Enterprise Policy Assistant: A Retrieval-Augmented Generation System for Semantic Enterprise Policy Retrieval

Guttula Satish[1], Garaga Sri Pushpa Latha[2], Nandyala Himasri Chakravani[3],
Bokka Hema Siri Chandana[4], Budharouthula Keerthi Sri[5], Dr. Yalla Venkat[6]
[1]*Associate Professor, Department of Artificial Intelligence and Machine Learning, Srinivasa Institute of Engineering and Technology*
[2,3,4,5]*UG Scholar, Department of Artificial Intelligence and Machine Learning, Srinivasa Institute of Engineering and Technology*
[6]*Professor, Department of Artificial Intelligence and Machine Learning, Srinivasa Institute of Engineering and Technology*

***Abstract*—Enterprise organisations manage extensive internal policy documentation spanning Human Resources, Information Technology, Legal, and Compliance domains. Retrieving accurate information from these documents remains a time-consuming and error-prone task when relying on traditional keyword-based search systems. Furthermore, standalone AI tools often lack enterprise security, auditability, and architectural rigour. This project presents an Enterprise Policy Assistant built using Retrieval-Augmented Generation (RAG) and vector similarity search to deliver accurate, context-grounded responses from enterprise policy documents. The system employs an N-Tier enterprise architecture comprising a React.js with TypeScript frontend, a FastAPI-based REST backend, a PostgreSQL relational database, and a Qdrant vector database for semantic retrieval. OpenAI Large Language Models are invoked exclusively after relevant policy sections are retrieved, significantly reducing hallucination risks. Security is enforced using JWT-based authentication and Role-Based Access Control (RBAC). The infrastructure runs using rootless Podman containerisation, ensuring an enhanced security posture with zero privilege escalation risks. The RAG pipeline ingests policy documents (PDF, DOCX, TXT), splits them into overlapping text chunks (1300 characters, 150 overlap), generates 1536-dimensional vector embeddings using OpenAI text-embedding-3-small, and stores them in a Qdrant collection. On each user query, the system performs cosine similarity search (Top-K 10, threshold 0.4), constructs a strict retrieval-grounded prompt, and generates a source-cited response using the gpt-5.4-nano model. All query–response pairs, source documents, retrieval scores, and token usage persisted in PostgreSQL audit logs. Evaluation demonstrates API latency under 200ms (excluding LLM generation), retrieval relevance exceeding 90%, and hallucination rates below 5%. The system demonstrates how modern AI techniques can be responsibly integrated into enterprise-grade software while maintaining architectural rigour, compliance, and scalability.**

***Index Terms*—FastAPI, OpenAI, Podman, Qdrant, PostgreSQL, REST, Compliance domains, Enterprise Policy, Evaluation.**

I. INTRODUCTION

Enterprise Policy Assistant is an enterprise-safe, AI-based QA system allowing users to submit natural language questions about enterprise policy documentation and receive trustworthy, citation-based answers. It is based on the RAG architecture, combining semantic vector search with controlled LLM generation to ensure reliability and source grounding.

Unlike regular chatbots using pre-existing knowledge, the described solution is fully dependent on enterprise documentation, including HR policies, IT guidelines, and other types of documents. It generates answers solely on retrieved documents and provides sources in answers.

The system is developed as a full-stack web application and consists of React.js frontend, FastAPI

backend, Qdrant vector database used for search purposes, and PostgreSQL storing authentication data and audit logs. Containerized deployment uses rootless containers for safe execution.

Organizations store a vast number of policies in their knowledge base. Nevertheless, there is still an issue of accessing relevant information efficiently due to inconsistent storage, mismatch between the query and the semantics of the documents, and overall inefficiencies. Conventional keyword-based search systems are not capable of capturing user intent, which leads to inaccurate search results. Moreover, employees tend to ask their questions to HR or other managers in case of any doubts regarding the company's policies, which burdens the management team.

With recent developments in language models and embedding techniques, semantic searches are now possible. However, unrestricted AI systems can generate hallucinations and compromise sensitive data. Thus, one should consider adopting a retrieval-based approach, where information retrieval comes first, followed by the generation of a response based on retrieved sources.

Businesses encounter several problems when searching for information regarding their policies:

Fragmented Information: Policies are scattered across various mediums without a unified interface. Inefficient Search: Keyword-based search engines cannot infer the meaning behind a query. Increased Operational Cost: Repeated requests for policies burden HR and management departments. Security Threats: General-purpose AI systems lack enterprise-grade security measures. LLM Hallucinations: Large Language Models may hallucinate and output false information.

Inadequate Logging: No traceable record of interactions within the system exists. This project aims to address these problems by adopting the RAG paradigm with secure authentication and auditing capabilities.

II. SYSTEM ARCHITECTURE

The structural foundation of the Enterprise Policy Assistant is engineered to address the critical balance between high-performance data retrieval and stringent enterprise security. At its core, the system utilizes a modular, N-Tier design philosophy that ensures a complete decoupling of the user-facing interface from the underlying retrieval logic and data persistence engines. This architectural separation is vital for maintaining system integrity, as it allows for independent scaling of the processing backend while ensuring that the data storage remains isolated from direct external access.

Central to this design is the orchestration of a Retrieval-Augmented Generation (RAG) workflow that operates as a closed-loop system, where every component is purpose-built to facilitate secure, low-latency communication. The integration between the semantic search engine and the generative model is governed by a strict orchestration layer that validates and sanitizes all data flow, ensuring that the Large Language Model (LLM) acts solely as a synthesis tool for verified, retrieved information. Furthermore, the entire stack is deployed within a containerized environment that prioritizes a zero-privilege security posture, utilizing rootless execution to mitigate potential infrastructure vulnerabilities. This cohesive architectural framework not only supports high-concurrency natural language queries but also provides a transparent, auditable pathway for every piece of information processed by the system.

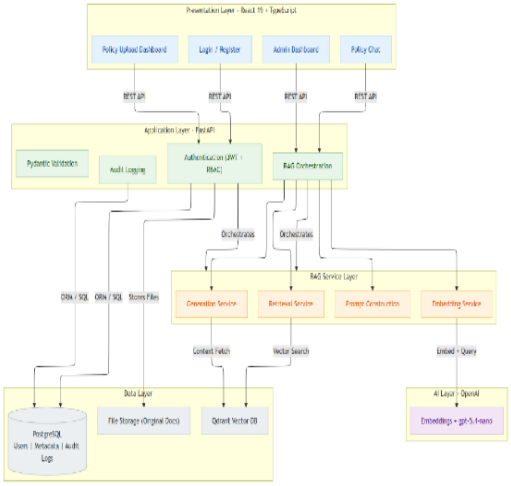

Figure 1 The high-level system architecture.

The frontend communicates with the backend exclusively via RESTful HTTP/HTTPS API calls. A React Context provides JWT token management, route protection via Private Route and Adm the Enterprise Policy Assistant is designed according to a five-layer N-Tier architecture that enforces a strict separation of concerns between each functional domain. This

architectural pattern ensures that each layer has a single, well-defined responsibility and communicates with adjacent layers only through defined interfaces.

Layer 2: Application Layer

The Application Layer is implemented as a FastAPI Python application running on a Uvicorn ASGI server. It is responsible for request routing, JWT authentication and authorisation enforcement (via FastAPI dependencies), input validation using Pydantic models, CORS policy enforcement, and routing to the appropriate service layer.

Four router modules handle different functional domains: auth.py (user registration and login), policies.py (document ingestion, search, and query), admin.py (user management and audit log access), and health.py (system health checks).

Layer 3: RAG Service Layer

The RAG Service Layer is the core intelligence of the system. It encompasses five dedicated service modules: the Document Parser (extract_text_from_upload), the Chunker (chunk_text with configurable size and overlap), the Embedding Service (embed texts calling OpenAI text-embedding-3-small), the Retrieval Service (search chunks via Qdrant cosine similarity), and the RAG Query Service (query policy which orchestrates retrieval, prompt construction, LLM generation, and audit logging). Optional components include the Query Rewrite Service and Re-ranking Service, configurable via environment variables.

Layer 4: Data Layer

The Data Layer consists of two complementary storage systems with distinct roles. PostgreSQL (v16) stores structured relational data: user accounts (users table), policy document metadata (policy metadata table), and query audit logs (audit logs table). Qdrant stores vector embeddings of policy document chunks. Each vector point carries a payload containing the chunk text, document ID, source filename, and chunk index, enabling the retrieval service to return full chunk content alongside similarity scores.

Layer 5: AI Layer

The AI Layer interfaces with OpenAI's API services. The Embeddings API (text-embedding-3-small model, 1536 dimensions) is called during both document ingestion (to embed chunks) and query processing (to embed the user query). The Chat Completions API (gpt-5.4-nano model, temperature 0.2) is called only after relevant chunks have been retrieved, with a server-enforced system prompt that restricts generation to retrieved context.

The data flow during a policy query proceeds as follows:

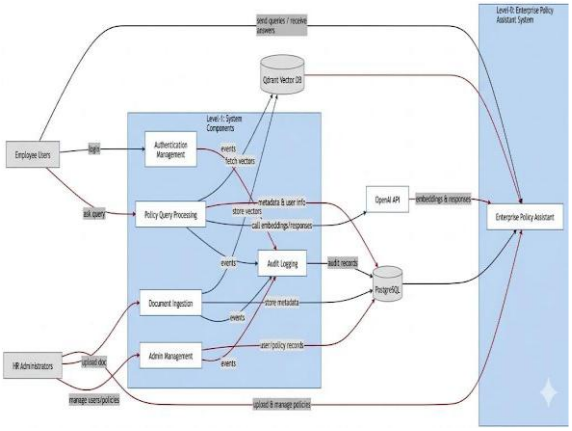

Figure 2 Data flow diagram of the system

III. METHODOLOGY

The implementation of the Enterprise Policy Assistant follows a structured and modular approach. Initially, a containerized infrastructure was established using Podman to deploy PostgreSQL for relational data storage and Qdrant for vector-based retrieval. A secure authentication mechanism was implemented using JSON Web Tokens (JWT) along with role-based access control to regulate user permissions.

Subsequently, a document ingestion pipeline was developed to process enterprise documents through parsing, text chunking, embedding generation, and storage in the vector database. The core Retrieval-Augmented Generation (RAG) pipeline was then implemented to handle user queries by combining semantic retrieval with language model-based response generation. A React-based frontend interface was developed to facilitate user interaction, and the entire system was validated through systematic testing of individual and integrated components.

The system operates through a sequential processing pipeline. Initially, policy documents are uploaded and their textual content is extracted using appropriate parsers. The extracted text is divided into smaller, overlapping chunks to preserve contextual continuity.

Each chunk is then converted into a vector representation using an embedding model and stored in the Qdrant vector database.

When a user submits a query, it is transformed into an embedding and compared against stored vectors using cosine similarity to retrieve the most relevant chunks. These retrieved segments are then provided as contextual input to the language model, which generates a response grounded in the retrieved data. Finally, each query and its corresponding response are logged for tracking and analysis.

The system components are integrated through well-defined interfaces. The frontend communicates with the backend using RESTful APIs to handle user interactions and data exchange. The backend processes requests and interacts with PostgreSQL for structured data storage and Qdrant for vector retrieval operations. Additionally, the OpenAI API is utilized for generating embeddings and producing context-aware responses.

The system is deployed using a containerized approach to ensure consistency and portability across environments. Podman is used to manage the deployment of database and vector services, while application components are configured using environment variables to maintain flexibility and security.

IV. RESULTS

The system delivers a seamless and user-friendly experience through two primary interfaces:

1. User Chat Interface:

A clean and intuitive conversational interface that enables employees to interact with the system effortlessly. Users can ask queries in natural language and receive accurate responses, along with highlighted source citations to ensure transparency and reliability of information.

2. Admin Dashboard:

A centralized management portal designed for administrators to efficiently handle system operations. It allows document uploads, monitors processing status in real time, and provides access to audit logs for tracking system activity and ensuring compliance. The following interface screenshots are referenced from the project documentation:

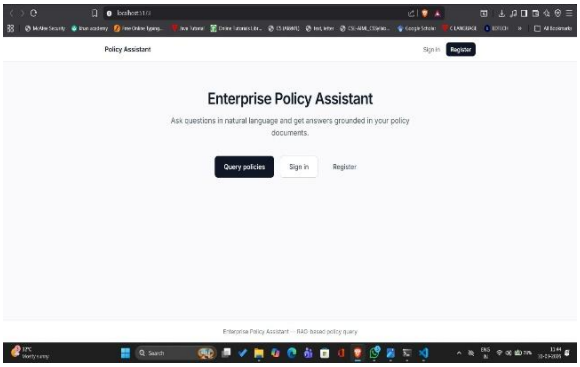

Figure 3 Welcome Page

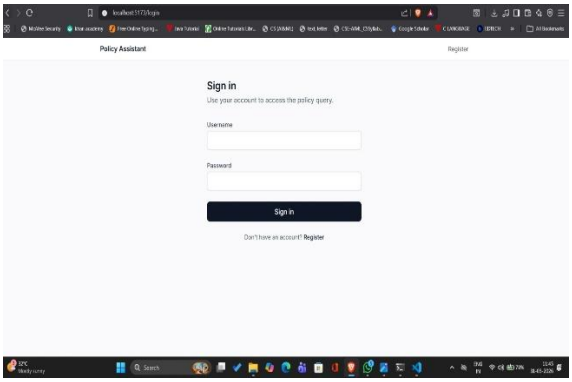

Figure 4 Sign in Page

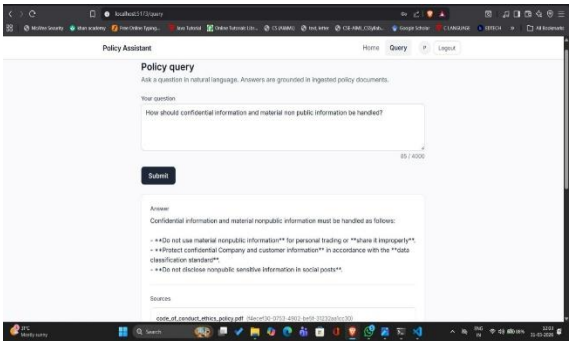

Figure 5 User chat interface

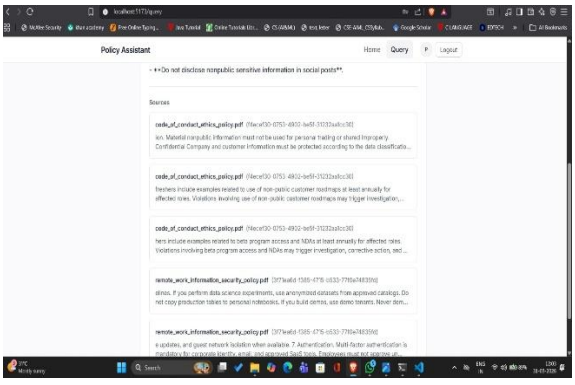

Figure 6 Answer of query with source

V. CONCLUSION

The Enterprise Policy Assistant successfully bridges the gap between complex corporate documentation and user-friendly AI interaction. By employing a RAG-based architecture, the system provides a reliable, secure, and highly accurate alternative to traditional search methods. The integration of rootless containerization and Role-Based Access Control ensures the platform is ready for the security demands of modern enterprise environments. Furthermore, the implementation of comprehensive audit logging provides a transparent trail for compliance monitoring and quality assurance. This architectural framework serves as a scalable foundation for deploying trustworthy generative AI across diverse corporate environments. Future enhancements will focus on expanding the system to support multi-turn conversations and integrating with existing enterprise identity providers like SAML or OIDC.

VI. FUTURE WORK

The following enhancements are identified as high-priority for subsequent development. Hybrid Search combining dense vector retrieval with sparse BM25 scoring will improve recall for queries containing specific terminology or document ref-erence numbers. Local LLM Integration using Ollama with open-source models (e.g., Llama, Mistral) will enable air-gapped and cost-controlled deployment without OpenAI de-pendency. OCR Integration using Tesseract or a cloud OCR service will support ingestion of scanned documents. Per-Department Collection Isolation will enable RBAC at the vector storage level. Multi-Turn Dialogue will add session-based conversation history to the RAG pipeline, supporting coherent follow-up questions. Context Compression will selectively extract the most relevant sentences from retrieved chunks before LLM generation, reducing token consumption. Kubernetes Deployment manifests will support production-grade container orchestration with auto-scaling. Multilingual Support will extend the system to policy documents in regional languages via multilingual embedding models and language-adaptive prompting.

REFERENCES

[1] P. Lewis et al., “Retrieval-augmented generation for knowledge-intensive NLP tasks,” in Proc. Adv. Neural Inf. Process. Syst. (NeurIPS), vol. 33, pp. 9459–9474, 2020.

[2] G. Izacard and E. Grave, “Leveraging passage retrieval with generative models for open domain question answering,” in Proc. 16th Conf. Eur. Chapter Assoc. Comput. Linguist. (EACL), pp. 874–880, 2021.

[3] N. Reimers and I. Gurevych, “Sentence-BERT: Sentence embeddings using Siamese BERT-networks,” in Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP), 2019.

[4] J. Johnson, M. Douze, and H. Jégou, “Billion-scale similarity search with GPUs,” IEEE Trans. Big Data, vol. 7, no. 3, pp. 535–547, Jul. 2021.

[5] S. Maynez, S. Narayan, B. Bohnet, and R. McDonald, “On faithfulness and factuality in abstractive summarization,” in Proc. 58th Annu. Meeting Assoc. Comput. Linguist. (ACL), pp. 1906–1919, 2020.

[6] M. Jones, J. Bradley, and N. Sakimura, “JSON Web Token (JWT),” IETF RFC 7519, May 2015. [Online]. Available: https://www.rfc-editor.org/rfc/rfc7519

[7] D. Ferraiolo and D. R. Kuhn, “Role-based access controls,” in Proc. 15th Nat. Comput. Secur. Conf., pp. 554–563, 1992.

[8] OpenAI, “OpenAI API reference,” 2024. [Online]. Available: https://platform.openai.com/docs/api-reference

[9] React Team, “React: A JavaScript library for building user interfaces,” 2024. [Online]. Available: https://react.dev

[10] PostgreSQL Global Development Group, “PostgreSQL 16 documentation,” 2024. [Online]. Available: https://www.postgresql.org/docs/16/

[11] Red Hat, Inc., “Podman: A tool for managing OCI containers and pods,” 2024. [Online]. Available: https://podman.io/docs

[12] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, “Distributed representations of words and phrases and their compositionality,” in Proc. Adv. Neural Inf. Process. Syst. (NIPS), vol. 26, 2013.

[13] S. Robertson and H. Zaragoza, "The probabilistic relevance framework: BM25 and beyond," Found. Trends Inf. Retr., vol. 3, no. 4, pp. 333–389, 2009.