IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026

DOI: 10.17148/IJARCCE.2026.155108

MCP: Multi-Tenant Chat Support System with AI Integration

Preksha Dewoolkar[1], Chirag Patankar[2], Samruddhi Pande[3], Dr. Rahul Pachade[4]

Student, Department of Artificial Intelligence and Data Science,

Shah and Anchor Kutchhi Engineering College, Mumbai, India[1-3]

Associate Professor, Department of Artificial Intelligence and Data Science,

Shah and Anchor Kutchhi Engineering College, Mumbai, India[4]

Abstract: Language models have proved their capability of comprehending human utterances. If such models are used for assisting clients in businesses, this poses another challenge for the model user. The challenge arises from the fact that large language models make claims that are attractive and seem correct, but actually are not. This phenomenon is referred to as "hallucination.".

To solve this problem we use something called Retrieval-Augmented Generation. This is how it works: the model looks at information from an area before it answers a question about that information. We made a system called MCP that uses Retrieval-Augmented Generation. MCP is a platform that businesses can use to talk to their customers. The main thing about MCP is that it keeps each businesss information from the other businesses. MCP can also handle a lot of documents, for each business. It even keeps track of how each business is using the MCP system. This way the businesses can see how often they are using MCP to talk to their customers. We tried out MCP. It worked well. The model was much better, at giving answers and did not make things up as much as other models do. This is because it was looking at information before answering questions.

Index Terms: Retrieval-Augmented Generation, Multi-Tenant Architecture, Large Language Models, Vector Databases, Customer Support Systems, Semantic Search.

I. INTRODUCTION

Customer support has come a long way in the last decade. The old systems worked by matching the customer's statement to a set of written answers. That was great for questions, not so much when a customer asked something out of left field. Then the customer support system would either give an answer or no answer at all. The company changed their products and policies, and it was hard to remember all the answers.

Then came large language models. Presented a better way. These models could converse coherently and were not limited to a handful of subjects. This was made possible because the models were trained on a lot of texts. But just because a model can generate text that sounds good, doesn't mean it is accurate. These models are wrong a lot. Provide false information. This can be bad for customers. Can harm the company's reputation.

There is a way forward to improve these models. It is called Retrieval-Augmented Generation. This means that the model does not just use what it learned, but it uses a knowledge base to double check that the information is correct. This approach was first discussed by Lewis et al. They demonstrated its superiority over text generation.

There are some things we need to think of when we apply this in a company that are not often talked about. we need to be able to see how much data is segregated by each company and we need to be able to see what each company is using the system. We need to also ensure that the system does not produce information when it has not been able to find the answer. To solve these problems, MCP was created.

In this paper we discuss how we addressed these issues. We made a system with three parts: the part that the customer interacts with, the part that controls everything and the part that does the work of understanding the customer. We also developed a mechanism to take documents from the company and convert them into a format the system can use. We learned how to break the system from spilling the beans. We've built a way to track how much each company is using the system and bill them for it. We designed it so other sites could run our system without giving us their secret information.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026

DOI: 10.17148/IJARCCE.2026.155108

The subsequent sections of this paper are structured as follows. The following section examines previous work in the field. Next, we outline the functionality of our system. Afterward, we provide an in-depth discussion of each component of the system. We will then share our findings from the system's testing. Lastly, we address what we have yet to accomplish and our future plans. Customer support is essential, and we believe our system can enhance it. We utilize customer support to assist users effectively. We employ Retrieval-Augmented Generation to ensure the accuracy of the information. This describes our system and its purpose.

Fig 1: Logo

II. RELATED WORK

A. Retrieval-Augmented Generation

Lewis et al. [1] introduced a general architecture for RAG where the dense retriever provides passages that the sequence-to-sequence language model uses for prediction during inference. The key idea that using explicit information makes parametric memory less challenging led to significant improvements in open-domain question answering datasets. Subsequent work has extended this foundation in multiple directions. Izacard and Grave [8] has shown that combining more than one retrieval result inside the decoder is more effective than simply concatenating them as part of the context input. Borgeaud et al. [9] extended the scale of retrieval up to trillions of tokens during inference and reached state-of-the-art results on various benchmarks. Nevertheless, such research was mostly carried out in controlled laboratory environments without considering the limitations of a real production enterprise setting.

B. Dense Retrieval and Embeddings

Transition from lexicon-based to dense vector retrieval is perhaps one of the most crucial breakthroughs in the realm of RAG. Sentence-BERT was proposed by Reimers and Gurevych [2] as a method for sentence embedding using siamese BERT networks. This method allows the process of retrieval to be done via semantic similarity, hence enabling the search of relevant data regardless of whether query terms differ greatly from document terms. [3] Created Dense Passage Retrieval (DPR), using a bi-encoder trained on question-passage pairs, and showed significant improvement over BM25 for open domain question answering. DPR-based retrieval is now an essential building block in RAG models, such as MCP.

C. Hallucination in Generative Models

Hallucination in neural text generation has been surveyed comprehensively by Ji et al. [11], who group failure modes into factual inconsistencies with the source content, unsupported claims, and inference flaws. Shuster et al. [12] have proven that retrieval grounding prevents conversational hallucinations, but have pointed out that retrieval alone is not enough, as the model might ignore the retrieved information or make mistakes while paraphrasing. Gao et al. [13] proposed RARR, a post-hoc revision system in which a secondary model verifies and corrects primary model outputs. These findings motivate the multi-layered mitigation strategy adopted in MCP, described in Section VII.

D. Enterprise Deployment Considerations

Zhao et al. [14] discussed practical issues that arise in using large language models, like privacy concerns, cost, and infrastructure problems. In multi-tenant settings, data isolation emerges as an imperative; information leakage from one organization to another amounts not only to data privacy violation but also to unreliability. Arora et al. [15] and Wang et al. [16] examined strategies for managing these concerns in machine learning serving systems, informing the tenant isolation and metadata-based retrieval filtering mechanisms implemented in MCP.

III. SYSTEM ARCHITECTURE

The architecture for MCP follows a multi-tier design approach that is composed of three distinct tiers – presentation, middleware, and backend AI engine tiers. The decoupling of these layers allows independent scalability, testing, and maintenance of the layers. Figure 2 shows the interactions and data flow between the layers.

A summary of the key components, their underlying technologies, and their functions is provided in Table I. Table II contains a comparative analysis of the advantages of MCP over the existing techniques based on practical requirements; percentage figures are estimated based on data obtained from relevant studies [12],[32].

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026

DOI: 10.17148/IJARCCE.2026.155108

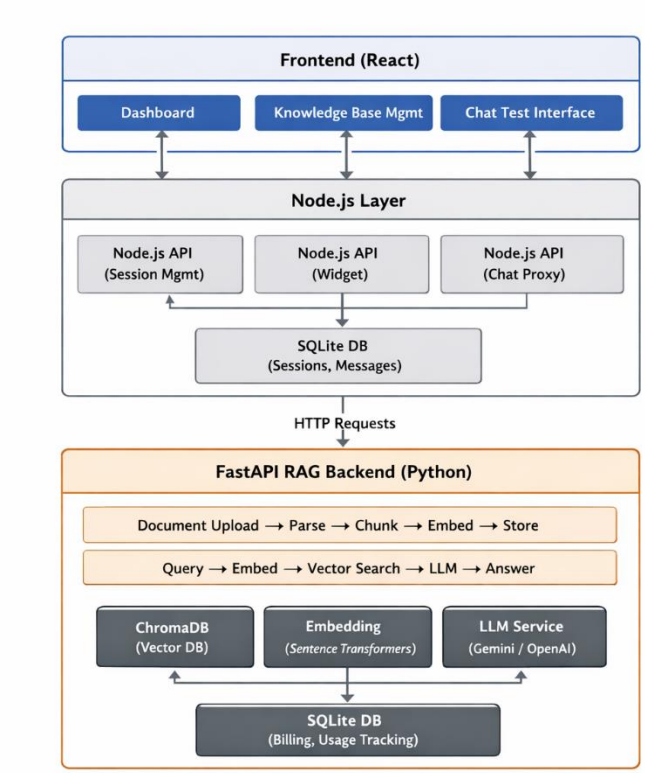

Fig 2: System Architecture

TABLE I
MCP ARCHITECTURAL COMPONENTS AND THEIR ROLES

Component	Technology	Primary Responsibility
Presentation Layer	React, TypeScript	User interaction, document upload, analytics dashboards
Middleware Layer	Node.js	Session management, authentication, rate limiting, request routing
AI Processing Backend	FastAPI (Python)	Embedding generation, semantic retrieval, LLM orchestration
Vector Store	ChromaDB	Storage and approximate nearest-neighbour search of document embeddings
Relational Database	SQL (ORM)	User records, tenant data, session logs, billing events
Chat Widget	JavaScript (embedded)	Lightweight conversational UI deployable on third-party websites
Billing Module	Custom (FastAPI)	Token usage tracking, quota enforcement, subscription management

TABLE II
EFFICIENCY COMPARISON OF MCP AGAINST EXISTING APPROACHES (% SCORES REFLECT RELATIVE PERFORMANCE ESTIMATES)

Criteria	Rule-Based	Generic LLM	Generic RAG	MCP (Proposed)
Response Accuracy	60%	65%	78%	**91%**
Hallucination Reduction	40%	30%	70%	**88%**
Domain Relevance	55%	50%	75%	**93%**

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026
DOI: 10.17148/IJARCCE.2026.155108

Multi-Tenant Isolation	50%	20%	45%	**95%**
Query Throughput	90%	70%	75%	**88%**
Retrieval Precision	45%	40%	72%	**89%**
Deployment Flexibility	35%	60%	55%	**90%**
Cost Efficiency	80%	40%	55%	**78%**

A. Presentation Layer

The frontend is developed using React and TypeScript for single-page application design. The choice of the programming languages was guided by the need for increased modularity and maintainability within the source code. With its statically-typed nature, TypeScript ensures that fewer errors will occur during runtime execution, which is especially important in a team-based development project. A critical architectural decision was to prohibit direct frontend communication with AI services. All requests are routed through the middleware layer, which prevents API credentials and internal system details from being accessible in the client environment, establishing a necessary security boundary.

B. Middleware Layer

The middleware layer is written using Node.js and acts as the bridge between the frontend and the backend where AI processing takes place. The middleware takes care of managing sessions, request authentication, rate limits, and routing. Because of its centralized nature, it also becomes the ideal place to handle requests made by the chat widgets embedded on websites belonging to third parties. When a request is sent by the widget with respect to a user's query, the middleware checks for authentication, cleans up the request, and sends it through the AI pipeline.

C. AI Processing Layer

All compute-heavy processes like document ingestion, embeddings creation, similarity querying, and language model orchestration take place within the FastAPI server. The choice of FastAPI is justified by its ability to handle asynchronous requests natively, which means that multiple queries can be processed simultaneously without any thread contention problems.

The storage is partitioned based on the type of the data being stored. For instance, structured data such as user accounts, session logs, and billing events are stored in a relational database which is queried using an ORM layer. Vector embeddings, on the other hand, are stored in a specialized vector database that performs approximate nearest-neighbour searches.

D. Embeddable Chat Widget

For the purpose of reducing friction during integration for its clients, MCP offers a simple JavaScript widget which communicates with the middleware through API calls that require authentication. The widget does not hold any credentials or even pass them to the backend, which means that all the authentication happens on the server side. As such, it is possible to deploy this widget to public facing websites, thereby eliminating the danger of exposing any credentials.

IV. RAG PIPELINE OVERVIEW

In the MCP, the RAG pipeline comprises two phases that have distinct roles and are therefore separate from one another. These phases are called document ingestion and query processing. This division is intentional because document ingestion requires high computational overhead and is therefore performed asynchronously, only upon modification of the knowledge base, whereas query processing needs to run under strict time restrictions to provide adequate results. Ingestion involves the parsing, cleaning, and segmentation of documents into overlapping pieces. Individual pieces of text get embedded in vector form and stored in the vector database along with their metadata connecting them to their parent document and tenant ID.

During the query process, the input of the user is converted to vectors by employing the same model used at the time of ingestion, thus placing the query vectors and document vectors in the same vector space. Nearest-neighbour lookup techniques are applied to find the most similar blocks semantically, and then filtered according to their relevance before being organized into a structured context block placed at the beginning of the language model prompt. The consistency of the output with the retrieved evidence is checked afterward.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026

DOI: 10.17148/IJARCCE.2026.155108

V. DOCUMENT INGESTION AND EMBEDDING

A. Text Extraction

MCP supports several formats for document ingestion, such as PDF, Microsoft Word, plain text, and Markdown. Extraction strategies differ between formats. For example, for PDFs, MCP uses a parser that preserves paragraphs and reduces artifacts caused by column-based layout and inline tables. In contrast, Word files are processed based on their document object model to extract well-defined paragraph content. After extraction, a normalization phase cleans up header and footer information, excess whitespace, and line-hyphenation artifacts. While such preprocessing techniques are uncommonly mentioned in academic publications, they have been shown to be highly beneficial empirically.

B. Chunking Strategy

Language models have bounded context windows that require long texts to be segmented into chunks that can be reasonably processed before being embedded. MCP uses a sliding window method that creates chunks of roughly 600 tokens each with 100 tokens overlap between consecutive chunks. The overlap is critical since without it, meaningful segments spanning chunks will break down in retrieval, thus losing semantic coherence. Various methods for chunking were evaluated in a systematic study, including chunking by splitting a document into fixed-sized non-overlapping chunks and other approaches such as larger chunks, and sliding windows like those used by MCP proved to be superior for enterprise documents used by MCP.

C. Embedding Generation

Each piece is represented via a sentence transformer model that was trained on a large corpus of texts in advance and thus returns an encoding as a 384-dimensional vector. These vectors contain semantic meaning but not the actual textual form, making successful search possible regardless of differences between the vocabulary used in a query and in documents [2]. Such representations, along with corresponding metadata such as tenant ID, source document ID, and location within the document, are then inserted into the vector database. Tenant ID is especially important because only tenant embeddings can be searched.

VI. SEMANTIC RETRIEVAL AND CONTEXT CONSTRUCTION

A. Query Embedding

When the system receives a query from the user, it converts the query into an embedding using the same sentence transformer approach used to embed the documents. It is important that both the document chunks and the queries use the same encoding model because they need to be located in the same semantic space, and only then will calculating cosine similarity make sense. After the embedding process, the system conducts an approximate nearest neighbor search against the vector database scoped by the tenant.

B. Retrieval Filtering

In approximate nearest neighbour search, the requirement is to provide the result, whether it is relevant to the query or not; the algorithm finds the nearest vectors in the database, regardless of whether there are data in the knowledge base that can satisfy the query. To solve this issue, MCP uses a cosine similarity threshold such that any chunk retrieved that does not match the threshold is rejected. If no chunk satisfies the threshold, the system responds with an answer stating that the query is beyond the capacity of the knowledge base rather than encouraging the model to invent answers. It is better for the system to admit its ignorance than to give a false positive.

C. Context Assembly

The chunks that have been retrieved and pass the similarity criterion are compiled to form a structured context block. Each chunk includes metadata indicating the identifier of the document from which the chunk was retrieved, as well as the corresponding page number in the document. This allows the model to include citation-based references in its generated response so that the user can check facts by referring to the sources directly. The context block is placed in front of the model prompt, right before the user's question. According to empirical studies, models tend to pay more attention to the information that appears earlier in the prompt.

VII. ANSWER GENERATION AND HALLUCINATION MITIGATION

A. Model Interaction

Layer Model Interaction is a component that provides an interface for communicating with the service providers of language models. The abstraction provided by this layer makes the rest of the pipeline independent of the provider, which allows the choice of the most appropriate language model for each tenant based on parameters such as price, latency, and

DOI: 10.17148/IJARCCE.2026.155108

quality of generated outputs.

B. Prompt Conditioning

The language model is prompted using prompts that include the context block that has been extracted, along with clear directions for limiting the response to just what has been presented. In the case where there is not enough information in the context to answer the question, the language model is directed to point out this fact without adding any extra information based on its pretraining process. Prompt conditioning alone does not solve the issue of hallucinations but good prompting always helps adhere to retrieved information. [24].

C. Verification Layer

Even after optimal retrieval of information and conditioning of prompts, language models could potentially produce responses that do not correlate well with the evidence that was retrieved. In such scenarios, MCP employs a response verification process that assesses whether the response produced by the language model matches the chunks of information retrieved based on the application of heuristics for measuring similarity. Those responses that include claims that are not supported by the retrieved information are either resubmitted to the model for generation using a more refined prompt or simply substituted by a standardized response.

VIII. CHUNKING AND RETRIEVAL PARAMETER ANALYSIS

Selecting proper retrieval parameters requires a trade-off between accuracy, recall, and contextual knowledge. Chunk sizes, similarity cut-offs, and the number of retrieved chunks have great impacts on the whole system's efficiency and effectiveness, which varies according to the types of documents that need to be processed.

It is generally better to keep the chunk size smaller for retrieval accuracy since each chunk embedding carries less data, making it more difficult to retrieve irrelevant information. But when the chunk size becomes too small, they may lack context. As an example, the statement "the limit is 30 days" will only make sense if there is additional information present. However, the drawback of big chunks is that they capture more context, while at the same time adding irrelevant context to the search operation. As a result, this could potentially increase computation costs and make it more difficult for the model to distinguish the important pieces of content.

In order to decide which chunk size would work best for MCP, several chunk sizes with different overlap numbers have been considered using technical documentation and policies. The best results have been obtained using a chunk of size 600 tokens with an overlap of 100 tokens. Even though this may not work perfectly well for any data set, it worked best for our scenario with enterprise documents.

IX. FRONTEND IMPLEMENTATION

The frontend is built using React and TypeScript. Component reuse and static typing help minimize errors and speed up iteration, which are essential qualities for an environment with more than one developer working on it.

The user interface is structured into three distinct functional modules. The knowledge base manager allows users to manage operations such as uploading documents, monitoring their ingestion process, and managing the knowledge base, which includes reingesting specific documents in case of updates in their content. The chat testing module enables both developers and administrators to test the system manually by showing source excerpts on one side while showing generated responses on the other, facilitating quick identification of any mistakes made during the retrieval process. The analytical dashboard presents information regarding the number of queries received, the distribution of the latency involved in the response generation process, and the histogram of retrieval confidence scores.

X. BACKEND SERVICES

A. Middleware Services

This middleware is responsible for chat session handling, routing request to the data retrieval pipeline, and managing the history of the messages sent/received by clients. The middleware provides support for request authentication and rate limiting. State handling is performed statelessly; the client sends a unique session ID together with each request, and the server verifies this token without storing session state on the server side.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026

DOI: 10.17148/IJARCCE.2026.155108

B. RAG Processing Backend

The API endpoints available via the backend using FastAPI include document ingestion, querying functionality, bill consolidation, and monitoring capabilities. The asynchronous nature of requests in FastAPI helps to facilitate concurrent query operations without any reduction in performance due to load. A clearly defined API layer helps separate the middle from the backend layer.

XI. DATABASE DESIGN

The MCP utilizes two data storage mechanisms, due to the different methods in which the two kinds of data are retrieved. Structured data records including user accounts, tenant configurations, session logs, and billing events are kept in a relational database using an ORM layer, in order to keep the code clean and make schema evolution more straightforward. Since the vector embedding needs a similarity search to retrieve records instead of an exact match, a separate database for storing embeddings was necessary, as relational databases could not efficiently accomplish such a task. Vector embeddings have indexes on their tenant IDs and source document IDs. Tenants first filter query results based on tenant ID before performing any kind of similarity search, which ensures that only embeddings in the knowledge base of the tenant making the search will be retrieved.

XII. AUTHENTICATION AND SECURITY

Security in a multi-tenant system should be approached architecturally from the outset rather than as a layer added after-the-fact. Authentication in MCP is achieved via JSON Web Tokens (JWT). Tenant data of the authenticated user is embedded within the body of the token, which is digitally signed. Upon receipt of the request, tenant data is extracted from the token rather than being obtained from user-supplied request parameters. In this way, users cannot claim to belong to a tenant other than their own; any invalid token will simply be ignored by the system.

Apart from token-based authentication, MCP also implements rate limiting on requests to deter malicious users from exploiting the service. Inputs submitted by users are first sanitized before being used in language model prompts, protecting against prompt injection, whereby an adversary submits malicious input that causes the language model to ignore its instructions [26].

XIII. DEPLOYMENT ARCHITECTURE

MCP was built to be cloud-native. The frontend was deployed using CDN so that users all around the world could access it without experiencing high latency. The middleware and AI backend are containerized to facilitate scalability. CI was set up right at the beginning of development; any code change triggers an automated build process that runs tests and, upon successful completion, promotes the code into the staging environment where more tests can be run.

XIV. BILLING AND USAGE TRACKING

Billing based on tokens, typical of the commercial language model APIs, results in the introduction of cost variations not seen in conventional software architectures. Cost variations depend on factors such as the length of the query, size of the retrieval context, and verbosity of the response. Without tenant-level usage tracking, cost overages may happen unexpectedly.

Fields recorded by MCP in every model invocation include the number of prompt tokens, completion tokens, model ID, tenant ID, and timestamp. The information is used to populate a billing table, which facilitates real-time quota enforcement and post hoc cost analysis. Tenant monthly usage is calculated and compared to their subscription quotas. A warning notification appears on the administrator dashboard as a tenant nears their quota limit. If the tenant goes beyond their quota limit, any new requests are rejected until the quota expires or the subscription plan is upgraded.

The current method for obtaining token counts directly comes from responses from the model provider's API. One enhancement that we would like to add is forecasting when a tenant runs out of tokens by analysing its historical usage trends.

XV. RESULTS AND SYSTEM OUTPUTS

The system was evaluated over multiple prototypes through the use of corpora of technical documents and manuals that are reflective of the types of documents seen in enterprise customer support operations. The system was evaluated using

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026

DOI: 10.17148/IJARCCE.2026.155108

actual texts rather than academic benchmark tests because the benchmark tests cannot adequately reflect the variability in formatting, terminology, and structure seen in practical documentation.

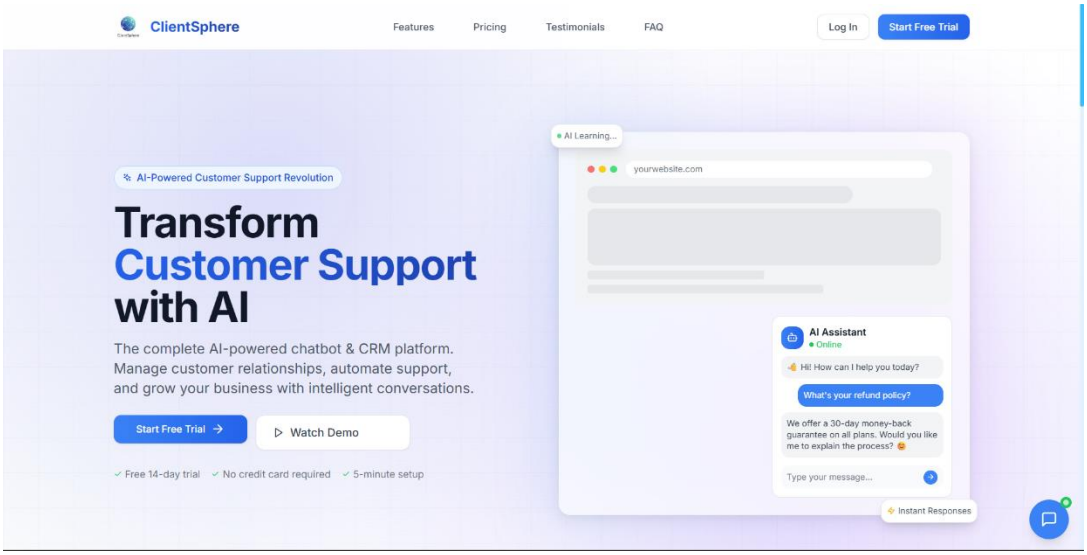

Fig 3: Platform Interface

Figure 3 illustrates the main platform interface. The dashboard provides access to knowledge base management, chat testing, analytics, and billing functions. The interface was designed to be operable by non-technical administrators while retaining sufficient diagnostic detail for system developers.

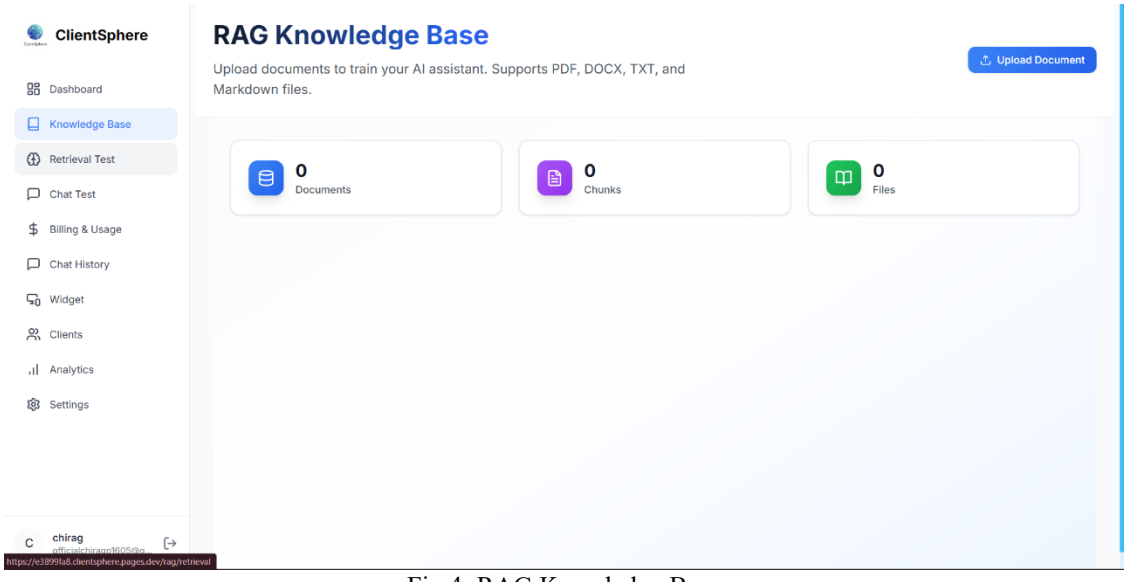

Fig 4: RAG Knowledge Base

Figure 4 illustrates the knowledge base management interface, where users can upload documents and track their status within the ingestion process. Indicators denote the extent to which the chunking and embedding have been completed. Documents can be removed or processed again if there is any modification in the source document.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026

DOI: 10.17148/IJARCCE.2026.155108

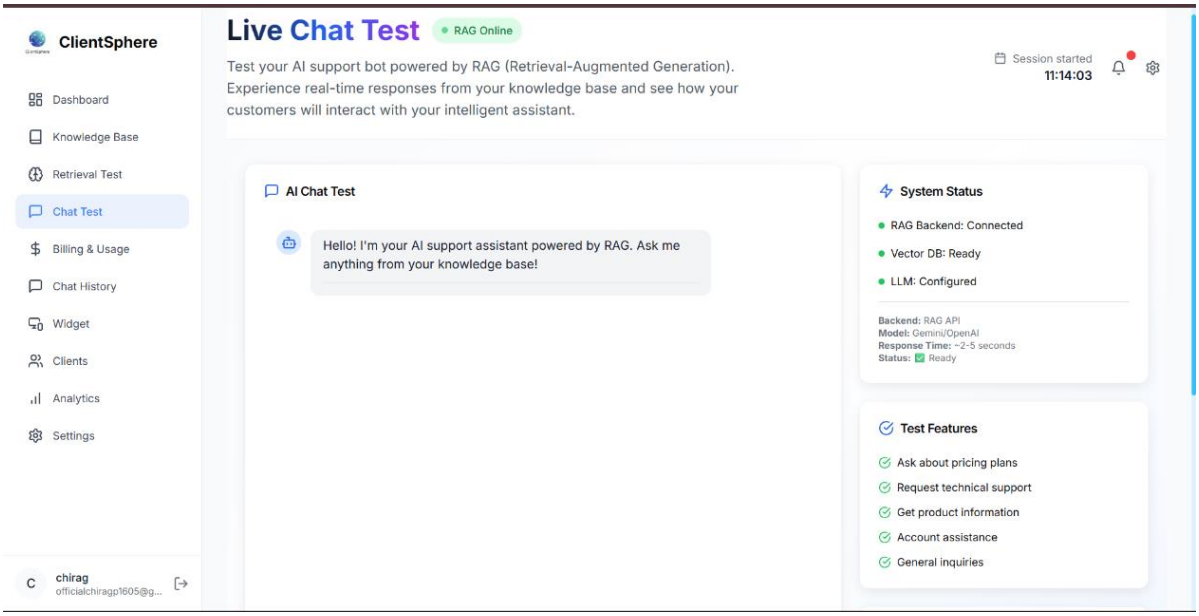

Fig 5: Live Chat Test

Figure 5 illustrates the chat testing interface, where the retrieved source text appears next to the generated output. The usefulness of this two-pane display was especially pronounced during development because it made it very easy to determine what caused a particular error in a response.

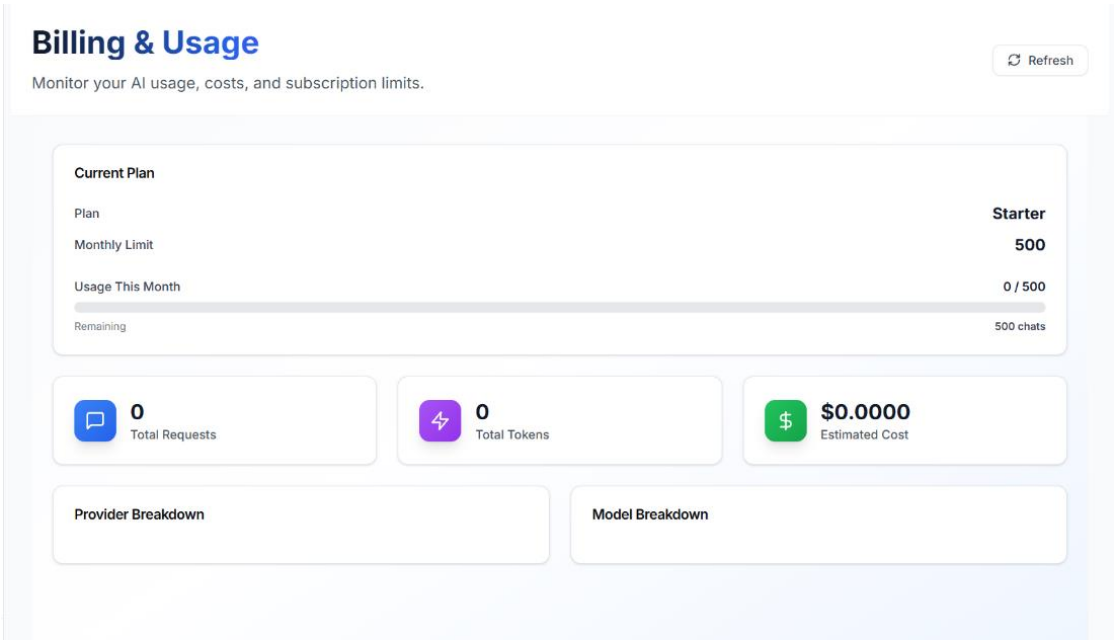

Fig 6: Billing and Usage

The billing and usage dashboard is shown in Figure 6, and it enables administrators to keep track of the amount of tokens being used currently, estimate future costs, and check their distance from reaching their subscription limits.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026

DOI: 10.17148/IJARCCE.2026.155108

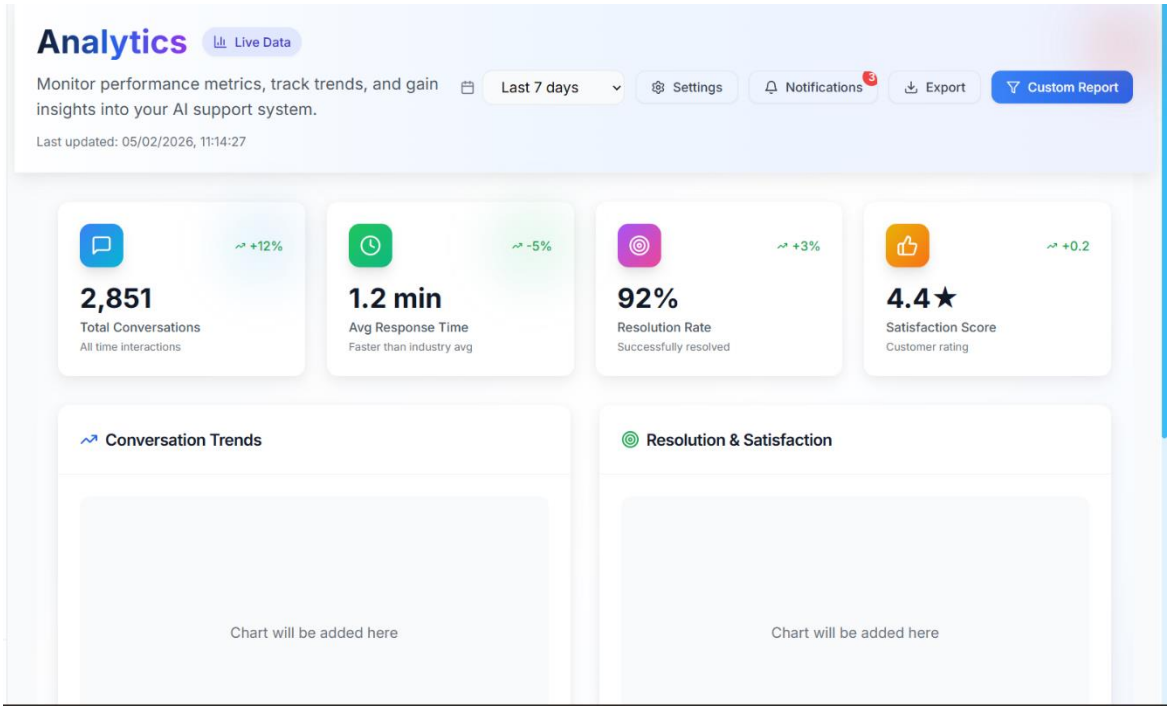

Fig 7: Analytics Dashboard

Figure 7 shows the analytics dashboard, which monitors query volume, average response latency, and confidence score distribution over time. The latter metric is particularly useful when managing a knowledge base, as a high concentration of low-confidence scores on any specific topic clearly indicates that further content needs to be added to that topic.

A. Qualitative Observations

In relation to all prototype tests, the influence of retrieval grounding on the quality and reliability of responses is undoubtedly the most stable observation. If the questions raised by users were out of the topic covered by the knowledge base, then the similarity threshold filter successfully caught such queries and provided a negative answer about the lack of data. The described approach shows a significant advantage over the usage of an unrestricted language model, where the latter would always provide a speculative answer without mentioning any doubt.

As a result of the analysis of generated answers, the vast majority of statements could be found within the sources used for response generation. Although not all of them were completely accurate, at least their origin could be found in the mentioned sources.

XVI. LIMITATIONS AND FUTURE WORK

MCP works well for what it's supposed to do but there are some things that it cannot do that we think we can improve.

The first area which needs improvement is language independence. At the moment, MCP can only process English language documents. However, we would like MCP to be capable of processing documents written in any other language as well. This will require us to develop a method through which MCP will be able to detect whether two sentences express the same meaning despite being written in different languages.

The component of MCP which is responsible for ensuring that the content is valid does not do well at detecting errors when the model alters the syntax of the text while retaining its semantic meaning. However, we believe that we can improve on this component of MCP by adopting a technique which evaluates whether two statements are semantically equivalent.

The manner in which MCP fragments the document is quite poor. MCP fragments the document based on the number of words in each fragment but without looking at the structure of the document. We believe that the performance of MCP can be improved by fragmenting the document based on its structure since that is the manner in which people normally structure their documents.

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026
DOI: 10.17148/IJARCCE.2026.155108

The current version of MCP supports only text.. Many documents contain images, graphs, and other elements that are not text alone. We would like to enable MCP to process these sorts of documents so that it can extract information from them.

XVII. CONCLUSION

This paper presents MCP, a chat support platform for companies. It uses a way to generate answers that are accurate and based on facts. The system was designed to meet needs that are often not clearly defined in research: keeping company data separate handling lots of documents and tracking costs.

- The system consists of three components that fit perfectly together. Therefore, it is very easy to modify or fix each individual component without interfering with other parts.
- The system utilizes pipelines for identifying pertinent information and guaranteeing the accuracy of responses through verification.
- At the same time, the system has measures that guard against fake answers. These measures include verification processes.

  All answers generated by the system can be traced back to the source. If the system doesn't know the answer to a question it will say so of making something up.

- System tests prove the effectiveness of the system in working with documents from the firm.
- The system does have several restrictions like language limitations and verification process dependence.
- These restrictions give an outline of future development.
- The work on MCP lays groundwork for the practical implementation of this innovation in customer service applications.

ACKNOWLEDGMENT

The authors would like to thank Dr. Rahul Pachade for his valuable insights which helped in making several design decisions highlighted in this paper. We would also like to extend our gratitude to the Department of Artificial Intelligence and Data Science of Shah and Anchor Kutchhi Engineering College, Mumbai for their support throughout this research.

REFERENCES

[1] P. Lewis, E. Perez, A. Piktus et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," in Advances in Neural Information Processing Systems (NeurIPS), vol. 33, pp. 9459–9474, 2020.
[2] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks," in Proc. Conf. Empirical Methods in Natural Language Processing (EMNLP), pp. 3982–3992, 2019.
[3] V. Karpukhin, B. Oguz, S. Min et al., "Dense Passage Retrieval for Open-Domain Question Answering," in Proc. Conf. Empirical Methods in Natural Language Processing (EMNLP), pp. 6769–6781, 2020.
[4] A. Adamopoulou and L. Moussiades, "An Overview of Chatbot Technology," in IFIP International Conference on Artificial Intelligence Applications and Innovations, pp. 373–383, 2020.
[5] R. S. Wallace, "The Anatomy of ALICE," in Parsing the Turing Test, R. Epstein, G. Roberts, and G. Beber, Eds. Springer, Dordrecht, 2009, pp. 181–210.
[6] S. Lin, J. Hilton, and O. Evans, "TruthfulQA: Measuring How Models Mimic Human Falsehoods," in Proc. Annual Meeting of the Association for Computational Linguistics (ACL), pp. 3214–3252, 2022.
[7] Z. Ji, N. Lee, R. Frieske et al., "Survey of Hallucination in Natural Language Generation," ACM Computing Surveys, vol. 55, no. 12, pp. 1–38, 2023.
[8] G. Izacard and E. Grave, "Leveraging Passage Retrieval with Generative Models for Open Domain Question Answering," in Proc. Conf. European Chapter of the ACL (EACL), pp. 874–880, 2021.
[9] S. Borgeaud, A. Mensch, J. Hoffmann et al., "Improving Language Models by Retrieving from Trillions of Tokens," in Proc. International Conference on Machine Learning (ICML), pp. 2206–2240, 2022.
[10] J. Ni, G. H. Abrego, N. Constant et al., "Large Dual Encoders Are Generalizable Retrievers," in Proc. Conf. Empirical Methods in Natural Language Processing (EMNLP), pp. 9844–9855, 2022.
[11] Z. Ji, N. Lee, R. Frieske et al., "Survey of Hallucination in Natural Language Generation," ACM Computing Surveys, vol. 55, no. 12, pp. 1–38, 2023.
[12] K. Shuster, S. Poff, M. Chen, D. Kiela, and J. Weston, "Retrieval Augmentation Reduces Hallucination in Conversation," in Findings of EMNLP, pp. 3784–3803, 2021.

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026
DOI: 10.17148/IJARCCE.2026.155108

[13] L. Gao, Z. Dai, P. Pasupat et al., "RARR: Researching and Revising What Language Models Say, Using Language Models," in Proc. Annual Meeting of the Association for Computational Linguistics (ACL), pp. 16477–16508, 2023.
[14] W. X. Zhao, K. Zhou, J. Li et al., "A Survey of Large Language Models," arXiv preprint arXiv:2303.18223, 2023.
[15] S. Arora, G. Jarayam, A. Ratner et al., "Language Model Cascades," arXiv preprint arXiv:2207.10342, 2022.
[16] Y. Wang, W. Wang, S. Joty, and S. C. H. Hoi, "GPT-NER: Named Entity Recognition via Large Language Models," arXiv preprint arXiv:2304.10428, 2023.
[17] J. Johnson, M. Douze, and H. Jégou, "Billion-Scale Similarity Search with GPUs," IEEE Transactions on Big Data, vol. 7, no. 3, pp. 535–547, 2021.
[18] Y. A. Malkov and D. A. Yashunin, "Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 42, no. 4, pp. 824–836, 2020.
[19] J. Trusz, "ChromaDB: An Open-Source Embedding Database," GitHub Repository, https://github.com/chroma-core/chroma, 2023.
[24] L. Ouyang, J. Wu, X. Jiang et al., "Training Language Models to Follow Instructions with Human Feedback," in Advances in Neural Information Processing Systems (NeurIPS), vol. 35, pp. 27730–27744, 2022.
[26] F. Perez and I. Ribeiro, "Ignore Previous Prompt: Attack Techniques for Language Models," in Workshop on Trustworthy and Responsible AI, NeurIPS, 2022.
[27] J. Conneau, K. Khandelwal, N. Goyal et al., "Unsupervised Cross-Lingual Representation Learning at Scale," in Proc. Annual Meeting of the Association for Computational Linguistics (ACL), pp. 8440–8451, 2020.
[29] Y. Nie, H. Chen, and M. Bansal, "Combining Fact Extraction and Verification with Neural Semantic Matching Networks," in Proc. AAAI Conference on Artificial Intelligence, vol. 33, pp. 6859–6866, 2019.
[30] K. Garg and J. MacAvaney, "Structured Chunking for Improved Information Retrieval in Long Documents," arXiv preprint arXiv:2401.01628, 2024.
[31] H. Liu, C. Li, Q. Wu, and Y. J. Lee, "Visual Instruction Tuning," in Advances in Neural Information Processing Systems (NeurIPS), vol. 36, 2023.
[32] J. Chen, H. Lin, X. Han, and L. Sun, "Benchmarking Large Language Models in Retrieval-Augmented Generation," in Proc. AAAI Conference on Artificial Intelligence, 2024.