AI-POWERED SPLUNK GUIDE

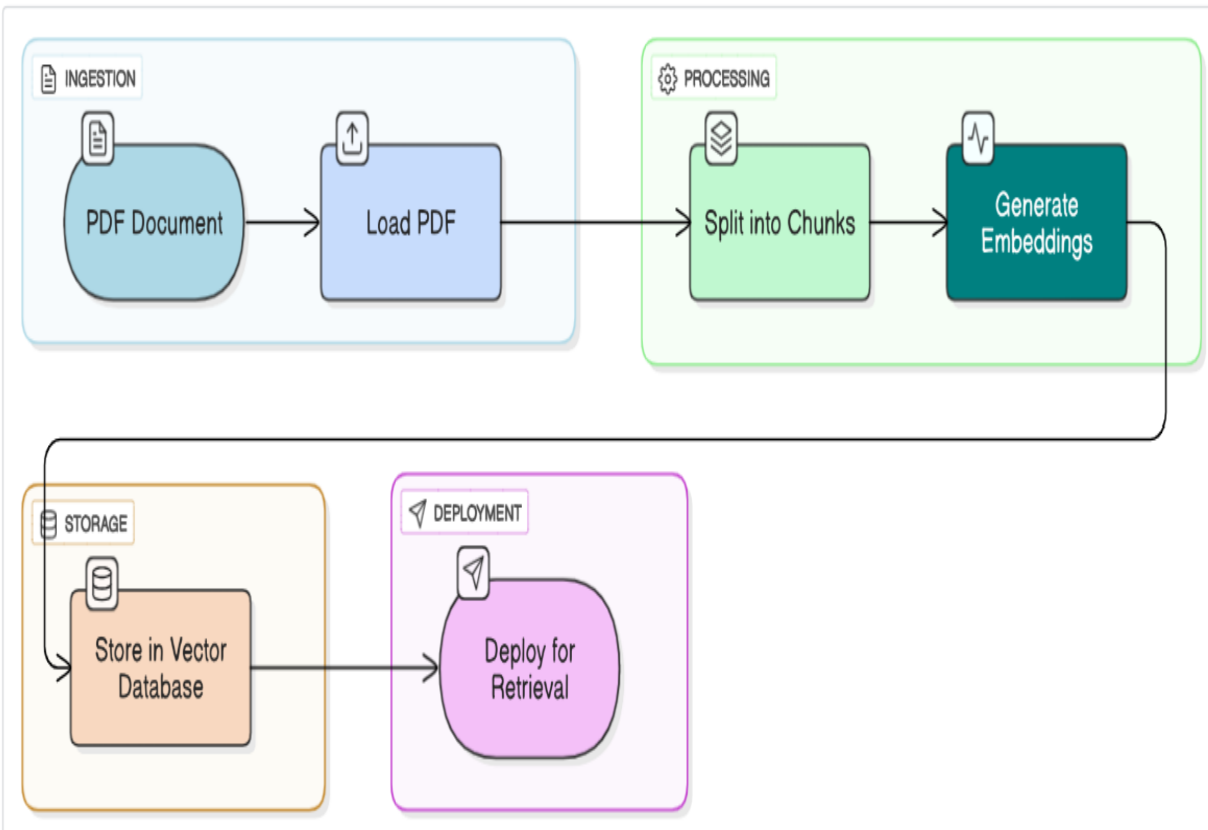
By: Abdelrahman Moustafa Attia

AUGUST 20, 2025
BARQ SYSTEMS

Table of Contents

1. Overview

This project implements an intelligent **Splunk Assistant** that can answer user questions about Splunk configuration, clustering, and best practices. It combines **document ingestion**, **chunking**, **vector-based semantic search**, and **large language model reasoning** to deliver accurate, context-aware answers.



2. Data Ingestion(PDF Reader)

When designing the system, one of the critical steps was choosing a reliable PDF reader to ingest the *Splunk Admin Guide (v8.2.1)*. Since this document is large (hundreds of pages, with both prose and configuration snippets), the PDF loader needed to:

- Accurately extract continuous text (paragraphs, lists, code blocks).
- Handle performance efficiently (fast loading, minimal memory leaks).
- Remain compatible with LangChain document loaders.

Three main libraries were tested:

2.1 PyPDF

- **Pros:**
  - Widely used, lightweight.
  - Works well for basic text extraction.
- **Cons:**
  - Breaks paragraphs into fragmented strings.
  - Loses formatting (newlines in wrong places, missing spacing).
  - No direct support for preserving layout or metadata.
- **Verdict:** Not suitable for technical manuals like Splunk Admin Guide where context continuity is critical.

2.2 pdfplumber

- **Pros:**
  - Excellent for structured tabular data extraction.
  - Can directly parse rows, columns, and bounding boxes.
  - Useful if the goal is data analysis from invoices, reports, or logs.

- **Cons:**
  - Performs poorly with continuous text:
    - Breaks sentences across lines.
    - Introduces extra whitespace.
    - Loses natural reading flow.
  - Slower on large PDFs.
- **Verdict:** Ideal for tabular extraction, but not good enough for a documentation-heavy text-based project like this.

2.3. PyMuPDF → Final Choice

- **Pros:**
  - Fast, lightweight, and highly optimized for text extraction.
  - Preserves paragraph flow better than other libraries.
  - Provides metadata support (page numbers, positions, coordinates).
  - Stable with large PDFs (Splunk Admin Guide ~800+ pages loaded cleanly).
  - Integrates seamlessly with LangChain's PyMuPDFLoader.
- **Cons:**
  - Slightly more complex API compared to PyPDF2.
  - Extracted text may still need minor cleaning (spaces around newlines).
- **Verdict:** Chosen as the production-ready loader since it balances performance and accuracy for Splunk's manual.

➔ **Final Decision**

After experimentation:

- **pdfplumber** was kept in consideration only for cases where structured tables in Splunk logs or reports needed extraction.
- **pymupdf** was selected as the main ingestion tool because this project's primary focus is on narrative/configuration text rather than tabular datasets.

3. Text Chunking

Once the Splunk Admin Guide was successfully loaded with **PyMuPDF**, the next challenge was splitting the document into meaningful **chunks** for embedding and retrieval. Large language models (LLMs) cannot process an entire 800+ page manual at once, so an effective chunking strategy was necessary.

3.1 Why Chunking is Important

- **Token Limitations** → Most LLMs have a context window (e.g., Mistral ~8K tokens). Without chunking, entire pages would overflow.
- **Retrieval Accuracy** → Smaller, semantically relevant chunks improve similarity search in FAISS.
- **Answer Precision → If chunks are too large → irrelevant noise enters context. If too small → fragmented meaning.**

Thus, the goal was to find a balance: large enough chunks for context, but small enough to stay within LLM limits.

3.2 Chunking Approaches

3.2.1 Recursive Character Text Splitter (Final Choice)

- **Mechanism:** Splits text hierarchically at natural boundaries:
    1. Headings
    2. Paragraphs
    3. Sentences
    4. Characters
- **Parameters Used:**
    - chunk_size = 1000 characters
    - chunk_overlap = 200 characters (ensures continuity across splits)
- **Pros:**
    - Preserves semantic coherence (paragraphs remain intact).
    - Handles edge cases where sentences break across chunks.
    - Compatible with LangChain out-of-the-box.

- **Cons:**
    - Chunks may still break mid-configuration line.
    - Overlap increases storage slightly, but improves QA accuracy.
- **Verdict:** Chosen as the primary splitter for production.

3.2.2. Semantic Text Splitter (with embeddings)

- **Mechanism: Uses embeddings (sentence-transformers/all-mpnet-base-v2) to split at semantically meaningful points (topic shifts).**
- **Pros:**
    - High-quality, context-aware splits.
    - Avoids splitting mid-topic (e.g., clustering setup remains in one chunk).
- **Cons:**
    - Much slower for large documents (~800 pages).
    - Heavy compute cost (requires embedding at split time).
    - Risk of over-segmentation if semantic thresholds are too strict.
- **Verdict:** Good for research papers or structured articles, but too slow for our use case. Rejected for production.

4. Embedding Models Tested

4.1 all-mpnet-base-v2 (Semantic Splitter Test)

- **Strengths**: Very high semantic accuracy in retrieval.
- **Weaknesses**: Heavy (~420MB), slower inference.
- **Result**: Overkill for this use case.

4.2 all-MiniLM-L6-v2 (Final Choice)

- **Strengths:**
  - Lightweight (~80MB).
  - Fast inference.
  - Good balance of semantic accuracy vs. performance.
- **Weaknesses:** Slightly less precise than MPNet in long documents.
- **Result: Chosen for embedding chunks due to speed and efficiency, especially when serving via FAISS.**

➔ **Final Decision**

- **Recursive Text Splitter + all-MiniLM-L6-v2 embeddings was chosen for production.**
- **This provides the best trade-off between:**
  - Semantic accuracy.
  - Processing speed.
  - Resource efficiency.

5. Vector Store (Indexing & Retrieval)

After splitting the document into semantically coherent chunks and embedding them into vector space, the next step was to **store and retrieve these embeddings efficiently**.

5.1 Why a Vector Store is Needed

- **Direct in-memory lists** (e.g., just storing embeddings in a Python array) work for small-scale experiments but quickly fail when:
    - The dataset grows (our Splunk Admin Guide has **hundreds of chunks**).
    - You need fast similarity search (linear scans become **O(n)**, too slow).
    - Persistence is required (in-memory data vanishes after restart).

Thus, a dedicated **Vector Store** was required to:

1. Index embeddings efficiently.
2. Support **Approximate Nearest Neighbor (ANN)** search.
3. Handle scalability and persistence.

5.2 FAISS

- **What is FAISS?**
    - FAISS = **Facebook AI Similarity Search**.
    - Open-source library for **efficient vector similarity search and clustering**.
    - Optimized in C++ with GPU support.
- **Implementation in this project**:
    - Store all embeddings of Splunk text chunks in a FAISS index.
    - Use cosine similarity (normalized embeddings) for retrieval.
- **Pros**:
    - **Very fast ANN search** (logarithmic time instead of linear).
    - Scales to millions of embeddings.
    - Easy integration with LangChain (FAISS.from_texts).
    - Supports persistence (can save/load FAISS index).

6. LLM Setup with Mistral

After preparing the Splunk Admin Guide dataset (loading, chunking, embedding, and indexing with FAISS), the final step was to connect it to a **Large Language Model (LLM)** that could understand user queries and generate human-like, context-aware answers.

For this project, the **Mistral 7B** model was chosen, deployed locally via **Ollama**.

6.1 Why Mistral?

Several LLMs were considered (e.g., OpenAI GPT models, LLaMA 2, Falcon), but **Mistral** was selected because:

- **Lightweight & Efficient**: Mistral-7B is small enough to run on a local machine (via Ollama/Colab-Xterm) while still providing high-quality answers.
- **Open Source**: No dependency on paid APIs like OpenAI or Anthropic.
- **Strong Reasoning Ability**: Compared to LLaMA 2 7B, Mistral has shown better performance in reasoning tasks and technical QA.
- **Private Deployment**: Since the model runs locally, all Splunk documentation stays secure without being sent to external servers.

6.2 Deployment with Ollama

Instead of hosting Mistral manually with complex dependencies, the project used **Ollama**, a tool that simplifies downloading and serving LLMs locally.

Steps Taken:

1. Installed Ollama in Colab-Xterm:
2. !curl -fsSL https://ollama.com/install.sh | sh
3. Started Ollama server in the background:

   ollama serve &
4. Pulled the Mistral model:

   ollama pull mistral
5. Verified setup by running a simple test prompt:

   from langchain_community.chat_models import ChatOllama

   llm = ChatOllama(model="mistral")

print(llm.invoke("Hello, name three red flowers."))

```
llm.invoke("Hello, tell me 3 names of red flowers.")
```

```
AIMessage(content=' Hello! Here are three examples of red flower species:\n\n1. Red Rose: Roses come in many colors, but the
classic red rose is one of the most popular and recognized red flowers.\n\n2. Carnation: Often associated with love and
admiration, carnations can be found in a variety of red shades.\n\n3. Poppy: The bright red poppy is a common sight in fields
during springtime and is the national flower of England.', additional_kwargs={}, response_metadata={'model': 'mistral',
'created_at': '2025-08-20T08:21:24.192361432Z', 'message': {'role': 'assistant', 'content': ''}, 'done_reason': 'stop', 'done':
True, 'total_duration': 8716561157, 'load_duration': 5279989619, 'prompt_eval_count': 15, 'prompt_eval_duration': 761123351,
'eval_count': 100, 'eval_duration': 2674458733}, id='run--c1642847-d0f0-4634-bbd8-75906b996415-0')
```

6.3 Integration with LangChain

The LLM was integrated into the pipeline using **LangChain**, which allowed combining:

- **Prompt engineering**
- **Memory management**
- **Document retrieval**
- **Answer formatting**

Custom Prompt Template

A **system prompt** was designed to ensure that answers:

- Stay **grounded in the Splunk Admin Guide** (avoid hallucination).
- Follow a structured format with four sections:
    1. **Overview**
    2. **Steps / How It Works**
    3. **Examples**
    4. **Notes / Best Practices**

This is my prompt

```
   "system",
        """You are a friendly and knowledgeable Splunk Assistant.
Your job is to answer the user's questions **only** using the provided
context.

Always reply in a clear, structured format with the following sections (if
possible):
1. **Overview** → Short explanation of the concept.
2. **Steps / How It Works** → Bullet points or ordered steps.
3. **Examples** → Code/config snippets or practical usage.
4. **Notes / Best Practices** → Warnings, defaults, or tips.

- If the context contains the answer → explain it clearly and concisely,
following the above structure.
- If the context does not contain the answer → reply politely with:
"I'm sorry, I couldn't find that information in the Splunk Admin Guide.
You might want to check the official Splunk documentation."

Always keep your answers professional, accurate, and easy to understand.
Use examples whenever possible.
"""
```

6.4 Memory Setup

To make the chatbot feel conversational, **ConversationBufferMemory** was added:

- Stores **past questions and answers**.

- Enables follow-up queries like:

  - Q1: *"What is index clustering?"*
  - Q2: *"Can you explain the steps again?"*

- Without memory, the LLM would forget context after each query.

6.5 Retrieval-Augmented Generation (RAG)

The final QA pipeline combined:

1. **User Query**
2. **FAISS Vector Store Search** → Retrieve top-k most relevant Splunk chunks.
3. **Mistral LLM + Custom Prompt** → Generate structured answer.
4. **Memory Buffer** → Maintain conversation history.

7. Deployment & Testing

7.1 Gradio User Interface

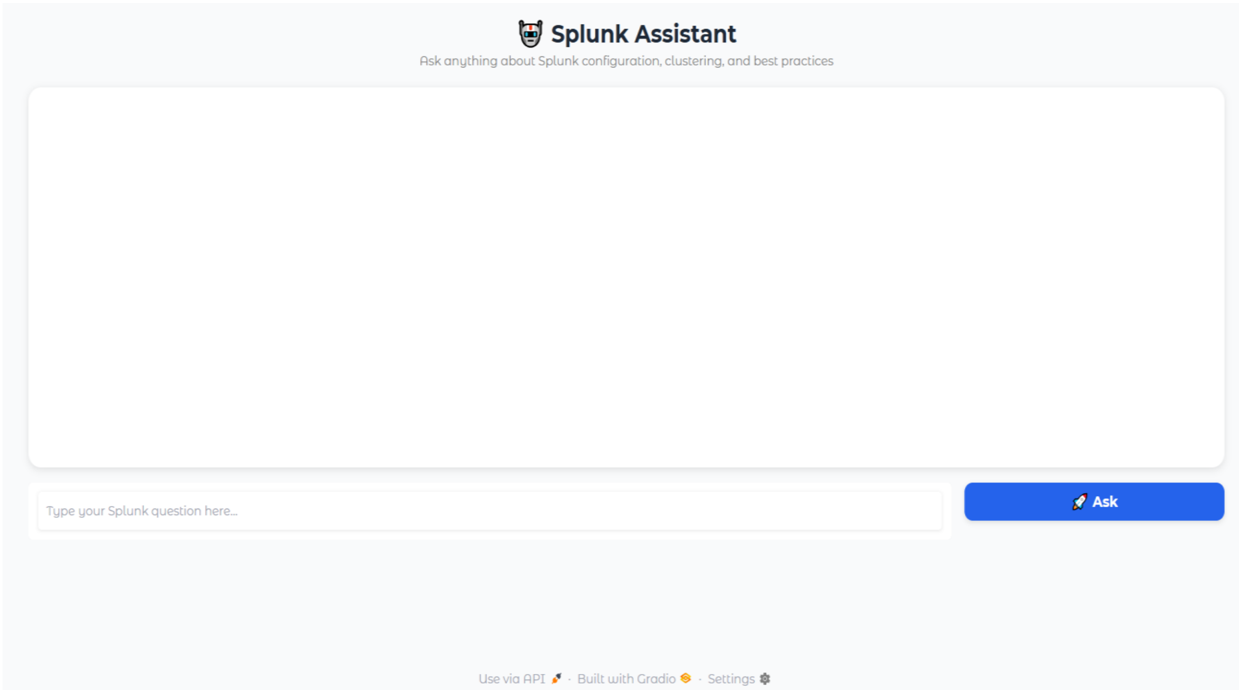
Once the document retrieval and LLM pipeline were working, the next step was to design a **user-friendly interface** where Splunk admins or learners could interact with the chatbot easily. For this, **Gradio** was chosen as the front-end framework.

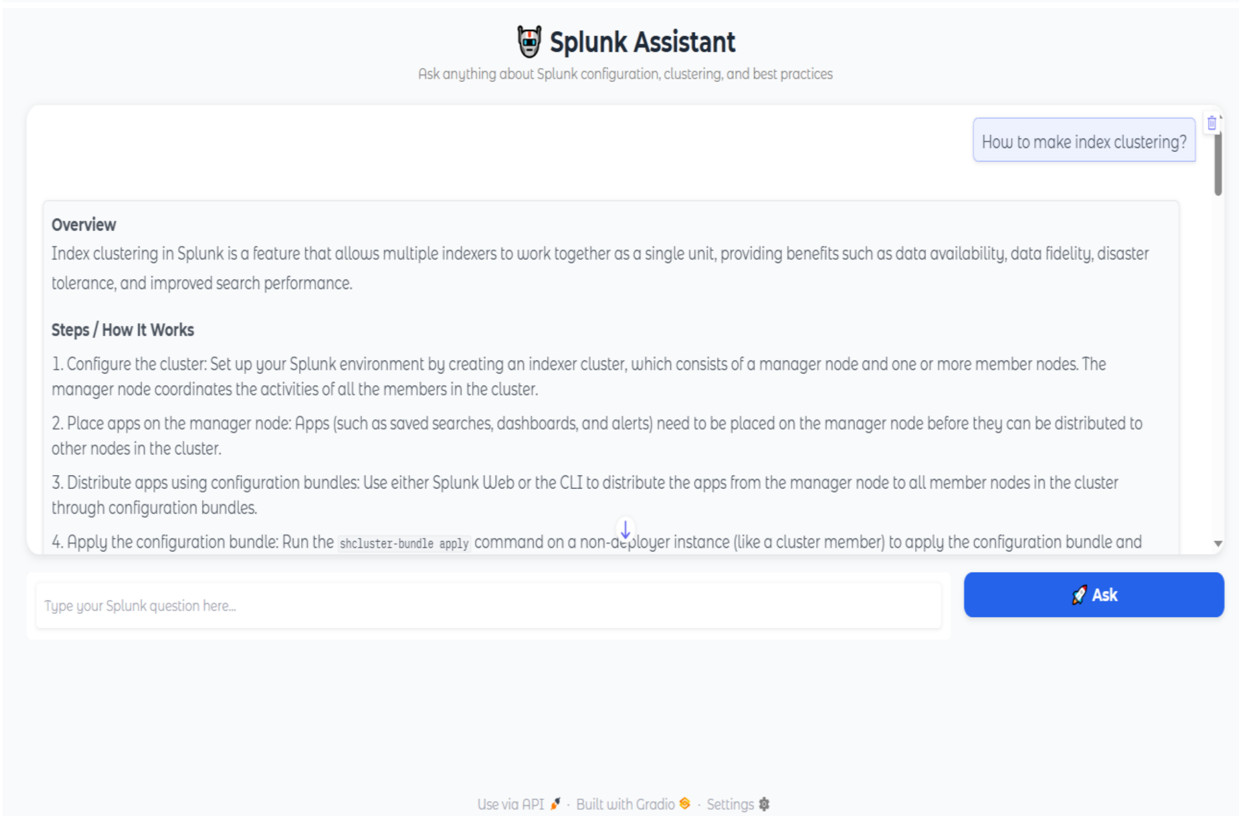
7.2 Design Goals

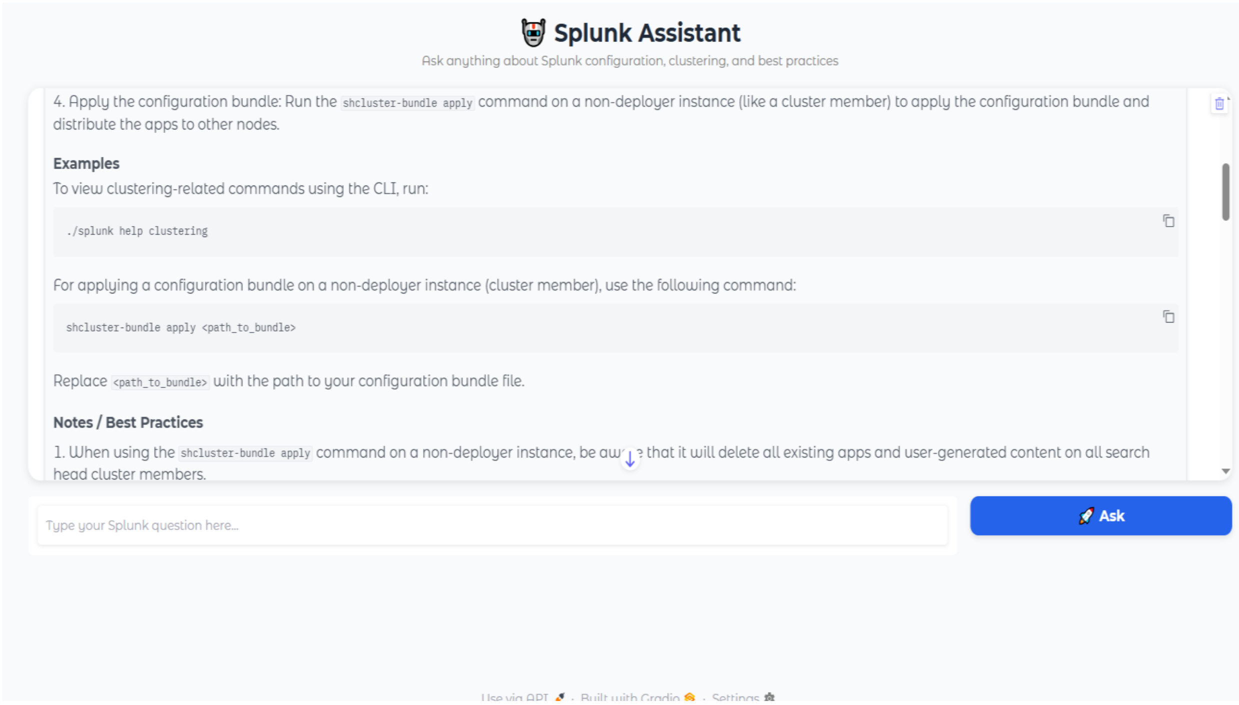
The UI design followed three main goals:

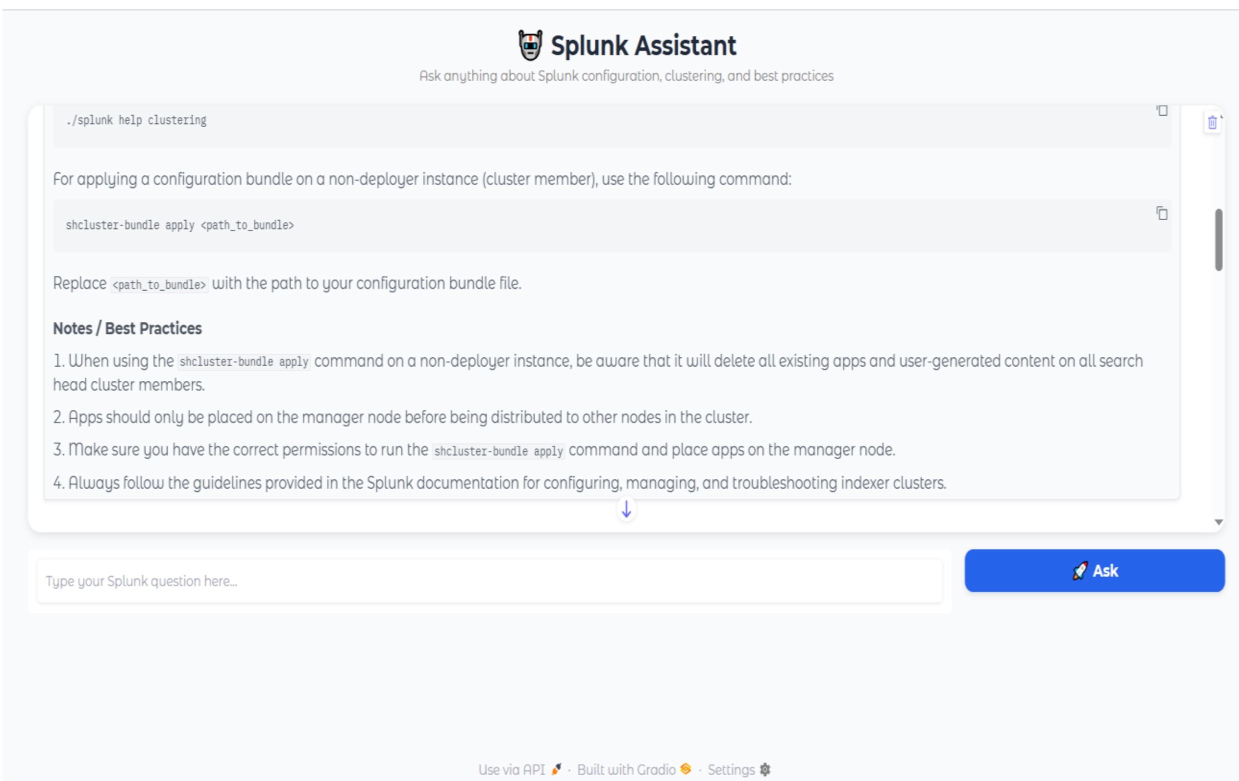
1. **Clean & Professional Layout**
    - Since this project is a technical assistant (Splunk Admin Guide), the interface needed to look **minimalist and business-like**, avoiding clutter.
    - Used **centered title, subtle colors, and clean typography** for a professional appearance.
    - Avoided overwhelming users with extra controls — only the chatbot and input box are visible.
2. **Responsive Chatbot with Styled UI**
    - The chat window was styled with **rounded corners, soft shadows, and smooth spacing** to mimic modern chat applications.
    - Responsive design ensures it works well on different screen sizes (desktop, tablet, mobile).
    - The chatbot UI supports displaying **multi-turn conversations** (memory-enabled), making it feel like a real assistant.
3. **Clear Input Box + "Ask" Button**
    - At the bottom, a **large input field** is provided where users can type Splunk-related questions.
    - Next to it, a **bold "🚀 Ask" button** makes the interaction intuitive.
    - Keyboard support (press Enter to submit) was also included for smooth workflow.

7.3 screenshots






Splunk Assistant
Ask anything about Splunk configuration, clustering, and best practices
4. Apply the configuration bundle: Run the shcluster-bundle apply command on a non-deployer instance (like a cluster member) to apply the configuration bundle and distribute the apps to other nodes.
Examples
To view clustering-related commands using the CLI, run:
./splunk help clustering
For applying a configuration bundle on a non-deployer instance (cluster member), use the following command:
shcluster-bundle apply <path_to_bundle>
Replace <path_to_bundle> with the path to your configuration bundle file.
Notes / Best Practices
head cluster members.
Type your Splunk question here...
Ask


Splunk Assistant
Ask anything about Splunk configuration, clustering, and best practices
./splunk help clustering
For applying a configuration bundle on a non-deployer instance (cluster member), use the following command:
shcluster-bundle apply <path_to_bundle>
Replace <path_to_bundle> with the path to your configuration bundle file.
Notes / Best Practices
1. When using the shcluster-bundle apply command on a non-deployer instance, be aware that it will delete all existing apps and user-generated content on all search head cluster members.
2. Apps should only be placed on the manager node before being distributed to other nodes in the cluster.
3. Make sure you have the correct permissions to run the shcluster-bundle apply command and place apps on the manager node.
4. Always follow the guidelines provided in the Splunk documentation for configuring, managing, and troubleshooting indexer clusters.
Type your Splunk question here...
Ask
Use via API · Built with Gradio · Settings

8. Conclusion

In this project, we successfully developed a **Splunk Assistant Chatbot** that can intelligently answer user queries about Splunk administration and configuration by leveraging modern NLP and vector search techniques. The workflow integrated **document loading**, **text chunking**, **semantic embeddings**, **FAISS vector storage**, and **Mistral LLM inference** into a seamless pipeline, with a user-friendly **Gradio interface** for interaction.

Key takeaways include:

- **PDF Loading & Preprocessing** → After experimenting with multiple libraries (PyPDF, Plumber, PyMuPDF), we chose **PyMuPDF** for its reliability in text extraction.
- **Chunking & Embeddings** → Recursive chunking combined with **all-MiniLM-L6-v2 embeddings** provided a balance between performance and accuracy.
- **Vector Store with FAISS** → Enabled efficient similarity search, allowing the chatbot to retrieve the most relevant Splunk documentation sections.
- **LLM Setup with Mistral** → Integrated via **Ollama**, offering lightweight yet powerful reasoning to generate structured, human-readable answers.
- **Gradio UI** → Delivered a professional and intuitive chatbot interface, making the system accessible to both technical and non-technical users.

Overall, the project demonstrates how **domain-specific assistants** can be built using open-source tools to bridge the gap between complex documentation and practical user needs. The Splunk Assistant not only simplifies access to Splunk knowledge but also provides a blueprint for building similar assistants in other domains.

Future enhancements could include:

- Expanding support to multiple Splunk versions and additional datasets.
- Improving context retention with advanced memory mechanisms.
- Deploying the chatbot on cloud platforms for broader accessibility.