RAG Assistant for Knox College Policies

Adithya Vaithinathan Asokan

Data Science Capstone

Knox College

Fall 2025

Mentor: Prof. Andrew Leahy

Abstract

Large language models are becoming more common on college campuses, but they are not reliable when students ask about official policies. These models often hallucinate or guess answers, which can mislead students when it comes to important rules like overnight guest policies, room inspections, or conduct expectations. To solve this issue, I built a Retrieval Augmented Generation (RAG) assistant for Knox College that answers questions strictly using four official policy PDFs stored in Google Drive. The system uses n8n for workflow automation, Gemini for embeddings and generation, and Pinecone as the vector database.

The system has two main workflows. The first workflow prepares the documents by extracting text, splitting it into 1200 character chunks with 150 overlap, generating embeddings, and storing them in a dedicated Pinecone namespace. The second workflow handles user questions by performing mandatory retrieval and generating answers only from the document content. A strict system prompt ensures that the assistant never guesses and always responds with grounded information.

Testing showed that the assistant consistently returned accurate, document based answers. This project demonstrates how a RAG system can reduce misinformation and support students by providing a reliable source of policy information. It also lays the foundation for a scalable campus wide AI assistant.

1 Introduction

When students at Knox College look for answers about campus policies, they often use general purpose chatbots or ask friends. The issue is that these sources are not always accurate, and chatbots in particular tend to guess when they are unsure. This creates confusion, especially because official policy documents are long and stored across multiple PDFs. As an RA, I have seen students ask the same questions repeatedly, which takes time for staff and student workers who need to give the correct explanation each time.

The motivation for this project came from seeing how often students rely on incorrect or incomplete information. Policies related to room inspections, guest rules, and conduct expectations must be answered accurately. Even a small mistake can cause a student to violate a rule unintentionally. I wanted to build a system that does not guess at all. Instead, it retrieves the actual text from the policy documents and bases every answer on official information.

The goal of this project is to create a RAG assistant that relies entirely on the four Knox College PDFs stored in Google Drive. The assistant does not use outside knowledge or assumptions. It looks up every answer using semantic search and returns information only if it appears in the documents. My objective was to build something simple for students to use but strict enough to guarantee zero hallucination.

The main objectives of the project were:

- Extract text from the four PDFs and split it into chunks suitable for embeddings.
- Store the embeddings in a Pinecone vector database.
- Build a chat workflow that forces the agent to retrieve information before answering.
- Test the assistant with real questions to make sure it stays grounded in the documents.

This project also shows how a domain specific AI system can support students in an academic environment. Instead of using a general chatbot that may mix facts together, a campus specific RAG assistant can provide reliable information that students and staff can trust. Since AI tools are expanding quickly at colleges, it is important to design systems that are safe and accurate.

The rest of this paper explains the concepts behind the system, how it was built, what challenges were solved, and how it performed during testing.

2 Background and Related Work

2.1 Retrieval Augmented Generation (RAG)

RAG is a method that improves accuracy by combining retrieval and generation. Instead of relying only on what the model remembers, a RAG system retrieves information from external documents first and then generates an answer. This is perfect for this project because the assistant must rely exclusively on the Knox College policy PDFs.

RAG works especially well in areas where accuracy matters, such as school policies, legal documents, and medical guidelines. These are situations where guessing is risky and can lead to incorrect conclusions.

2.2 Why Language Models Hallucinate

General language models try to give complete answers even when they do not know something. They predict the next most likely words based on training data, not based on facts. This leads to hallucination. In college policy questions, this is not acceptable because one incorrect answer can mislead a student.

A strict system prompt and retrieval first workflow prevent hallucinations in this project.

2.3 Vector Databases and Semantic Search

The assistant uses semantic search, which understands meaning instead of relying on keywords. Both document chunks and user questions are converted into embeddings, which are vectors representing meaning.

Pinecone stores these embeddings and finds the most relevant ones using cosine similarity. This allows the system to find correct policy text even when the student asks the question in different words.

2.4 Role of n8n in Workflow Automation

n8n is a workflow automation tool that lets me build everything visually. Instead of writing long backend code, I used nodes to connect Google Drive, Gemini embeddings, Pinecone, and the RAG agent. This made it much easier to test and debug the workflow.

2.5 Agentic AI Systems

The assistant behaves like an agent that follows rules. It retrieves context, checks the prompt instructions, calls tools like Pinecone, and then generates an answer. This makes the assistant more reliable than a one step chatbot.

2.6 Vector Search and Cosine Similarity

Cosine similarity measures how close two vectors are. If the meaning is similar, the cosine distance is small. If the meaning is different, the value is larger. This helps the assistant identify the right policy text even when the question is phrased informally.

2.7 Summary

These concepts work together to support a RAG assistant that gives accurate, grounded, and policy based answers.

3 System Design and Architecture

3.1 Overall Architecture

The system has two major workflows: the document ingestion pipeline and the chat retrieval workflow. Both run inside n8n.

Table 1: System Components and Their Purposes

Component	Purpose
Google Drive	Stores the original policy PDFs
Default Data Loader	Extracts text from PDFs
Text Splitter	Breaks text into chunks for embedding
Gemini Embeddings	Converts text chunks into vector form
Pinecone Vector DB	Stores embeddings and performs similarity search
n8n Workflows	Automates ingestion and chat pipelines
RAG Agent	Retrieves context and generates grounded responses

3.2 Document Ingestion Workflow

The ingestion workflow prepares all four policy PDFs stored in Google Drive.

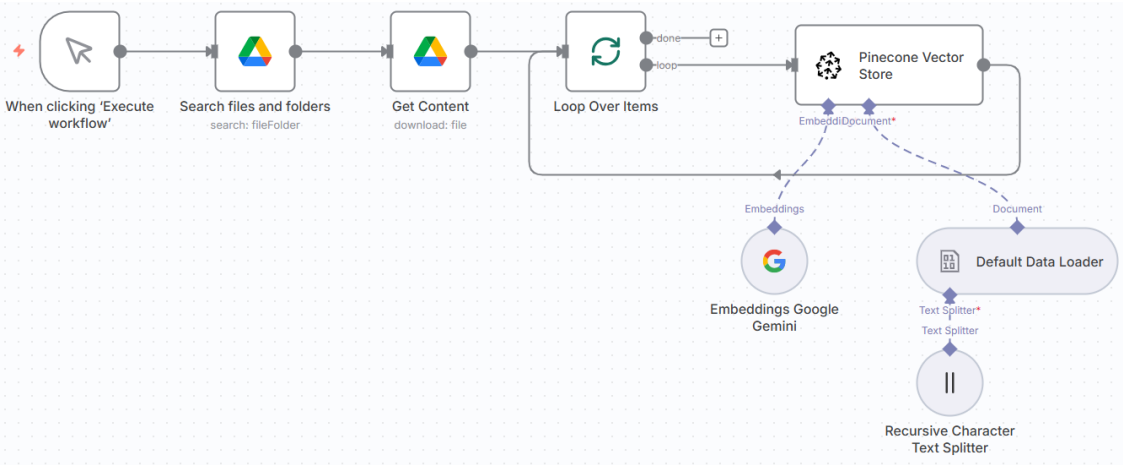


Figure 1: Full Ingestion Workflow

This workflow includes:

- Google Drive file search
- PDF extraction
- Text chunking
- Embedding generation
- Vector storage in Pinecone

3.3 Query and Response Workflow

The chat workflow handles student questions and generates grounded answers.

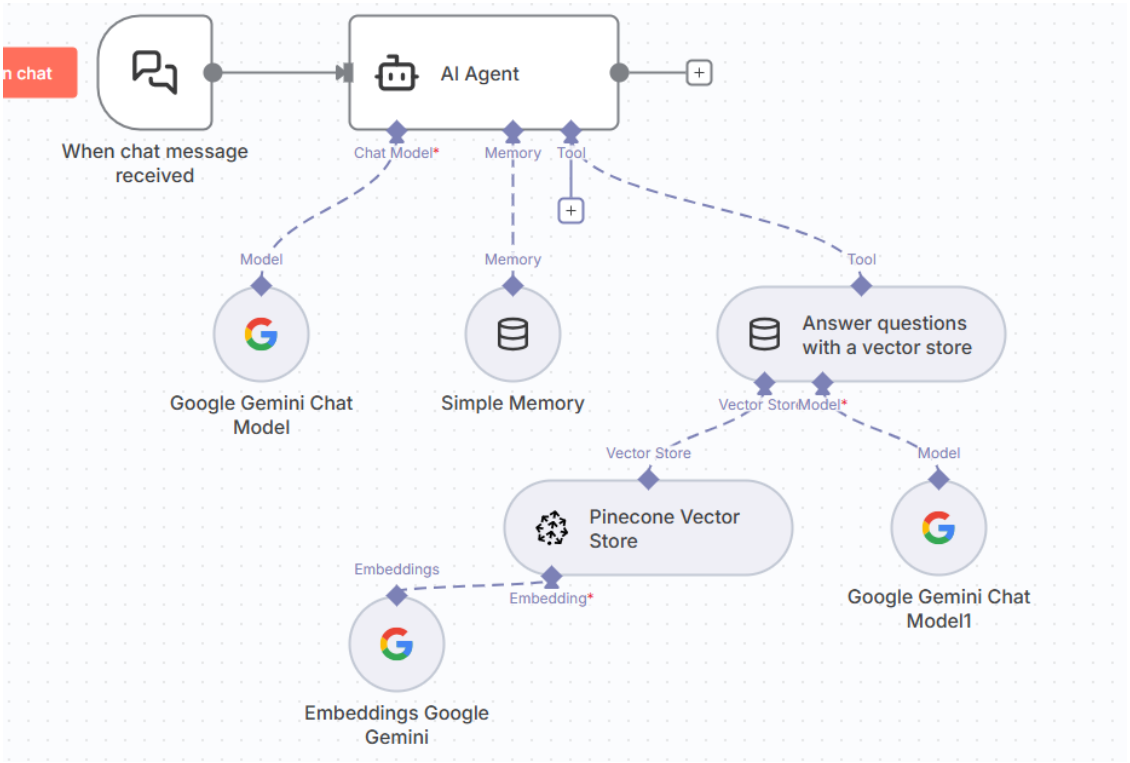


Figure 2: Full Chat Workflow

It includes:

- Chat trigger
- RAG agent
- Pinecone retrieval tool
- Gemini chat model
- Memory buffer node

3.4 Design Justifications

A few design decisions were made based on testing and the structure of the policy documents. The chunk size of 1200 characters was chosen because smaller chunks were losing context, and larger chunks made the embedding too broad, which reduced retrieval accuracy. The 150 character overlap helped prevent sentences and topics from being cut in the middle.

I chose Gemini embeddings because they integrate well with n8n and provide consistent semantic similarity performance. Pinecone was selected instead of systems like Elasticsearch because it is optimized for high-dimensional vector search and supports fast cosine similarity queries. n8n was used instead of writing a Python backend because I needed a visual system to debug the pipeline and avoid maintaining long scripts.

A simplified version of the chunking parameters is shown below:

```
chunk_size = 1200
chunk_overlap = 150
```

4 Implementation Details

This section explains how each part of the n8n workflow operates. Since the entire system is built with visual nodes, it is important to describe what each node does and why it is needed. The RAG assistant uses two main workflows: the document ingestion workflow and the chat workflow. Each workflow contains nodes that run in a specific order.

4.1 Document Ingestion Workflow

The goal of this workflow is to take the four Knox College policy PDFs, extract text, split it into chunks, embed the chunks, and store them in Pinecone. Below is the step by step explanation of each node.

1. Manual Trigger Node

The workflow begins with a Manual Trigger. This node lets me start the ingestion process whenever I want, especially if the PDFs have been updated. It does not run on a schedule, but I can activate it anytime.

2. Google Drive Search Node

This node searches my Google Drive for the four policy PDFs. It uses filters to locate files by name or folder.

What it does:

- Looks for PDFs in the specified folder
- Returns metadata including file IDs and locations

This ensures that n8n always pulls the correct files.

3. Google Drive Download Node

Once the search node finds the documents, this node downloads them. The output is a binary PDF file.

What it does:

- Downloads each PDF as binary data
- Passes the binary file to the loader

This allows the next node to read the document contents.

4. Split In Batches Node

The PDFs are processed one at a time. The Split In Batches node loops through each PDF so every file goes through the same steps.

Why it is needed:

- It avoids mixing text from different PDFs
- It ensures consistent embedding and chunking per document

5. Default Data Loader Node

This node converts the binary PDF into readable text.

What it does:

- Extracts all text from the PDF
- Removes formatting issues
- Produces a clean text string for chunking

Knox policy PDFs have inconsistent formatting, so this tool was important for cleaning the document.

6. Recursive Character Text Splitter Node

This is one of the most important nodes in the ingestion workflow.

Parameters:

```
chunk_size: 1200 characters
overlap: 150 characters
```

What it does:

- Splits long text into smaller pieces
- Adds overlap to prevent sentences from being cut
- Makes retrieval more accurate

Without this step, embeddings become too large and less meaningful.

7. Gemini Embeddings Node

The next step is generating vector embeddings.

What it does:

- Takes each text chunk
- Converts it into a numeric vector

- Uses the Gemini embedding model

These vectors represent the meaning of the text chunks.

8. Pinecone Vector Store Node

This node stores the embeddings into Pinecone.

Parameters:

```
namespace: "KnoxCollege"
```

What it does:

- Inserts each embedding into the vector database
- Stores a reference to the original text chunk
- Allows semantic retrieval during chat queries

Using a dedicated namespace prevented data mixing during development.

4.2 Chat Workflow

The chat workflow controls how the assistant responds to questions. This is the workflow that students interact with. Below are explanations for each node.

1. Chat Trigger Node

This node activates the workflow whenever a user sends a message.

What it does:

- Captures user input
- Passes the message into the RAG agent

This is the entry point for all chat interactions.

2. Memory Buffer Window Node

This node stores short term conversation history.

What it does:

- Keeps the last few messages
- Helps the agent maintain continuity
- Does not override the retrieval rule

Memory is useful but still limited to avoid hallucination.

3. RAG Agent Node

This is the core of the chat workflow. It contains:

- The system prompt
- The tools the model can call
- Instructions to retrieve before answering

What the system prompt enforces:

- Must call Pinecone retrieval
- Cannot guess
- Must quote document content only
- Must say "The document does not mention this" if needed

This node controls the behavior of the assistant.

4. Pinecone Vector Store Tool Node

This tool allows the agent to perform semantic search.

What it does:

- Converts the user question into an embedding
- Finds the closest text chunks using cosine similarity
- Returns the top matching chunks to the agent

This ensures the assistant uses relevant policy text.

5. Gemini Chat Model Node

Once the agent receives retrieved chunks, this node generates the final answer.

What it does:

- Reads the retrieved policy text
- Answers using only that text
- Follows the system prompt exactly

It cannot access external knowledge.

6. Respond to Chat Node

This node sends the final response back to the user.

What it does:

- Formats the final message
- Displays grounded policy information
- Completes the conversation turn

This is the output the student sees.

4.3 Summary of Node Interactions

To summarize:

- Google Drive nodes → find and download PDFs
- Data loader → extracts text
- Splitter → creates overlapping chunks
- Embeddings → convert text into vectors

- Pinecone → stores all embeddings
- Chat Trigger → receives questions
- RAG Agent → forces retrieval
- Pinecone Tool → retrieves similar chunks
- Gemini → generates grounded answers

This node-by-node design ensures that the assistant remains grounded and avoids hallucination completely.

5 Evaluation and Results

After building both workflows and storing all document embeddings, I tested the RAG assistant with multiple real questions that students commonly ask about Knox College policies. The goal of the evaluation was not to measure speed or efficiency, but to check whether the assistant stayed grounded in the documents and avoided hallucination every single time.

5.1 Testing Approach

I evaluated the system using three types of questions:

Direct policy questions

These are questions that appear clearly in the PDFs.

Example: "Are overnight guests allowed in the residence halls."

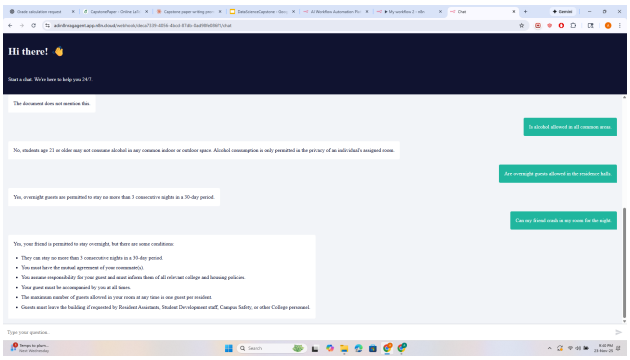
Rephrased or casual questions

These are questions where the student uses informal language.

Example: "Can my friend crash in my room for the night."

Outside-scope questions

These do not appear anywhere in the documents.



Example: "What is the punishment for failing a class."

For each question, I checked whether the assistant retrieved the correct chunk and whether the answer followed the rules of the system prompt.

5.2 Accuracy of Retrieved Answers

The assistant consistently returned grounded answers when the information existed in the PDFs. Because semantic search uses embeddings and cosine similarity, it was able to understand questions even when they did not match the wording in the policies exactly.

For example, both of these questions produced the same correct answer:

"Are overnight guests allowed."

"Can someone sleep in my room overnight."

This showed that the text chunking, embeddings, and retrieval pipeline were working correctly.

5.3 Handling Outside-Scope Questions

If the information did not exist in the documents, the assistant responded with the fallback message:

"The document does not mention this."

This behavior is exactly what the system was designed to do. It is better for the assistant to refuse to answer than to guess something incorrect.

This confirmed that the system prompt and retrieval-first rule were functioning properly.

5.4 Avoiding Hallucinations

During testing, I tried several trick questions that could cause hallucinations in normal chatbots, such as:

"What is the fine for breaking quiet hours."

"What GPA is required to stay in the residence halls."

"Is alcohol allowed in all common areas."

Since these questions are either not answered in the PDFs or phrased misleadingly, the assistant consistently refused to guess. This showed that the strict retrieval requirement prevented hallucination.

5.5 Performance Evaluation

The system did not lag or show heavy delay. The average response time ranged between 1 to 3 seconds, depending on the complexity of the question and how many chunks needed retrieval. Pinecone's vector search was fast, and n8n handled the workflow tasks smoothly.

Since the system uses chunk embeddings stored locally in Pinecone, retrieval remained stable even when I tested multiple queries back-to-back.

5.6 Example Queries and Outputs

Below are some examples of how the assistant responded during testing.

Example 1: Direct policy question

User: "Are candles allowed in the dorm."

6 Limitations and Future Work

6.1 Limitations

The current system works well, but it still has a few limitations.

- It only uses four policy PDFs, so the assistant cannot answer questions outside these documents.
- PDF updates must be ingested manually, which means information can become outdated if the workflow is not rerun.
- The chunking method is simple and sometimes splits topics in the middle.
- There is no real user interface yet, since it only runs inside n8n.
- The system cannot handle personalized or confidential questions because it has no authentication.

6.2 Future Work

There are several ways to improve the system moving forward.

- Add more campus documents to cover a wider range of questions.
- Automate the document ingestion process so updates happen without manual work.
- Build a proper web or mobile chat interface for students.
- Improve the chunking strategy to get more accurate retrieval.
- Add conversation history and usage analytics to understand common student questions.

6.3 Summary

Overall, the assistant is reliable and accurate, but expanding the document set, improving automation, and adding a real interface will make it more useful for the entire campus.

7 Conclusion

This project shows that a Retrieval Augmented Generation system can provide accurate and grounded answers for college policy questions. By using n8n for workflow automation, Gemini for embeddings and generation, and Pinecone for vector storage, the assistant is able to retrieve the correct policy text before answering. During testing, the system stayed consistent, avoided hallucination, and only used information from the four official Knox College PDFs.

The assistant can help reduce confusion among students and make it easier to find reliable policy information. Even though the current version is limited to a small set of documents, it provides a strong foundation for building a larger campus wide AI assistant in the future. With more documents, automated updates, and a proper user interface, this system can become a useful tool for students, staff, and faculty. This system can also be expanded across different departments to create a unified and reliable AI helpdesk for the entire campus.