JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue
JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)
An International Scholarly Open Access, Peer-reviewed, Refereed Journal

# DESIGN AND IMPLEMENTATION OF A MULTI-AGENT AI CUSTOMER SUPPORT SYSTEM USING DJANGO, PYTHON, RAG, CLAUDE API, AND LANGCHAIN

**[1]Omkar Balkrishna Shinde, [2]Nikita Ravindra Fatangare**

[1,2]Department of MCA (Master of Computer Applications), Year II, Semester III
[1,2]Navsahyadri Group of Institutes (NGI), Pune, Maharashtra, India

*Abstract :* As artificial intelligence continues to advance, traditional customer support systems face challenges in handling large volumes of customer queries, retrieving accurate business information, and making consistent decisions. This research proposes a Multi-Agent AI Customer Support System for a fictional air-conditioning company named CoolBreeze AC. The system integrates multiple specialized AI agents to automate customer interactions, verify order and delivery information, assess potential fraud risks, and manage refund-related escalations.

The proposed system utilizes Python and Django for backend development, the Anthropic Claude API for language understanding and decision-making, Retrieval-Augmented Generation (RAG) for accessing company policies, and ChromaDB for vector-based document retrieval. Additionally, LangChain is used to implement agent orchestration, while Server-Sent Events (SSE) provide real-time updates through a staff monitoring dashboard.

The system consists of three primary agents: Maya, a customer support agent; a manager agent, responsible for reviewing escalated cases; and a risk agent, responsible for assessing predefined fraud indicators. By combining specialized agents with verified business data and policy-based retrieval, the proposed system aims to improve support consistency, transparency, and operational efficiency. The project initially implements the agent loop without an orchestration framework and subsequently recreates the workflow using LangChain to compare both approaches.

***Index Terms*** - Multi-Agent Systems; Agentic Artificial Intelligence; Customer Support Automation; Django; Python; Claude API; LangChain; Retrieval-Augmented Generation; ChromaDB; Embeddings; Server-Sent Events; Fraud Risk Assessment; Real-Time Monitoring

## I. INTRODUCTION

### *A. Background*

Customer support plays an important role in modern businesses. Organizations regularly receive inquiries related to product information, order status, delivery delays, warranty policies, refunds, and technical issues. Handling these requests manually can increase operational costs, delay responses, and create inconsistencies in customer service.

Traditional customer support systems often depend on human agents to retrieve information from databases, examine company policies, and determine whether a request requires escalation. Although conventional chatbots can automate basic conversations, they may struggle with complex requests, changing business information, and decisions that require multiple steps.

Recent developments in Large Language Models (LLMs) have introduced new opportunities for automating customer support. These models can understand natural-language questions, generate contextual responses, and request external tools to complete specific tasks. However, language models may generate unsupported information when they lack access to verified business records or relevant documentation.

Agentic Artificial Intelligence provides an approach in which AI agents can analyze requests, select tools, examine results, and determine subsequent actions. Multi-agent systems extend this approach by distributing responsibilities among specialized agents.

### *B. Problem Statement*

Manual and chatbot-only support workflows struggle to retrieve accurate business information, apply company policies consistently, and decide when a request needs escalation. Language models used without verified records or documentation may produce unsupported answers, which is a particular risk for order, delivery, and refund queries.

*C. Objectives*

- To design a multi-agent customer support system for CoolBreeze AC in which Maya, a manager agent, and a risk agent have separate responsibilities.
- To ground agent responses in verified business data from Django and in company policy documents retrieved through RAG.
- To provide staff with a live monitoring dashboard using Server-Sent Events (SSE).
- To implement the agent loop without a framework first, and then recreate the workflow using LangChain to compare both approaches.
- To keep financial actions under controlled tool access, deterministic validation, and human oversight.

*D. Research Questions*

1. How can specialized AI agents be organized to handle routine inquiries, risk assessment, and escalated refund review in a customer support workflow?
2. How can RAG and approved backend tools reduce unsupported responses in a customer support setting?
3. How does a framework-free agent loop compare with a LangChain-based implementation of the same workflow?

*E. Importance and Contribution of the Study*

This research proposes a multi-agent customer support system for CoolBreeze AC. The system uses Django to manage application logic and business data, the Claude API to process customer queries, and Retrieval-Augmented Generation to retrieve information from company documents. Maya handles routine customer inquiries and checks order information through approved tools. When a request requires additional review, the case is transferred to a manager agent. A risk agent evaluates predefined indicators and provides an assessment to support the manager's decision.

Furthermore, a live streaming dashboard allows staff to monitor important agent activities, tool executions, and case updates in real time. The system is designed to demonstrate how AI agents can be integrated into a practical web application while maintaining security, controlled tool access, and appropriate human oversight.

## II. Literature Review

*A. Overview*

Artificial intelligence has developed from traditional rule-based systems to advanced language models capable of understanding natural-language instructions and interacting with external tools. Several technologies are relevant to the development of an intelligent customer support system.

Large Language Models provide natural-language understanding and generation capabilities. Agentic AI extends these capabilities by allowing a model to request tools and participate in multi-step workflows. Retrieval-Augmented Generation connects language models with external knowledge sources, while vector databases enable semantic retrieval of relevant information.

Django provides the backend infrastructure required to manage business data, authentication, and application logic. LangChain offers reusable components for connecting language models with tools and retrieval systems. Server-Sent Events enables continuous server-to-browser communication for live monitoring.

*B. Large Language Models*

Large Language Models are machine-learning systems trained to process and generate human language. They can interpret questions, summarize information, generate responses, and select available tools when supported by the application.

In the proposed system, the Anthropic Claude API provides the language model interface. The model processes customer messages and assists agents in selecting suitable actions.

However, an LLM does not automatically have access to current order records or private company policies. The application must retrieve verified information and provide it as context. Model-generated outputs should also be validated before they are used for business decisions.

*C. Agentic Artificial Intelligence*

Agentic AI refers to systems that use AI models to select actions and work toward defined objectives. An agent can receive a request, identify the required operation, invoke an approved tool, evaluate its result, and continue until the task is completed or a stopping condition is reached.

In the CoolBreeze AC system, Maya uses this process to answer customer questions and verify delivery information. When a case requires managerial review, the orchestration layer transfers the relevant information to specialized agents. The use of an agent loop enables the system to perform structured tasks rather than relying exclusively on a single generated response.

*D. Multi-Agent Systems*

Multi-agent systems divide complex workflows among multiple agents with specialized responsibilities. Each agent operates within a defined role and communicates with other agents through structured messages or handoffs. The proposed system contains three agents:

- **Maya:** responsible for customer support.
- **Manager agent:** responsible for reviewing escalated cases.
- **Risk agent:** responsible for assessing predefined fraud indicators.

This separation of responsibilities supports modular development and controlled decision-making. However, multi-agent systems may introduce additional model calls, processing time, and coordination complexity. Therefore, agent handoffs should be used only when necessary.

*E. Retrieval-Augmented Generation*

Retrieval-Augmented Generation combines information retrieval with language generation. It retrieves relevant passages from external documents and supplies them to a language model before the model generates an answer.

In this project, company documents such as refund policies, warranty guidelines, delivery rules, and product manuals are processed and indexed for retrieval. The system uses pypdf to extract text from PDF documents and ChromaDB to store and retrieve document embeddings. When a customer asks a policy-related question, the retrieval system searches for relevant passages and provides them to the appropriate agent.

RAG can improve the grounding of responses, although its effectiveness depends on document quality, retrieval accuracy, and the relevance of the retrieved information.

*F. LangChain Framework*

LangChain provides abstractions for integrating language models, tools, prompts, and retrieval components. It can simplify the development of agent-based applications by providing reusable building blocks.

The proposed project first implements the agent loop without a framework. The same workflow is subsequently recreated using LangChain. This approach helps developers understand the underlying mechanisms of tool calling and agent orchestration before using higher-level abstractions.

*G. Review of Related Studies*

Five related studies were reviewed. Barrionuevo-Valenzuela et al. [1] developed an Agentic-RAG copilot for human operators and reported better response quality, relevance, and accuracy than naïve RAG. Tiwari et al. [2] presented ShopEase, a multi-agent framework with hybrid RAG, and compared FAISS, BM25, and rank-fusion retrieval strategies. Jiang et al. [3] proposed ChatMap, which uses multi-agent collaboration to mine service strategies from historical dialogues, and reported Taobao A/B test improvements. Fuchs et al. [4] studied chunking, contextualization, and query refinement in a production RAG assistant at eBay. Pandya and Holia [5] built Sahaay, an organization-specific chatbot using LangChain and Flan-T5. A detailed comparison is given in Table 1 (Appendix A).

*H. Research Gap*

The reviewed studies [1]–[5] address different aspects of automated customer support, including multi-agent collaboration, retrieval optimization, customer-context management, historical service-strategy mining, and organizational knowledge integration. However, these capabilities represent distinct research directions rather than evidence that any single approach is universally superior.

The proposed system aims to integrate these capabilities into a unified application using Django, Python, LangChain, Claude API, Retrieval-Augmented Generation (RAG), and multiple specialized AI agents. Its intended features include real-time response streaming, customer-context retrieval, risk assessment, and escalation to human support when required.

## III. METHODOLOGY

*A. System Architecture*

The proposed system consists of several interconnected components, including a customer interface, Django backend, AI agent orchestration layer, retrieval subsystem, business database, and live monitoring dashboard.

The customer submits a request through the interface. Django validates the request and forwards it to the agent orchestration layer. Maya processes the request and uses approved tools to retrieve business information. When necessary, the manager and risk agents collaborate to review escalated cases.

The RAG subsystem retrieves relevant passages from company documents stored in a vector database. Claude API supports natural-language processing and agent decisions. The system records important workflow events and streams permitted updates to the staff dashboard through SSE.

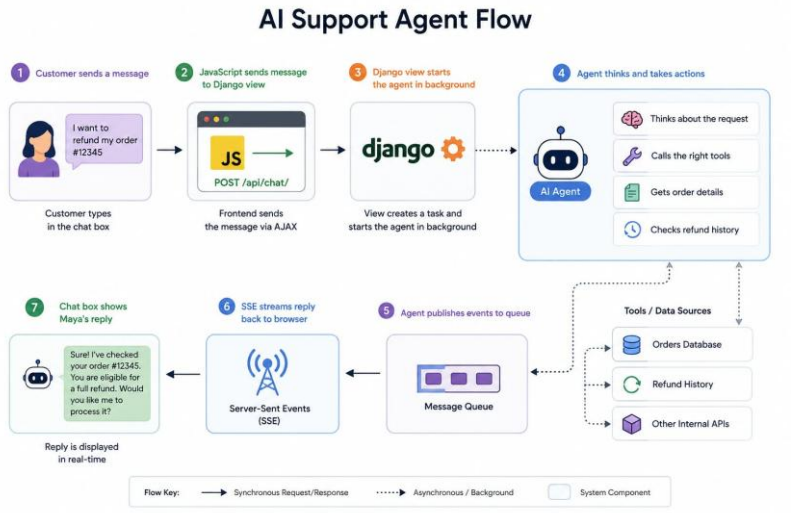



**Figure 3.1:** *AI Support Agent Flow*

*B. Dataset and Knowledge Sources*

The proposed system uses two primary categories of information.

**Business Data:** This includes customer records, order details, delivery status, support cases, and refund information stored in a relational database managed by Django.

**Company Documents:** This includes refund policies, warranty conditions, delivery guidelines, and product manuals used by the RAG subsystem.

Business records provide current operational facts, while company documents provide policy-related knowledge. These sources serve different purposes and should not be treated as interchangeable.

*C. Data Pipeline*

The data pipeline processes company documents before they are used by AI agents. It consists of the following stages:

- **Document Extraction:** Text is extracted from PDF documents using pypdf.
- **Text Chunking:** Extracted content is divided into smaller sections suitable for retrieval.
- **Embedding Generation:** Text chunks are converted into numerical vector representations.
- **Vector Storage:** Embeddings and associated metadata are stored in ChromaDB.
- **Query Processing:** Customer questions are converted into queries suitable for semantic retrieval.
- **Similarity Search:** Relevant document passages are retrieved.
- **Context Construction:** Retrieved passages are supplied to the appropriate AI agent.
- **Response Generation:** The agent generates a response based on the available evidence.

The quality of the final answer depends on document quality, retrieval relevance, and the model's ability to interpret the supplied context.

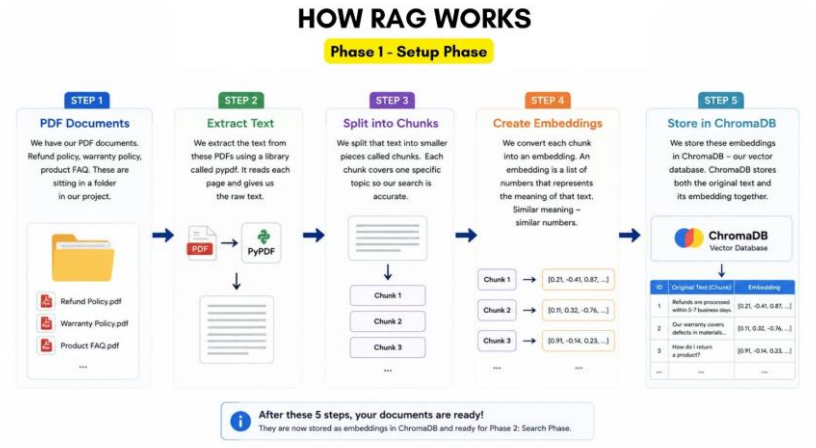



**Figure 3.2:** *RAG Workflow – Phase 1 (Setup Phase)*




**Figure 3.3:** *RAG Workflow – Phase 2 (Search Phase)*

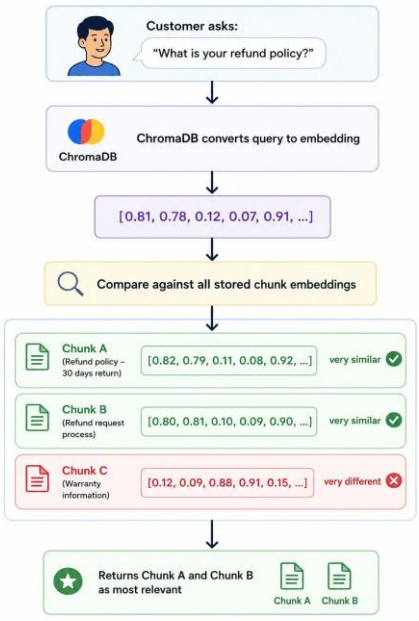



**Figure 3.4:** *Query Embedding and Similarity-Based Chunk Retrieval*

*D. Multi-Agent Customer Support System*

The core component is the multi-agent orchestration system, which coordinates specialized agents and approved backend tools.

**Maya Support Agent:** Maya identifies customer intent, retrieves relevant policies, checks order details, and responds to routine questions.

**Manager Agent:** The manager agent reviews escalated cases, considers verified order information and policy evidence, and generates refund recommendations.

**Risk Agent:** The risk agent examines predefined indicators and returns a structured risk assessment for the manager.

**Django Backend:** Django manages authentication, database access, case history, and business-rule validation.

**Claude API:** The external language model supports natural-language understanding and tool selection.

**RAG Module:** The retrieval subsystem supplies relevant company-document passages to agents.

**Monitoring Dashboard:** The dashboard displays authorized agent events and case updates through SSE.




**Figure 3.5:** *AI Agent Tools Overview*

*E. Workflow*

The proposed system operates as follows:

1. A customer submits a question through the customer support interface.
2. Django creates or retrieves the associated support case.
3. Maya identifies the customer's intent and determines the required operation.
4. If business information is needed, Maya requests an approved tool call.
5. Django validates the request and returns the authorized result.
6. If company policy information is required, the RAG subsystem retrieves relevant document passages.
7. Maya generates a response or determines that escalation is required.
8. For escalated cases, the risk agent assesses available indicators and the manager agent reviews the evidence.
9. The application validates the recommendation against business rules and authorization requirements.
10. The system stores relevant events and sends permitted updates to the monitoring dashboard.

Financial actions such as refunds must remain subject to deterministic validation and appropriate authorization. The language model alone must not execute unrestricted transactions.

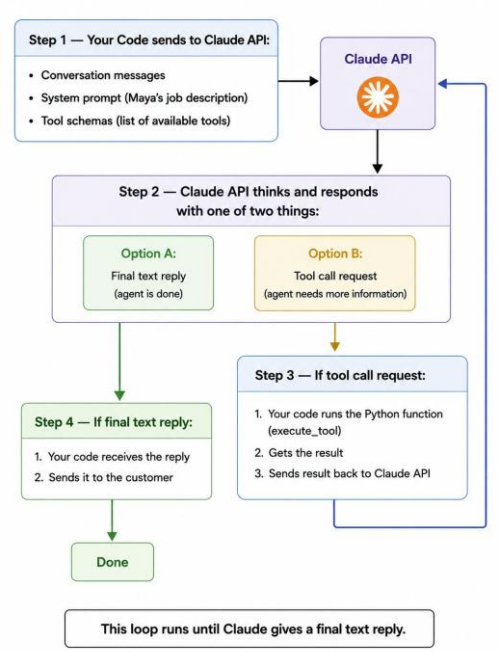



**Figure 3.6:** *Claude API Tool-Calling Agent Loop*

## IV. Results

The system described in this paper is a proposed design for a fictional company, and no quantitative evaluation has been carried out yet. This section therefore reports the outputs the system is designed to produce and the application areas it supports, rather than measured performance figures.

### *A. System Outputs*

- Customer responses grounded in verified order data and retrieved policy passages.
- Escalated cases accompanied by a structured risk assessment from the risk agent and a refund recommendation from the manager agent.
- Stored workflow events, including tool calls and case updates, that are streamed to the monitoring dashboard through SSE.
- Two implementations of the same agent workflow: a framework-free agent loop and a LangChain version.

### *B. Application Areas*

- **Automated Customer Support:** Handles routine questions related to products, orders, delivery, and company policies.
- **Order and Delivery Tracking:** Retrieves verified order details and delivery status through approved backend tools.
- **Refund Management:** Collects relevant evidence, retrieves refund policies, and routes complex requests for review.
- **Risk Assessment:** Identifies predefined patterns that may require additional verification.
- **Knowledge Management:** Provides policy-grounded responses using searchable company documents.
- **Staff Monitoring:** Allows authorized staff to observe important agent activities and escalation events in real time.
- **Customer Service Analytics:** Stores structured workflow events that can support later analysis of common questions, escalation frequency, and system performance.

## V. Discussion

### *A. Advantages*

**Automation:** Routine customer support requests can be handled with less manual intervention.

**Specialized Responsibilities:** Separate agents can handle customer communication, risk assessment, and managerial review.

**Policy-Grounded Responses:** RAG provides access to company-specific documents and helps reduce unsupported answers.

**Controlled Business Access:** Django tools allow the application to validate requests before accessing or modifying business records.

**Real-Time Monitoring:** SSE allows staff to observe important workflow events without repeatedly requesting dashboard updates.

**Modularity:** The application separates the backend, agents, retrieval subsystem, and monitoring interface.

**Educational Value:** Implementing the agent loop before using LangChain helps developers understand the underlying agent workflow.

### *B. Limitations*

**Implementation Complexity:** Coordinating multiple agents, Django services, retrieval components, and streaming connections requires careful design.

**External API Dependency:** The application depends on the availability, latency, pricing, and rate limits of the external AI service.

**Retrieval Errors:** Incomplete documents or irrelevant search results may cause agents to produce incomplete or unsupported responses.

**Risk Assessment Errors:** The risk agent may incorrectly flag legitimate customer behavior or fail to identify suspicious patterns.

**Computational and Operational Costs:** Multiple model calls, embedding generation, vector storage, and hosting may increase operating expenses.

**Security Requirements:** Customer information and business operations must be protected through access controls, validation, and secure credential management.

**Latency:** Multi-step agent workflows may take longer than conventional request-response operations.

**Human Oversight:** Sensitive refund decisions and uncertain cases may still require authorized human review.

### *C. Comparison with Previous Studies*

Earlier work shows that agentic RAG can improve response quality over naïve RAG [1], that retrieval strategy affects accuracy in multi-agent support systems [2], and that indexing quality and query refinement matter in production deployments [4]. Other studies address service-strategy mining [3] and organization-specific chatbots built with LangChain [5]. The present system differs by combining a three-agent structure, verified business tools, policy retrieval, risk assessment, and live monitoring in one application. Unlike the reviewed studies, it has not yet been evaluated with measured results, so no claim of superiority is made.

## VI. Conclusion and Future Work

This research proposes a Multi-Agent AI Customer Support System for CoolBreeze AC using Python, Django, Claude API, Retrieval-Augmented Generation, ChromaDB, and LangChain.

The proposed architecture combines specialized AI agents with verified business data and company-document retrieval. Maya handles routine customer inquiries, the manager agent reviews escalated cases, and the risk agent evaluates predefined risk indicators. Django provides the backend infrastructure, while SSE supports live monitoring of important workflow events.

The system initially implements the agent loop without an orchestration framework and subsequently recreates the workflow using LangChain. This approach provides a practical foundation for understanding agentic AI and evaluating framework-based development.

The proposed system may improve support consistency, information retrieval, and workflow visibility. However, its actual effectiveness must be established through implementation, testing, and measured evaluation. Future work may include advanced evaluation metrics, improved retrieval strategies, stronger audit mechanisms, and integration with additional customer service channels.

## Appendix A: Combined Literature Review Table

**Table 1:** *Comparison of related studies on AI-based customer support*

| Reference | Methodology | Key Finding | Advantages | Disadvantages | Performance Matrix |
|---|---|---|---|---|---|
| [1] Supporting Human Operators During Customer Service Interactions with Agentic-RAG (2026) — Barrionuevo-Valenzuela et al., IWSDS 2026, pp. 348–356. Paper link | Developed an Agentic-RAG copilot using agents for routing, retrieval validation, and response generation. Evaluated using the ABCD dataset, manual annotation, and an LLM-based evaluator. | Reported improvements in response quality, relevance, and accuracy compared with naïve RAG, particularly for ambiguous and multi-topic conversations. | Real-time operator assistance; grounded responses; conversation summaries and suggested replies. | Multi-agent coordination increases implementation complexity; effectiveness depends on retrieval quality. | Dataset: ABCD. Baseline: Naïve RAG. Outcome: Improved response quality, relevance, and accuracy. Exact numerical scores require the paper's results tables. |
| [2] ShopEase: A Generative AI-Based Multi-Agent Framework for Intelligent Enterprise Customer Support Using Hybrid RAG (2026) — Tiwari et al., arXiv:2609.13856. Paper link | Combines intent, CRM, memory, hybrid RAG, escalation, and supervisor components. Compares FAISS, BM25, Reciprocal Rank Fusion (RRF), and cross-encoder reranking. | FAISS-only reportedly achieved the highest accuracy of 85.37% on 2,632 held-out customer queries. Weighted RRF achieved 85.07%. | Combines semantic and keyword retrieval; maintains customer context; supports escalation; compares retrieval strategies. | Unknown queries remain challenging; cross-encoder reranking adds overhead and reduced accuracy in tested configurations. | Dataset: 2,632 queries across six categories. FAISS: 85.37%; Weighted RRF: 85.07%; BM25: 55.74%; RRF + Cross-Encoder: 83.24%; Top-10 Hybrid + Cross-Encoder: 81.88%. |
| [3] ChatMap: Mining Human Thought Processes for Customer Service Chatbots via Multi-Agent Collaboration (2025) — Jiang et al., Findings of ACL 2025, pp. 11927–11947. Paper link | Uses multiple agents to extract requests and solutions from historical customer-service dialogues, cluster requests, infer service strategies, and refine outputs through quality inspection and reflection. | Offline experiments performed comparably to human-annotated service thoughts and outperformed other baselines. Taobao A/B tests reported improved customer satisfaction and request handling. | Learns service strategies from historical conversations; reduces manual annotation; includes quality inspection; tested in e-commerce. | Depends on historical dialogue quality and coverage; multi-agent processing may add computational cost. | Offline: Comparable to manual annotations and better than other baselines. Online: Taobao A/B testing reported improvements in customer satisfaction and request handling. Exact scores require the full paper. |
| [4] Optimizing Retrieval-Augmented Generation for E-Commerce How-To Assistance (2026) — Fuchs et al., ACL 2026 Industry Track, pp. 459–466. Paper link | Studies a production RAG assistant for eBay customer support, focusing on document chunking, contextualization, query refinement, and automatic LLM-based evaluation. | Highlights the importance of indexing quality, query refinement, and systematic evaluation in production customer-support RAG systems. | Uses company-specific knowledge; supports real-time assistance; provides production-oriented optimization insights. | Requires reliable knowledge-base maintenance and careful retrieval configuration; findings may not generalize to all organizations. | Application: eBay customer support. Evaluation focus: chunking, contextualization, query refinement, and LLM-based evaluation. Exact metrics should be verified from the full paper. |

| Reference | Methodology | Key Finding | Advantages | Disadvantages | Performance Matrix |
|---|---|---|---|---|---|
| [5] Automating Customer Service using LangChain: Building Custom Open-Source GPT Chatbot for Organizations (2023) — Pandya and Holia, arXiv:2310.05421. Paper link | Presents Sahaay, an organization-specific customer-service chatbot using LangChain, web scraping, embeddings, and Google Flan-T5 models. Demonstrated in an educational institution. | Demonstrates how organizational information and language models can be integrated to provide context-aware customer support. | Uses open-source technologies; supports organization-specific knowledge; provides personalized responses; can reduce dependence on static FAQs. | Depends on accurate source data; requires maintenance of scraped information and model integration; quantified performance limitations require further verification. | Application: Educational institution. Models: Flan-T5 XXL, Base, and Small. Approach: Web scraping and embeddings. Exact accuracy and latency figures should be verified from the full paper. |

## REFERENCES


[1] Barrionuevo-Valenzuela, J., Calderón-González, D., Callejas, Z., & Griol, D. (2026). Supporting human operators during customer service interactions with agentic-RAG. In *Proceedings of the 16th International Workshop on Spoken Dialogue System Technology (IWSDS 2026)* (pp. 348–356). Association for Computational Linguistics. https://aclanthology.org/2026.iwsds-1.35/

[2] Tiwari, A. K., et al. (2026). ShopEase: A generative AI-based multi-agent framework for intelligent enterprise customer support using hybrid retrieval-augmented generation. *arXiv preprint arXiv:2609.13856*. https://arxiv.org/abs/2609.13856

[3] Jiang, et al. (2025). ChatMap: Mining human thought processes for customer service chatbots via multi-agent collaboration. In *Findings of the Association for Computational Linguistics: ACL 2025* (pp. 11927–11947). Association for Computational Linguistics. https://aclanthology.org/2025.findings-acl.617/

[4] Fuchs, et al. (2026). Optimizing retrieval-augmented generation for e-commerce how-to assistance. In *Proceedings of the ACL 2026 Industry Track* (pp. 459–466). Association for Computational Linguistics. https://aclanthology.org/2026.acl-industry.31/

[5] Pandya, K., & Holia, M. (2023). Automating customer service using LangChain: Building custom open-source GPT chatbot for organizations. *arXiv preprint arXiv:2310.05421*. https://arxiv.org/abs/2310.05421