**Policy Chat Assistant POC**

**Cortex AI**

**Implementation Documentation**

**Contents**

[1. Executive Summary 4](#_Toc203658970)

[2. System Architecture Overview 4](#_Toc203658971)

[**2.1 Core Components** 4](#_Toc203658972)

[**2.2 Technology Stack** 4](#_Toc203658973)

[3. Implementation Details 5](#_Toc203658974)

[**3.1 Document Storage and Stage Setup** 5](#_Toc203658975)

[**3.2 Raw Text Extraction Process** 5](#_Toc203658976)

[**3.2.1 Document Parsing Implementation:** 5](#_Toc203658977)

[**3.2.2 Key Features** 6](#_Toc203658978)

[4. Chunk Generation Process 6](#_Toc203658979)

[**4.1 Chunking Implementation** 6](#_Toc203658980)

[**4.1.1 Chunking Parameters** 6](#_Toc203658981)

[**4.1.2 Document Categorization** 6](#_Toc203658982)

[**4.1.3 Automatic Classification** 6](#_Toc203658983)

[5. Cortex Search Service Creation 7](#_Toc203658984)

[**5.1 Service Features** 7](#_Toc203658985)

[6. Chat Assistant Application Components 7](#_Toc203658986)

[**6.1 Search and Retrieval Logic** 7](#_Toc203658987)

[**6.2 Chat History Management** 8](#_Toc203658988)

[**6.3 Response Generation** 8](#_Toc203658989)

[7. Identified Limitations and Improvements 9](#_Toc203658990)

[**7.1 Content Structure Challenges** 10](#_Toc203658991)

[**7.2 Future Enhancements** 10](#_Toc203658992)

[8. Recommendations and Best Practices 11](#_Toc203658993)

[**8.1. Document Format Guidelines** 11](#_Toc203658994)

[**8.2 Proposed Content Processing Strategy** 11](#_Toc203658995)

[**8.3 System Usage Guidelines** 11](#_Toc203658996)

[**8.3.1 Legal and Contract Documents** 11](#_Toc203658997)

[**8.4 Technical Optimizations** 12](#_Toc203658998)

[**8.4.1 Performance Recommendations** 12](#_Toc203658999)

[**8.4.2 NUM\_CHUNKS Parameter Tuning** 12](#_Toc203659000)

[9. Monitoring and Maintenance 13](#_Toc203659001)

[10. Cost Analysis and Resource Planning 13](#_Toc203659002)

[**10.1 Snowflake Credit Consumption** 13](#_Toc203659003)

[**10.1.1 Document Processing Costs** 13](#_Toc203659004)

[**10.1.2 Search Service Costs** 13](#_Toc203659005)

[**10.1.3 AI Response Generation Costs** 13](#_Toc203659006)

[11. Maintenance and Training 14](#_Toc203659007)

[**11.1 Regular Maintenance Tasks** 14](#_Toc203659008)

[**11.2 Technical Maintenance** 14](#_Toc203659009)

[**11.3 Maintenance Skill Requirements** 15](#_Toc203659010)

[**11.4 End User Training Requirements** 15](#_Toc203659011)

[**11.4.1 Basic User Training** 15](#_Toc203659012)

[**11.4.2 Technical User Training** 16](#_Toc203659013)

[**11.5 Training Materials Needed** 16](#_Toc203659014)

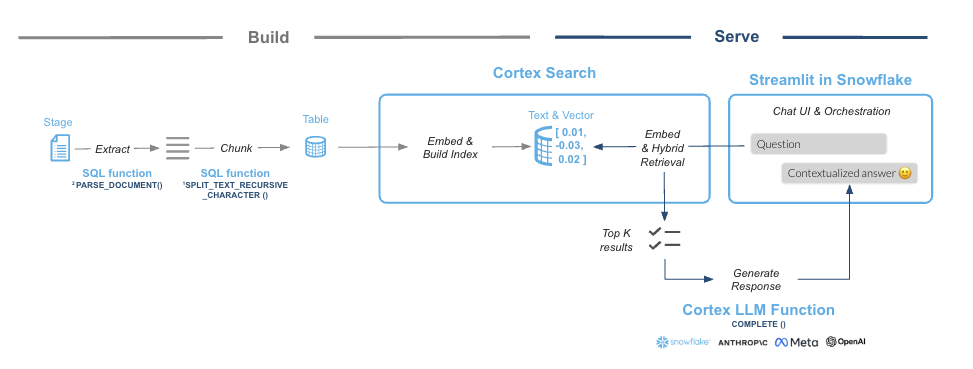
[12. Conclusion 16](#_Toc203659015)

# **1. Executive Summary**

The Policy Chat Assistant is a Snowflake-based RAG (Retrieval-Augmented Generation) system designed to help users query organizational policy documents through natural language interactions.

The system leverages Snowflake's Cortex AI capabilities for document processing, text chunking, semantic search, and response generation. This documentation outlines the complete implementation, identified limitations, and recommendations for optimal usage.

# **2. System Architecture Overview**



## **2.1 Core Components**

* **Platform**: Snowflake with Cortex AI capabilities
* **Frontend**: Snowflake Native – Streamlit application with chat interface
* **Document Storage**: Snowflake internal stage with encryption
* **Text Processing**: Snowflake Cortex PARSE\_DOCUMENT with layout-based extraction
* **Search Engine**: Snowflake Cortex Search Service with semantic search capabilities, including automatic text embeddings generation, vector similarity matching, and intelligent ranking algorithms that understand context and meaning rather than just keyword matching.
* **LLM Integration**: Multiple model support with configurable selection

## **2.2 Technology Stack**

These are the main software libraries (tools) that the chat application uses:

* **streamlit**: Creates the web interface where users can chat and interact with the system
* **snowflake.snowpark**: Connects the app to the Snowflake database where documents are stored
* **snowflake.cortex**: Provides access to AI functions for search and text generation
* **snowflake.core**: Helps manage database objects and services
* **pandas**: Handles data tables and organization
* **json**: Processes structured data that the search service returns

# **3. Implementation Details**

## **3.1 Document Storage and Stage Setup**

The stage is configured with:

* **Encryption**: Snowflake Server-Side Encryption for security
* **Directory Support**: Enables file browsing and management
* **File Access**: Supports presigned URLs for document downloads

## **3.2 Raw Text Extraction Process**

### **3.2.1 Document Parsing Implementation:**

Here**,** a temporary table that processes all PDF files in stage is created, which also extracts their text content. Here's what happens step by step:

1. **RELATIVE\_PATH**: Gets the file name and location
2. **SIZE**: Records size of each file
3. **FILE\_URL**: Creates a link to access the original file
4. **build\_scoped\_file\_url**: Creates a secure, temporary download link
5. **SNOWFLAKE.CORTEX.PARSE\_DOCUMENT**: The PARSE\_DOCUMENT function is a Cortex AI function that reads the document and converts it to text, using 'LAYOUT' mode which tries to preserve how the document is organized (headers, paragraphs, etc.).

The following table lists the limitations and requirements of input documents for the same:

|  |  |
| --- | --- |
| Maximum file size | 100 MB |
| Maximum pages per document | 300 Pages |
| Allowed file type | PDF, PPTX, DOCX, JPEG, JPG, PNG, TIFF, TIF |
| Stage encryption | Server-side encryption |
| Font size | 8 point or larger for best results |

1. **EXTRACTED\_LAYOUT**: Stores the converted text content

CREATE OR REPLACE TEMPORARY TABLE RAW\_TEXT AS

SELECT

RELATIVE\_PATH,

SIZE,

FILE\_URL,

build\_scoped\_file\_url(@policy\_documents, relative\_path) as scoped\_file\_url,

TO\_VARCHAR(

SNOWFLAKE.CORTEX.PARSE\_DOCUMENT(

'@policy\_documents',

RELATIVE\_PATH,

{'mode': 'LAYOUT'}

):content

) AS EXTRACTED\_LAYOUT

FROM DIRECTORY('@policy\_documents');

### **3.2.2 Key Features**

* **Layout Mode**: Preserves document structure better than OCR mode
* **Scoped URLs**: Generates secure, time-limited access URLs
* **Metadata Capture**: Stores file size, path, and URL information
* **Content Extraction**: Converts PDF content to structured text format

# **4. Chunk Generation Process**

Each row in this table represents one piece of text with information about where it came from.

## **4.1 Chunking Implementation**

CREATE OR REPLACE TABLE POLICY\_DOCS\_CHUNKS (

RELATIVE\_PATH VARCHAR(16777216), -- Document file path

SIZE NUMBER(38,0), -- File size in bytes

FILE\_URL VARCHAR(16777216), -- Public file URL

SCOPED\_FILE\_URL VARCHAR(16777216), -- Secure access URL

CHUNK VARCHAR(16777216), -- Text chunk content

CHUNK\_INDEX INTEGER, -- Sequential chunk identifier

CATEGORY VARCHAR(16777216) -- Document classification

);

### **4.1.1 Chunking Parameters**

* **Chunk Size**: 1512 characters for optimal context retention
* **Overlap**: Overlap 256 characters across two consecutive chunks to maintain continuity
* **Separators**: Hierarchical splitting preserving document structure
* **Format**: Markdown-aware processing for better text organization

### **4.1.2 Document Categorization**

This step organizes documents into categories based on their file names.

1. Gets a list of all unique documents (using the first chunk from each document)
2. Extracts category from filename:
   * If the filename contains a dash (-), it takes everything before the first dash as the category
   * For example: "HR-Employee-Handbook.pdf" would be categorized as "HR"
   * If there's no dash, it marks the document as "UNCATEGORIZED"
3. Apply to all chunks: Finally, it updates every chunk in the database with the category of its parent document

The UPDATE statement then makes sure every piece of text (chunk) knows which category it belongs to, so users can filter their searches by document type.

### **4.1.3 Automatic Classification**

docs\_category\_cte AS (

SELECT

relative\_path,

CASE

WHEN CONTAINS(relative\_path, '-') THEN

SPLIT\_PART(SPLIT\_PART(relative\_path, '/', -1), '-', 1)

ELSE 'UNCATEGORIZED'

END AS category

FROM unique\_documents

)

# **5. Cortex Search Service Creation**

This creates an intelligent search engine that can find relevant text chunks when users ask questions. It is a smart search function for documents that understands meaning, not just keywords;

1. **CORTEX SEARCH SERVICE**: Creates an AI-powered search engine
2. **ON chunk**: Searches through the text content of each chunk
3. **ATTRIBUTES category**: Allows filtering results by document category (HR, Finance, etc.)
4. **WAREHOUSE = POC**: Assigns computing resources to run the search
5. **TARGET\_LAG = '1 day'**: Updates the search index daily to include any new documents

CREATE OR REPLACE CORTEX SEARCH SERVICE POLICY\_SEARCH\_SERVICE

ON chunk

ATTRIBUTES category

WAREHOUSE = POC

TARGET\_LAG = '1 day'

AS (

SELECT

chunk,

chunk\_index,

relative\_path,

file\_url,

category

FROM POLICY\_DOCS\_CHUNKS

);

**Note**: The **TARGET\_LAG** can be set as desired, and the search service must be refreshed to reflect updates:

ALTER CORTEX SEARCH SERVICE POLICY\_SEARCH\_SERVICE REFRESH;

## **5.1 Service Features**

* **Semantic Search**: Vector-based similarity search on chunk content
* **Attribute Filtering**: Category-based filtering for targeted searches
* **Warehouse Assignment**: Dedicated compute resources for search operations
* **Refresh Schedule**: Daily updates to maintain search index currency

# **6. Chat Assistant Application Components**

## **6.1 Search and Retrieval Logic**

def get\_similar\_chunks\_search\_service(query):

if st.session\_state.category\_value == "ALL":

response = svc.search(query, COLUMNS, limit=NUM\_CHUNKS)

else:

filter\_obj = {"@eq": {"category": st.session\_state.category\_value}}

response = svc.search(query, COLUMNS, filter=filter\_obj, limit=NUM\_CHUNKS)

return response.json()

This function finds the most relevant text chunks to answer a user's question

1. **Check category filter:** If the user selected "ALL", it searches through every document
2. **Apply category filter:** If they selected a specific category (like "HR"), it only searches documents in that category
3. **Search execution:** It uses the AI search service to find the most relevant chunks based on meaning, not just keywords
4. **Limit results:** It returns only the top n most relevant chunks (NUM\_CHUNKS, typically 3) to avoid overwhelming the AI with too much information
5. **Return results:** Sends back the search results in a structured format

## **6.2 Chat History Management**

def get\_chat\_history():

chat\_history = []

start\_index = max(0, len(st.session\_state.messages) - slide\_window)

for i in range(start\_index, len(st.session\_state.messages) - 1):

chat\_history.append(st.session\_state.messages[i])

return chat\_history

def summarize\_question\_with\_history(chat\_history, question):

prompt = f"""

Based on the chat history below and the question, generate a query that extends the question

with the chat history provided. The query should be in natural language.

Answer with only the query. Do not add any explanation.

<chat\_history>{chat\_history}</chat\_history>

<question>{question}</question>

"""

return Complete(st.session\_state.model\_name, prompt).replace("'", "")

This manages the conversation memory so the chat assistant can understand context from previous questions:

1. **get\_chat\_history()**
   * Remembers the last few conversation exchanges (controlled by slide\_window, typically 7 messages)
2. **summarize\_question\_with\_history():** 
   * Takes the current question and the recent chat history
   * Uses AI to combine them into a better search query that includes context

This allows for natural conversations where users don't have to repeat context.

## **6.3 Response Generation**

A custom prompt is used to guide the LLM to generate answers in natural language based on the provided context.

**Note**: This prompt can be modified to include any specific instructions for answer generation as a guideline, for example – response length, warnings etc.

def create\_prompt(myquestion):

prompt = f"""

You are an expert chat assistant that extracts information from the CONTEXT provided

between <context> and </context> tags.

You offer a chat experience considering the information included in the CHAT HISTORY

provided between <chat\_history> and </chat\_history> tags.

When answering the question contained between <question> and </question> tags

be concise and do not hallucinate.

If you don´t have the information, just say so.

Do not mention the CONTEXT used in your answer.

Do not mention the CHAT HISTORY used in your answer.

Only answer the question if you can extract it from the CONTEXT provided.

<chat\_history>{chat\_history}</chat\_history>

<context>{prompt\_context}</context>

<question>{myquestion}</question>

Answer:

"""

return prompt, relative\_paths, chunks\_data

This creates detailed instructions for the AI to generate helpful and accurate responses:

1. **Sets the AI's role:** Instructs the AI to act as an expert assistant that pulls information from documents
2. **Provides context:** Gives the AI the relevant text chunks found by the search, plus recent conversation history
3. **Sets behavior rules:** 
   * Be concise and clear
   * Don't make up information ("don't hallucinate")
   * If the answer isn't in the provided documents, say so
   * Don't mention that it's using context or chat history (keeps the response natural)
   * Only answer based on the actual document content provided
4. **Structured format:** Organizes everything with clear tags so the AI knows what's what:
   * <chat\_history>: Recent conversation
   * <context>: Relevant document chunks
   * <question>: The user's current question

# **7. Identified Limitations and Improvements**

## **7.1 Content Structure Challenges**

During implementation, we identified that certain document structures, particularly complex layouts with structured data presentations like tables, posed challenges for the text chunking process. Several approaches were attempted to address these issues.

Attempted Solutions:

1. **OCR Mode Testing**: Switched from LAYOUT to OCR mode in document parsing

{'mode': 'OCR'} -- Results: Significantly inferior text quality

1. **Enhanced Preprocessing**: Implemented content detection and formatting

CASE

WHEN CONTAINS(EXTRACTED\_LAYOUT, '| :---: | :---: |') THEN

CONCAT('TABLE CONTENT: ', REPLACE(...))

ELSE EXTRACTED\_LAYOUT

END

1. **Chunking Parameter Optimization**: Adjusted chunk sizes and overlap

SPLIT\_TEXT\_RECURSIVE\_CHARACTER(

cleaned\_format, 'markdown',

1200, -- Reduced from 1512

200, -- Adjusted overlap

['\n\n', '\n', ' ', '']

)

1. **Content Structure Enhancement**: Improved formatting preservation

REPLACE(REPLACE(EXTRACTED\_LAYOUT, '|', ' | '), ' | | ', ' and ')

**Results**: While these approaches provided some improvements for the specific use case, fundamental challenges remained with complex structured tabular content preservation during the chunking process.

## **7.2 Future Enhancements**

**Case-by-case document processing**

As a future enhancement, we could implement intelligent document analysis that automatically detects document complexity and structure before processing.

This would allow the system to apply different processing strategies based on each document's characteristics.

for example, using specialized table extraction for data-heavy documents, optimized chunking parameters for narrative content, or hybrid approaches for mixed-format documents.

Such adaptive processing would significantly improve content preservation and retrieval accuracy across diverse document types.

# **8. Recommendations and Best Practices**

## **8.1. Document Format Guidelines**

**Optimal Document Characteristics**

* **Simple layouts** with primarily narrative content
* **Minimal complex structures** that may fragment during processing
* **Clear hierarchical organization** using headers and sections
* **Consistent formatting** throughout documents

## **8.2 Proposed Content Processing Strategy**

1. **Quality Assessment**: Evaluate document complexity before processing
2. **Format Optimization**: Convert complex layouts to simpler formats when possible
3. **Manual Review**: Flag documents with unusual structures for review
4. **Dual Processing**: Consider separate handling for different content types

## **8.3 System Usage Guidelines**

### **8.3.1 Legal and Contract Documents**

End users should NOT rely 100% on results generated by Cortex AI for legal documents, contracts, or any content requiring precise legal interpretation.

The system should be used as a preliminary research tool only for such specific use cases.

**Note**: For each answer, the chat interface displays links to the referenced documents from stage and details of the specific text chunks used, making it easier to verify that the correct semantic information has been retrieved.

**Legal Content Limitations**

* **AI-generated responses** may miss critical legal nuances
* **Document chunking** may separate important contextual information
* **Legal interpretation** requires human expertise and complete document review
* **Compliance verification** must be performed by qualified legal professionals

**Recommended Workflow for Legal Content**

1. **Initial Research**: Use the chat assistant for preliminary information gathering
2. **Source Verification**: Always review the original document sections referenced
3. **Professional Review**: Consult legal experts for interpretation and compliance
4. **Complete Context**: Read full documents rather than relying on extracted chunks

## **8.4 Technical Optimizations**

### **8.4.1 Performance Recommendations**

* **Regular index refresh** to maintain search accuracy
* **Warehouse sizing** appropriate for expected query volume
* **Category-based filtering** to improve search relevance
* **Chunk overlap optimization** for better context preservation

### **8.4.2 NUM\_CHUNKS Parameter Tuning**

The NUM\_CHUNKS parameter in the code controls how many relevant text pieces the system retrieves to answer each question.

This is currently set to 3, but users can experiment with different values to improve response quality.

NUM\_CHUNKS = 3 *# Current setting - retrieves 3 most relevant text chunks*

When the user asks a question, the search engine finds the most relevant pieces/chunks of text from the documents and sends them to the AI for generating an answer.

NUM\_CHUNKS determines how many of these pieces to include.

**How to experiment with NUM\_CHUNKS**

1. **Start with the default**: Try questions with NUM\_CHUNKS = 3
2. **Increase for complex questions**: If answers seem incomplete or missing important details, try increasing to 5 or 7
3. **Decrease for simple questions**: If answers are too verbose or include irrelevant information, try reducing to 1 or 2
4. **Test systematically**: Ask the same question with different NUM\_CHUNKS values and compare the responses

**Impact on answer quality**

* **Too few chunks (1-2)**: May miss important context or provide incomplete answers
* **Optimal range (3-5)**: Usually provides good balance of relevant information without overwhelming the AI
* **Too many chunks (8+)**: May include irrelevant information, confuse the AI, or slow down responses

**Validation approach**

For critical questions, especially those related to compliance or legal matters, test with different NUM\_CHUNKS values (try 3, 5, and 7) to ensure getting comprehensive and consistent answers. If responses vary significantly, this indicates the need for human verification of the source documents.

# **9. Monitoring and Maintenance**

**Ongoing System Health**

* **Query performance monitoring** for response times
* **Search result quality assessment** through user feedback
* **Document processing error tracking** for continuous improvement
* **Regular system updates** as Cortex capabilities evolve

# **10. Cost Analysis and Resource Planning**

## **10.1 Snowflake Credit Consumption**

The Policy Chat Assistant utilizes several Snowflake AI features, each with specific credit consumption rates. Understanding these costs is essential for budget planning and resource optimization.

### **10.1.1 Document Processing Costs**

For initial document processing, cost is based on the number of pages processed.

**Parse Document – Layout**: 3.33 Credits per 1,000 pages

**Parse Document – OCR**: 0.5 Credits per 1,000 pages

Layout mode (which we use) costs more than OCR mode because it provides better quality extraction.

For example, processing 100 policy documents averaging 10 pages each (1,000 total pages) would cost 3.33 credits using Layout mode.

### **10.1.2 Search Service Costs**

Monthly cost is based on how much text data is stored in the search index.

**Cortex Search:** 6.3 Credits per GB/month of indexed data

A typical policy document collection (50-100 documents) usually requires less than 1 GB of indexed data, resulting in approximately 6.3 credits per month for search capability.

### **10.1.3 AI Response Generation Costs**

Each conversation with the AI consumes credits based on the model used and the length of input/output. Here are costs for some popular LLMs;

|  |  |  |
| --- | --- | --- |
| **Method** | **Large Language Model** | **Credits per 1 million tokens** |
| AI Complete | llama3.1-70b | 1.21 |
| AI Complete | llama3.1-8b | 0.19 |
| AI Complete | mistral-7b | 0.12 |
| AI Complete | mixtral-8x7b | 0.22 |

**What this means**: A typical question-answer exchange uses 500-2,000 tokens depending on context length and response detail. More powerful models like llama3.1-70b cost more but may provide better responses.

**Cost Optimization Strategies**

* **Model Selection**: Use smaller models (llama3.1-8b, mistral-7b) for simple queries to reduce costs
* **NUM\_CHUNKS Tuning**: Optimize the number of chunks retrieved to balance response quality with token consumption
* **Category Filtering**: Encourage users to filter by document category to improve search efficiency
* **Document Preprocessing**: Process documents once during initial setup rather than repeatedly

# **11. Maintenance and Training**

## **11.1 Regular Maintenance Tasks**

**Daily/Automatic**:

* **Search Index Refresh**: Configured with TARGET\_LAG = '1 day' for automatic updates
* **System Monitoring**: Check for any processing errors or performance issues

**Weekly**:

* **Performance Review**: Monitor query response times and user satisfaction
* **Credit Usage Tracking**: Review Snowflake credit consumption patterns
* **Error Log Analysis**: Check for any document processing or search failures

**Monthly**:

* **Document Updates**: Process any new policy documents added to the system
* **Category Management**: Review and update document categorization as needed
* **Usage Analytics**: Analyze user query patterns and popular document sections

**Quarterly**:

* **System Optimization**: Review NUM\_CHUNKS settings and model performance
* **Cost Analysis**: Evaluate credit usage and optimize model selection
* **User Feedback Review**: Collect and analyze user experience feedback
* **Document Cleanup**: Remove outdated documents and update existing ones

## **11.2 Technical Maintenance**

**Database Maintenance**

*-- Regular cleanup of temporary tables*

DROP TABLE IF EXISTS RAW\_TEXT;

DROP TABLE IF EXISTS docs\_categories;

*-- Refresh search service if needed*

ALTER CORTEX SEARCH SERVICE POLICY\_SEARCH\_SERVICE REFRESH;

**Monitoring Queries**

*-- Check chunk distribution*

SELECT category, COUNT(\*) as chunk\_count

FROM POLICY\_DOCS\_CHUNKS

GROUP BY category;

*-- Monitor document processing status*

SELECT relative\_path, COUNT(\*) as chunks\_generated

FROM POLICY\_DOCS\_CHUNKS

GROUP BY relative\_path;

## **11.3 Maintenance Skill Requirements**

**Basic Level (Day-to-day operations)**:

* Understanding of Snowflake credit monitoring
* Basic SQL knowledge for status checking
* Familiarity with Streamlit application management

**Intermediate Level (Weekly/Monthly tasks)**:

* SQL proficiency for data analysis and cleanup
* Understanding of search service configuration
* Experience with document processing workflows

**Advanced Level (System optimization)**:

* Snowflake administration experience
* Performance tuning and optimization skills
* AI model selection and parameter tuning knowledge

## **11.4 End User Training Requirements**

### **11.4.1 Basic User Training**

**Session 1: System Overview and Access**

* How to access the Policy Chat Assistant
* Understanding the user interface and sidebar controls
* Overview of available document categories
* Basic chat interaction principles

**Session 2: Effective Querying Techniques**

* How to ask clear, specific questions
* Using category filters to narrow searches
* Understanding when to rephrase questions for better results
* Interpreting response quality and completeness

**Session 3: Source Verification and Best Practices**

* How to access original source documents
* Understanding system limitations and when to seek human verification
* Appropriate use cases vs. when to consult legal professionals directly

### **11.4.2 Technical User Training**

**Power User Features**:

* Understanding different AI model options and when to use each
* Interpreting source chunk information for context verification
* Using chat history effectively for follow-up questions
* Troubleshooting common issues and error messages

## **11.5 Training Materials Needed**

**User Documentation**

* Quick start guides with screenshots
* FAQ document addressing common questions
* Video tutorials for basic operations
* Reference guide for advanced features

**Training Support**

**Monthly Check-ins**: Brief sessions to address user questions and share best practices

**Quarterly Updates**: Training on new features or system improvements

**Annual Review**: Comprehensive review of usage patterns and advanced training for power users

# **12. Conclusion**

The Policy Chat Assistant provides a powerful framework for document querying and information retrieval. While the system handles standard policy documents effectively, users should be aware of its limitations, particularly when dealing with complex document structures or legal content requiring precise interpretation.

The key to successful implementation lies in proper document preparation, understanding system limitations, and maintaining appropriate human oversight, especially for critical business and legal decisions.

Regular monitoring and continuous improvement of the document processing pipeline will ensure optimal performance as the system evolves and organizational needs change.