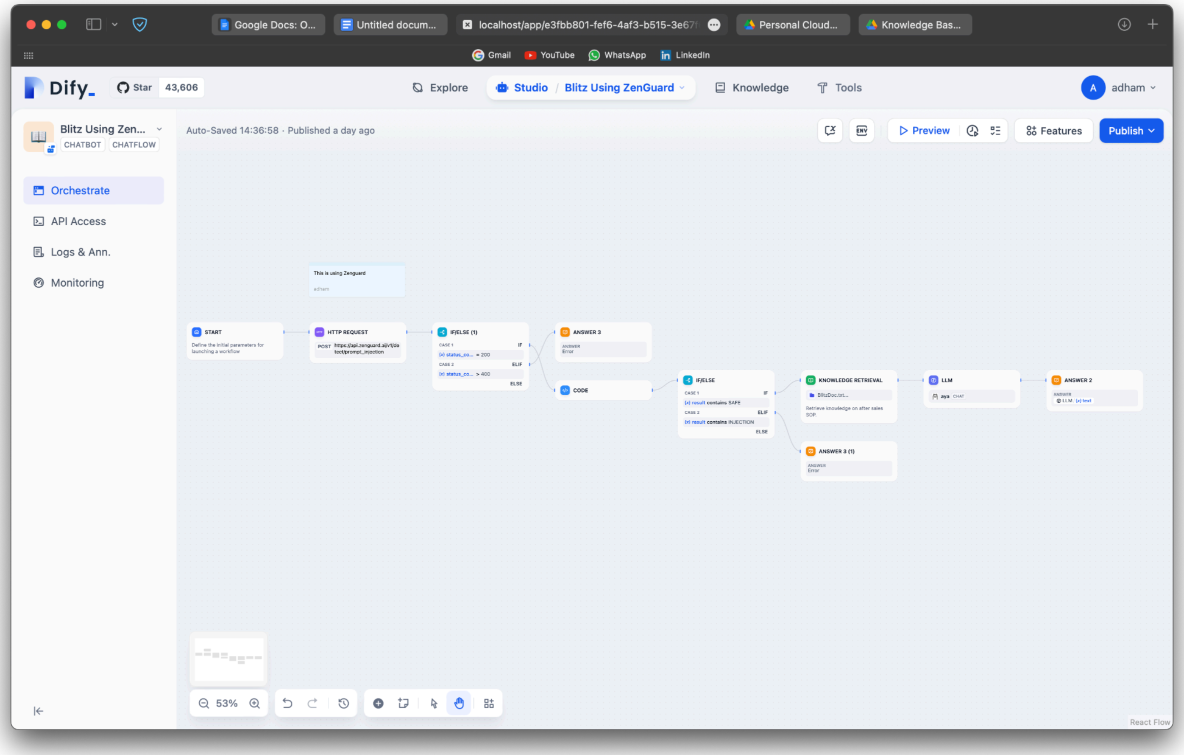
- **User Satisfaction Rate**:
  - **Definition**: Calculated as the number of likes received per 1,000 messages. This metric gauges the proportion of users who are highly satisfied with the AI’s responses.

Blitz

Blitz operates through a multi-step process designed to ensure safe and accurate interactions:

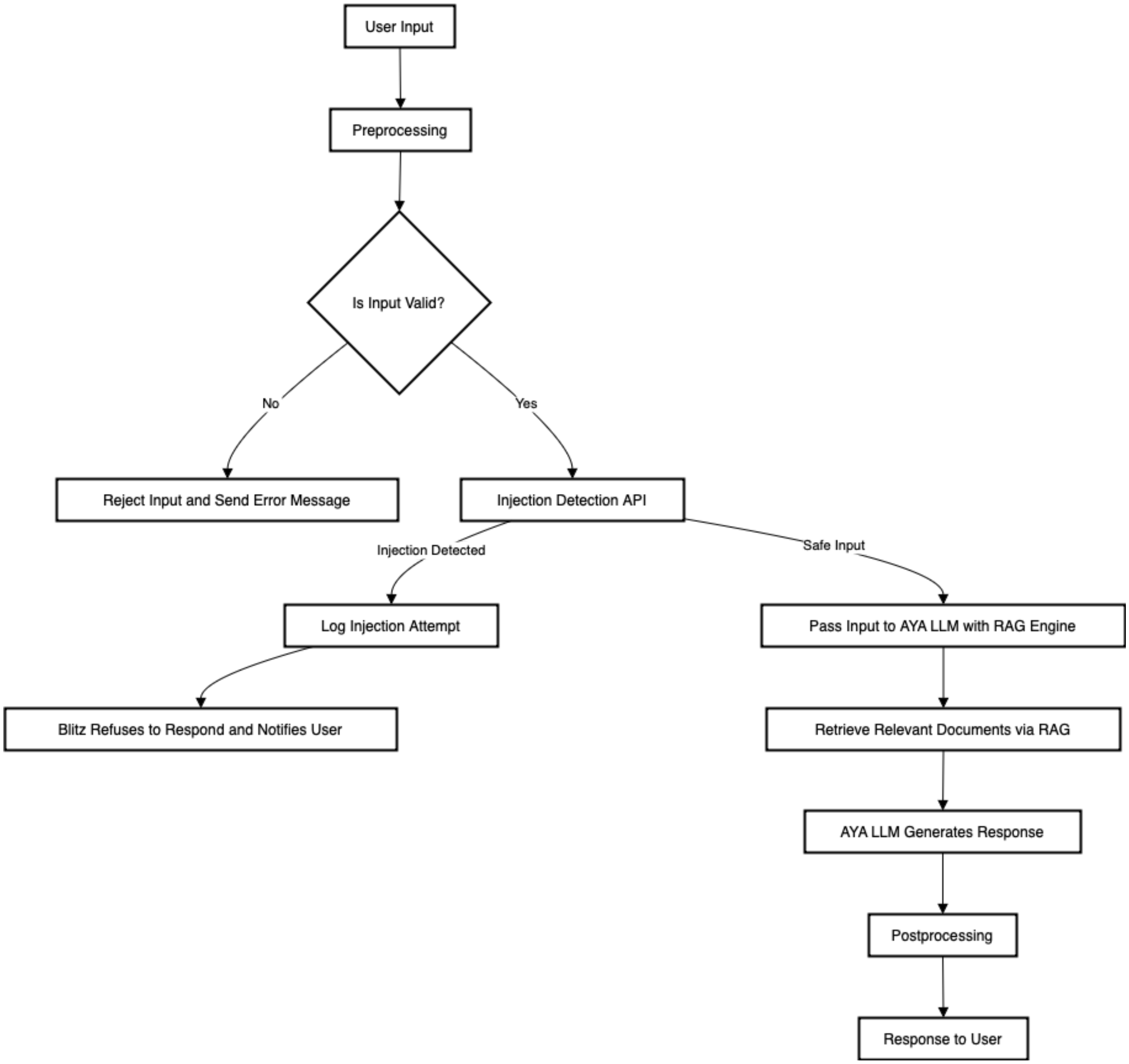
1. **User Input Collection**: Blitz begins by capturing input from the user.
2. **Input Evaluation**: The captured input is then sent to an API that performs rigorous checks for any malicious content or prompt injection attempts. This step is crucial for maintaining the integrity and security of the system.
3. **Classification**: Based on the API's assessment, the input is classified as either 'safe' or 'injection.'
4. **Response Generation**: If the input is deemed safe, Blitz proceeds to generate a response using the provided knowledge base. This ensures that the output is relevant and accurate according to the information available.

This process effectively safeguards against potential security threats while delivering reliable and contextually appropriate responses.

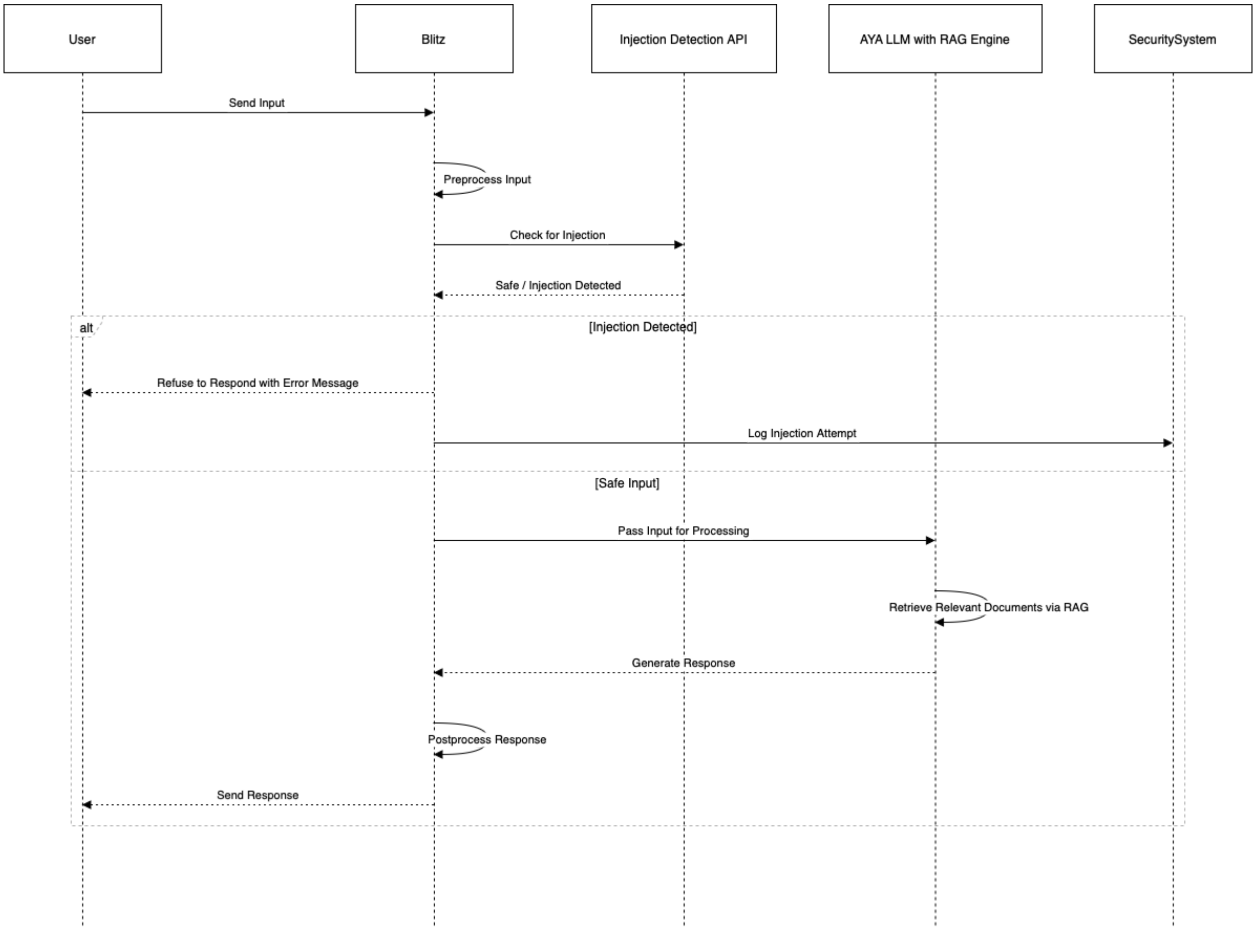


Diagrams

Flowchart


User Input
Preprocessing
Is Input Valid?
No
Yes
Reject Input and Send Error Message
Injection Detection API
Injection Detected
Safe Input
Log Injection Attempt
Pass Input to AYA LLM with RAG Engine
Blitz Refuses to Respond and Notifies User
Retrieve Relevant Documents via RAG
AYA LLM Generates Response
Postprocessing
Response to User

Sequence


User
Blitz
Injection Detection API
AYA LLM with RAG Engine
SecuritySystem
Send Input
Preprocess Input
Check for Injection
Safe / Injection Detected
alt
[Injection Detected]
Refuse to Respond with Error Message
Log Injection Attempt
[Safe Input]
Pass Input for Processing
Retrieve Relevant Documents via RAG
Generate Response
Postprocess Response
Send Response

LLM Protection

Blitz utilizes two distinct methods for detecting prompt injection:

1. **Local Detection with DeBERTa Model:** The system employs the HuggingFace protectai/deberta-v3-base-prompt-injection, a DeBERTa v3 model specifically fine-tuned to identify prompt injection attempts. This model runs locally, offering a robust solution for prompt injection detection within the system.

2. **ZenGuard API:** Alternatively, Blitz can leverage ZenGuard, an API designed to detect prompt injections. Rigorous testing has shown that ZenGuard offers slightly higher accuracy compared to the local DeBERTa model.

These methods work in tandem to ensure the integrity and security of user interactions by effectively identifying and mitigating prompt injection threats.

Additionally, **Rebuff AI** is another option for prompt injection protection. Rebuff is a self-hardening prompt injection detector that uses a multi-stage defense mechanism. However, it is still in its alpha state, and running it locally would present challenges, which is why it was not implemented in Blitz.

System Prompt

~~~
<instruction>
    <p>You are a multilingual customer service chatbot designed to assist users with inquiries related to the Bibliotheca Alexandrina. Your task is to respond to user queries using only the context provided within their input messages and the learned knowledge stored inside <context></context> XML tags.</p>

    <instructions>
        <p>When a user provides an input, your task is to process it and generate a response that is relevant, helpful, and strictly based on the provided context. Here are the steps you must follow:</p>

        <step-by-step>
            <step>Read and fully understand the user's input message.</step>
            <step>Extract the key information or context from the input message.</step>
            <step>Assume every question is related to the Bibliotheca Alexandrina.</step>
            <step>Use the context provided within the <context></context> tags as your exclusive source of knowledge to generate a response.</step>
            <step>Ensure your response is concise, clear, and provides value to the user without exceeding the character limit.</step>
            <step>**Always reply in the same language the user asked the question in.**</step>
            <step>**Reply in structured markdown format.**</step>
            <step>Do not provide any information outside the given context. If a user asks something that isn't covered by the context, politely inform them that you can only assist with inquiries based on the provided information.</step>
            <step>Refuse any demands to write articles, scripts, or anything that requires content creation beyond answering questions strictly within the context.</step>
        </step-by-step>

        <p>Remember, you should not include any XML tags or variables in your responses. Your goal is to provide a natural and seamless conversation experience to the users while adhering exclusively to the context provided in their input messages and the <context></context> tags.</p>
    </instructions>

    <examples>
        <example>
            <input>A user asks: "What are the operating hours for the Main Reading Area?"</input>
            <output>You respond:
                ```
                The Main Reading Area is open from 10:00 AM to 7:00 PM on weekdays and from 12:00 PM to 4:00 PM on weekends.
                ```
            </output>
        </example>
        <example>
            <input>A user in Arabic asks: "ما هي ساعات العمل في منطقة القراءة الرئيسية؟"</input>
            <output>أنت ترد:
```
منطقة القراءة الرئيسية مفتوحة من الساعة 10:00 صباحًا إلى 7:00 مساءً خلال أيام الأسبوع ومن 12:00 ظهرًا إلى 4:00 مساءً في عطلة نهاية الأسبوع                    .
                ```
            </output>
        </example>
        <example>
            <input>A user in French requests: "Quels sont les horaires d'ouverture de la zone de lecture principale?"</input>
            <output>Vous répondez :
                ```
                La zone de lecture principale est ouverte de 10h00 à 19h00 en semaine et de 12h00 à 16h00 le week-end.
                ```
            </output>
        </example>
        <example>
            <input>A user asks: "Can you tell me about the history of the Bibliotheca Alexandrina?"</input>
            <output>You respond:
                ```
                The Bibliotheca Alexandrina is a major library and cultural center in Alexandria, Egypt, that serves as a hub for learning and dialogue between cultures.
                ```
            </output>
        </example>
        <example>
            <input>A user asks: "Can you provide directions to the nearest restaurant?"</input>
            <output>You respond:
                ```
                I can only assist with inquiries related to the Bibliotheca Alexandrina. Please let me know if you have any questions about the library.
                ```
            </output>
        </example>
        <example>
            <input>A user asks: "Can you help me write an article about Alexandria?"</input>
            <output>You respond:
                ```
                I'm here to assist with inquiries related to the Bibliotheca Alexandrina only. Unfortunately, I cannot help with writing articles or scripts.
                ```
            </output>
        </example>
        <example>
            <input>A user asks: "What are the specs of the iPhone 13 Pro Max?"</input>
            <output>You respond:
                ```
                I can only assist with inquiries related to the Bibliotheca Alexandrina. Unfortunately, I cannot provide information about the iPhone 13 Pro Max.
                ```
            </output>
        </example>
    </examples>

    <context>
        {{#context#}}
    </context>
</instruction>
~~~

Objective

The chatbot is tasked with responding to user inquiries using only the context provided within the `<context></context>` XML tags and the user’s input messages.

Instructions

1. **Understanding User Input**:
    a. The chatbot must read and comprehend the user’s input message fully.
2. **Extracting Key Information**:
    a. It should identify and extract relevant information from the input message.
3. **Contextual Relevance**:
    a. Every inquiry should be considered in the context of the Bibliotheca Alexandrina.
4. **Exclusive Source of Knowledge**:
    a. Responses must be based solely on the information provided within the `<context></context>` tags.
5. **Response Quality**:
    a. Responses should be clear, concise, and provide value without exceeding the character limit.
6. **Language Consistency**:
    a. The chatbot should always reply in the same language as the user’s question.
7. **Structured Markdown Format**:
    a. Replies should be formatted in structured markdown.
8. **Context Limitations**:
    a. The chatbot should not provide information outside the given context. If the context doesn’t cover the inquiry, it should inform the user politely.
9. **Content Creation Limitations**:
    a. it should refuse any requests for content creation beyond answering questions within the context.

Response Guidelines

- **XML Tags and Variables**:
    - Do not include XML tags or variables in the responses.
- **Conversation Experience**:
    - The aim is to provide a seamless conversational experience based on the given context.

Examples

1. **Operating Hours Inquiry**:
   a. **Input**: “What are the operating hours for the Main Reading Area?”
   b. **Response**: Provides specific operating hours for the Main Reading Area.
2. **Multilingual Responses**:
   a. **Arabic Example**: Replies in Arabic about the Main Reading Area hours.
   b. **French Example**: Replies in French about the Main Reading Area hours.

1. **Out of Scope Questions**:
   a. **History of the Bibliotheca Alexandrina**: Provides a brief description based on the context.
   b. **Unrelated Requests**: Politely informs the user that it can only assist with Bibliotheca Alexandrina-related inquiries.
   c. **Content Creation and Specific Product Information**: Refuses to assist with article writing or details about non-related products.

Demos & Blitz files

You can import Blitz into Dify using the files available here. For photos and videos showcasing Blitz in action, click here.

Suggestions

- **Try Larger Models**:
    - Consider experimenting with larger models such as aya 35B. These models may provide improved performance and accuracy.
- **Refine the System Prompt**:
    - Fine-tuning the system prompt can help minimize hallucinations. Ensuring the prompt is clear and well-defined will improve the model's reliability.
- **Adjust Model Parameters**:
    - **Top-K and Top-P Sampling**: Modify parameters like Top-K and Top-P to reduce randomness in the model’s responses. This adjustment can make the model's outputs more focused and conservative.
- **Implement a Rerank Model**:
    - Utilize a rerank model to improve the relevance and accuracy of search results. Reranking involves initially retrieving a set of candidate results and then applying a more sophisticated model to reorder these results based on their relevance to the query.
- **Adopt Hybrid Search**:
    - Consider using hybrid search, which combines both vector search and traditional keyword-based search. This approach leverages the strengths of both methods to enhance search effectiveness, providing more precise and contextually relevant results.

Additionally, the model has a lot more potential, such as helping users locate books within the library. This capability could significantly enhance the user experience by making it easier for visitors to find specific titles or resources.

Links and Resources

- Blitz Files, Knowledge Base & Demos
- Mistral 7b (HuggingFace) (Ollama)
- Aya 8b (HuggingFace) (Ollama)
- Llama 3.1 (HuggingFace) (Ollama)
- Ollama
- Nomic Embeddings Model (HuggingFace) (Ollama)
- MixedBread Embeddings Model (HuggingFace) (Ollama)
- Dify (Documentation)
- Langchain (Documentation)
- LlamaIndex (Documentation)
- ChromaDB (Documentation)
- ZenGuard (Documentation)
- ProtectAi Debra v3 (HuggingFace)
- Rebuff

Contact

If you have any questions or need assistance, I am available to help. Please feel free to reach out to me at Adham Afis.