AI Customer Support Agent

Technical Report

A document-grounded customer-support application using Streamlit, LangChain, RAG, Ollama, Chroma, SQLite, and controlled support tools.

Project Status: Working MVP completed and validated

Abstract

This project implements an AI-powered customer support agent that combines retrieval-augmented generation with controlled operational tools. The application provides a conversational Streamlit interface through which customers can ask policy and knowledge-base questions, retrieve order information, create and inspect support tickets, and request escalation to a human agent. Knowledge questions use a document retrieval pipeline based on PyPDF, text chunking, embeddings, and Chroma retrieval. The retrieved context is supplied to a local Ollama language model, llama3.1:8b, to produce grounded responses. Operational requests are handled through predefined tools backed by SQLite rather than relying on the language model to invent or modify operational data.

The completed MVP was validated using nine representative scenarios covering policy reasoning, order lookup, ticket workflows, human escalation, and an unsupported knowledge question. All nine scenarios passed. A key model-selection experiment also showed that llama3.1:8b handled a simple return-window reasoning case more reliably than the previously tested llama3.2:3b model.

Keywords: Customer Support, RAG, Retrieval-Augmented Generation, LangChain, Ollama, Chroma, Streamlit, SQLite, Tool-based Agent

1. Introduction

Customer-support applications frequently need to answer repetitive policy questions while also performing operational tasks such as checking orders and creating support tickets. A useful AI support system therefore needs both language understanding and reliable access to organization-specific information and actions.

The AI Customer Support Agent is designed as a generic customer-support application rather than as a system tied to a single visible company brand. Its knowledge layer can be adapted to organization-specific support documents, while its operational tools provide controlled access to order and ticket information.

1.1 Problem Statement

A conversational assistant that relies only on a language model may produce plausible but unsupported answers to company-policy questions. It may also be unsuitable for operational actions if it has no controlled connection to order and ticket data. The project addresses this by separating knowledge retrieval and operational tool execution from natural-language generation.

1.2 Objectives

• Provide a simple customer-facing conversational support interface.

• Answer organization-specific questions using retrieved knowledge rather than unsupported generation.

• Support order lookup through a controlled application tool.

• Create and retrieve support tickets through controlled tools backed by SQLite.

• Support human-agent escalation.

• Handle insufficient knowledge explicitly instead of hallucinating company policy.

• Run locally using Ollama for language-model inference.

1.3 Scope

The completed scope is a local working MVP. It demonstrates end-to-end customer-support workflows, including knowledge-grounded responses and operational support actions. Production-scale deployment, authentication, advanced monitoring, and cloud infrastructure are outside the current MVP scope.

2. System Architecture

The architecture separates the customer interface, agent decision layer, retrieval pipeline, language model, support tools, and operational storage. The supplied architecture diagram illustrates the complete flow on a single page.

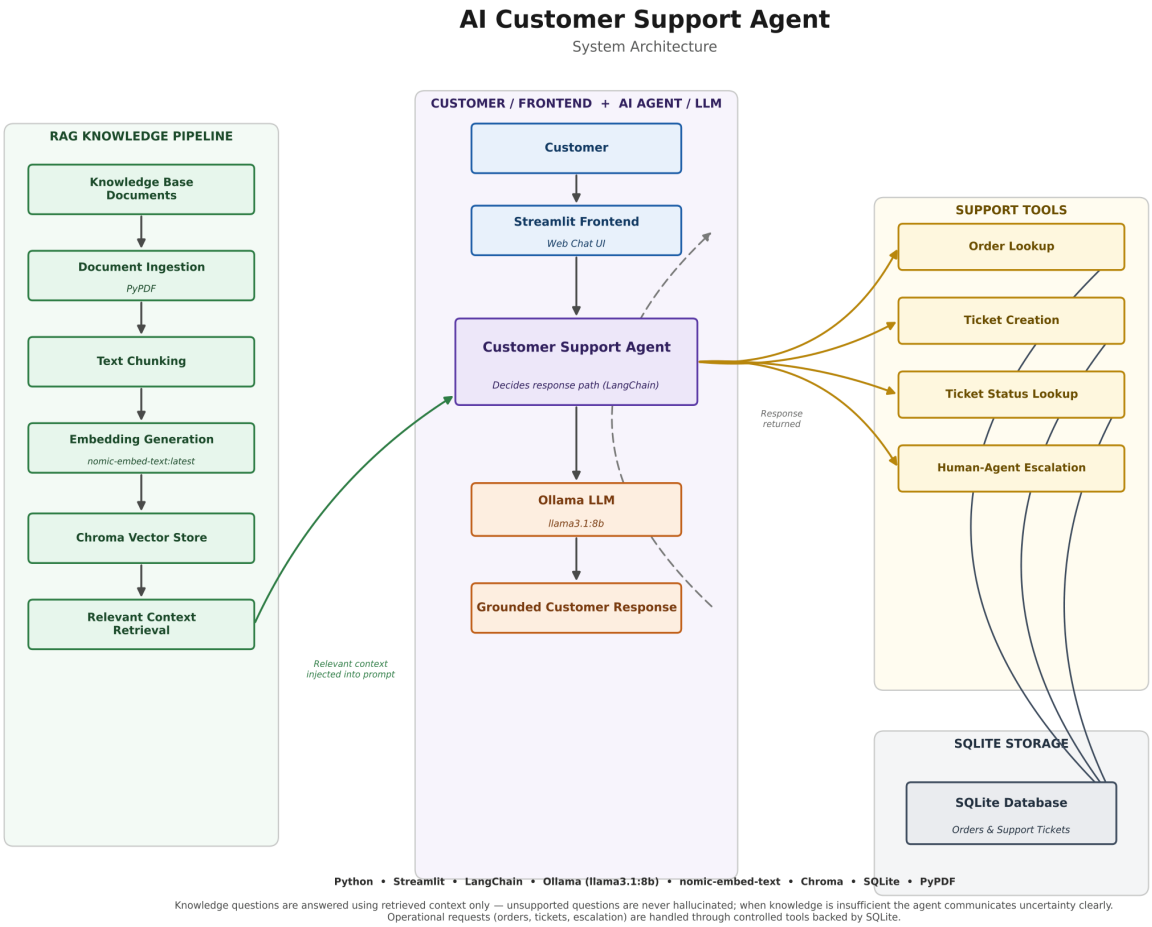

Figure 1. AI Customer Support Agent system architecture. The diagram shows the RAG knowledge pipeline on the left, the customer/frontend and agent/LLM path in the center, and controlled support tools with SQLite storage on the right.

2.1 Main Components

• **Customer / Streamlit Frontend:** receives customer messages and displays responses through the web chat UI.

• **Customer Support Agent:** determines the appropriate response path using the application logic and LangChain.

• **RAG Knowledge Pipeline:** processes documents, creates embeddings, stores them in Chroma, and retrieves relevant context.

• **Ollama LLM:** llama3.1:8b generates the final grounded natural-language response.

• **Support Tools:** order lookup, ticket creation, ticket status lookup, and human-agent escalation.

• **SQLite:** stores operational information such as orders and support tickets.

2.2 Response Paths

Knowledge questions follow the retrieval path: knowledge-base documents are ingested, chunked, embedded, stored in Chroma, and searched for relevant context. That context is injected into the agent/LLM prompt before the customer-facing answer is generated. Operational requests follow a tool path: the agent invokes a controlled support tool, obtains the operational result from SQLite, and returns the result to the customer.

3. Knowledge Retrieval and RAG Pipeline

The knowledge layer is document-driven. The architecture diagram identifies Knowledge Base Documents, PyPDF ingestion, text chunking, embedding generation using nomic-embed-text:latest, Chroma vector storage, and relevant-context retrieval as the main stages.

3.1 Document Ingestion

Knowledge-base documents are processed through the ingestion service. PyPDF is used for PDF document processing. The extracted text becomes the input to the chunking stage.

3.2 Text Chunking

Long document text is divided into smaller sections so that relevant portions can be retrieved for an individual customer question. Chunking supports targeted retrieval rather than supplying an entire document to the language model for every request.

3.3 Embeddings and Vector Store

The embedding model configured for the working application is nomic-embed-text:latest. The resulting representations are stored in a Chroma vector store. At query time, the customer's knowledge question is used to retrieve relevant context from this store.

3.4 Grounded Generation

The retrieved context is injected into the agent's prompt and passed to the Ollama language model. The purpose is to constrain knowledge-based answers to the available support information. When the knowledge base does not contain enough information, the system communicates uncertainty rather than inventing a company-specific answer.

3.5 Example

For the question “What is the return policy?”, the working system retrieved the relevant return-policy information and returned the 30-calendar-day return window. For “Can I return an item within 10 days?”, the final model correctly recognized that 10 days falls within the 30-day window and also communicated the applicable eligibility conditions.

4. Customer Support Agent and Tools

The customer support agent is the central application component. It receives the customer's message and routes the request either toward knowledge retrieval and language generation or toward a controlled operational tool.

4.1 Order Lookup

The order lookup tool retrieves order information using an order identifier. Validation included both a shipped order (NC1002) and a delivered order (NC1001). The returned information included item, amount, status, and delivery date when available.

4.2 Ticket Creation

Ticket creation is used for issues that require investigation or support intervention. A damaged-order request for NC1001 successfully created a support ticket with an automatically increasing ticket ID and Open status.

4.3 Ticket Status Lookup

The ticket lookup path retrieves an existing ticket using its identifier. A validation query for TKT0018 returned its Open status, issue description, order ID, and creation timestamp.

4.4 Human Escalation

When a customer explicitly requests human assistance, the escalation tool creates an escalation ticket. The tested workflow returned a ticket ID and Open status so that the request can be reviewed by support.

4.5 Separation of Generation and Actions

A key design choice is that the language model is not treated as the source of truth for operational records. Order and ticket information is obtained through predefined tools and SQLite storage. The LLM is responsible for interpreting the request and presenting the resulting information naturally.

5. Technology Stack and Implementation

Component	Technology / Configuration
Frontend	Streamlit
Language	Python
Agent / orchestration	LangChain + application agent logic
Chat model	Ollama – llama3.1:8b
Embedding model	nomic-embed-text:latest
Retrieval store	Chroma
Document processing	PyPDF
Operational database	SQLite
Testing	Pytest / manual functional validation

5.1 Project Structure

`app/main.py` provides the Streamlit application entry point. `app/agent/customer_support_agent.py` contains the central support-agent logic. The services package contains document ingestion, chunking, RAG pipeline, and vector-store components. The tools package contains order lookup, ticket, and escalation functionality. The database package contains SQLite database functionality.

5.2 Local Execution

The working application is run from the project root using the configured Python environment and the Streamlit entry point. The project uses the environment path configuration required for the app package to resolve correctly.

5.3 Model Selection Experiment

Two local chat models were tested during development. llama3.2:3b produced an incorrect interpretation for a simple policy-timeframe question, stating that a 10-day return was outside a 30-day window. The application was switched to llama3.1:8b, which correctly interpreted the same question and preserved the expected policy reasoning. The 8B model is therefore the final configured chat model.

6. Testing and Validation

The completed MVP was validated against nine representative customer-support scenarios. The tests were selected to cover both knowledge-grounded behavior and controlled operational actions.

ID	Scenario	Result
TC-01	Return policy	PASS
TC-02	Return within 10 days	PASS
TC-03	Non-returnable items	PASS
TC-04	Order NC1002 lookup	PASS
TC-05	Order NC1001 lookup	PASS
TC-06	Damaged-order ticket creation	PASS
TC-07	Ticket status lookup	PASS
TC-08	Human escalation	PASS
TC-09	Unsupported Antarctica delivery question	PASS

6.1 Representative Results

NC1002: Smart Watch, ■3999.00, status Shipped.

NC1001: Wireless Headphones, ■2499.00, status Delivered, delivery date 2026-08-10.

TKT0018: status Open, issue associated with damaged NC1001, order ID NC1001.

6.2 Uncertainty / Edge-Case Validation

The question “Does the company deliver to Antarctica?” was intentionally used as an unsupported knowledge case. The system did not invent a specific delivery policy and instead stated that the available knowledge base did not provide enough information to answer specifically about Antarctica.

6.3 Overall Result

All nine validation scenarios passed, giving a 100% pass rate for the tested MVP scenarios. This result reflects the defined validation set and should not be interpreted as proof of production-scale reliability across all possible customer-support conversations.

7. Engineering Decisions and Reliability

The implementation prioritizes a reliable core workflow over unnecessary feature expansion. Knowledge retrieval and operational actions are separated so that the language model can focus on language understanding and response generation while predefined tools handle operational records.

The use of a local Ollama deployment also keeps the MVP self-contained. The selected llama3.1:8b model was retained after testing showed improved policy reasoning over the smaller llama3.2:3b model.

8. Limitations

• The application operates on the configured knowledge base and local operational data.

• It cannot reliably answer company-specific questions that are absent from the available knowledge.

• The current MVP is designed for local demonstration and evaluation rather than production-scale deployment.

• The system does not currently provide a full administrative knowledge-base management interface.

• Advanced authentication, authorization, observability, and cloud-scale infrastructure are outside the current scope.

9. Future Improvements

• Add an administrator workflow for updating and re-indexing knowledge documents.

• Add authentication and role-based access control.

• Expand automated evaluation with a larger, versioned customer-support test set.

• Add monitoring, tracing, and production observability.

• Integrate a production database and deployment environment.

• Add more controlled customer-support tools and stronger tool-permission policies.

10. Conclusion

The AI Customer Support Agent demonstrates a complete working customer-support MVP that combines document-grounded RAG with controlled operational tools. The system can answer policy questions from retrieved context, retrieve order information, create and inspect tickets, escalate to human support, and communicate uncertainty when the knowledge base is insufficient. The nine-scenario validation set passed in full, and the final llama3.1:8b configuration improved the model's policy reasoning during testing.

The architecture and implementation provide a foundation that can be adapted to different organizations by replacing or updating the support knowledge base and operational data while preserving the core customer-support workflow.