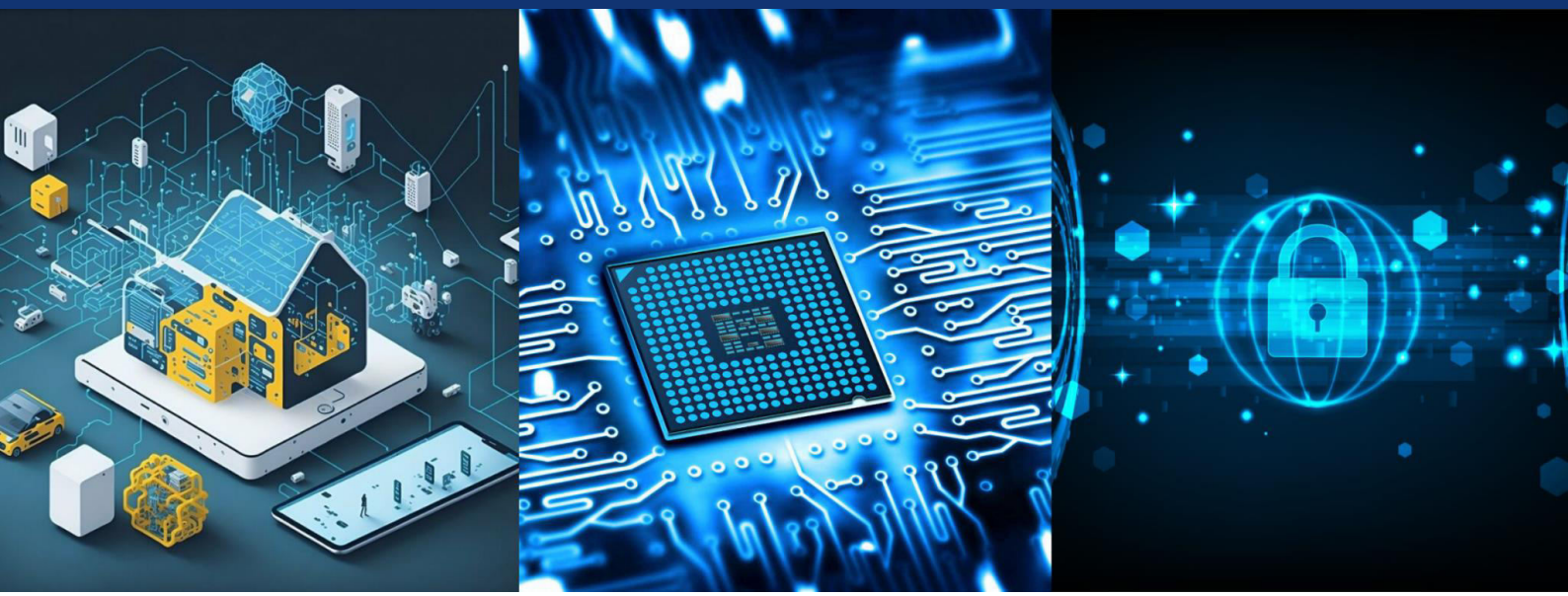
IJIRCCE

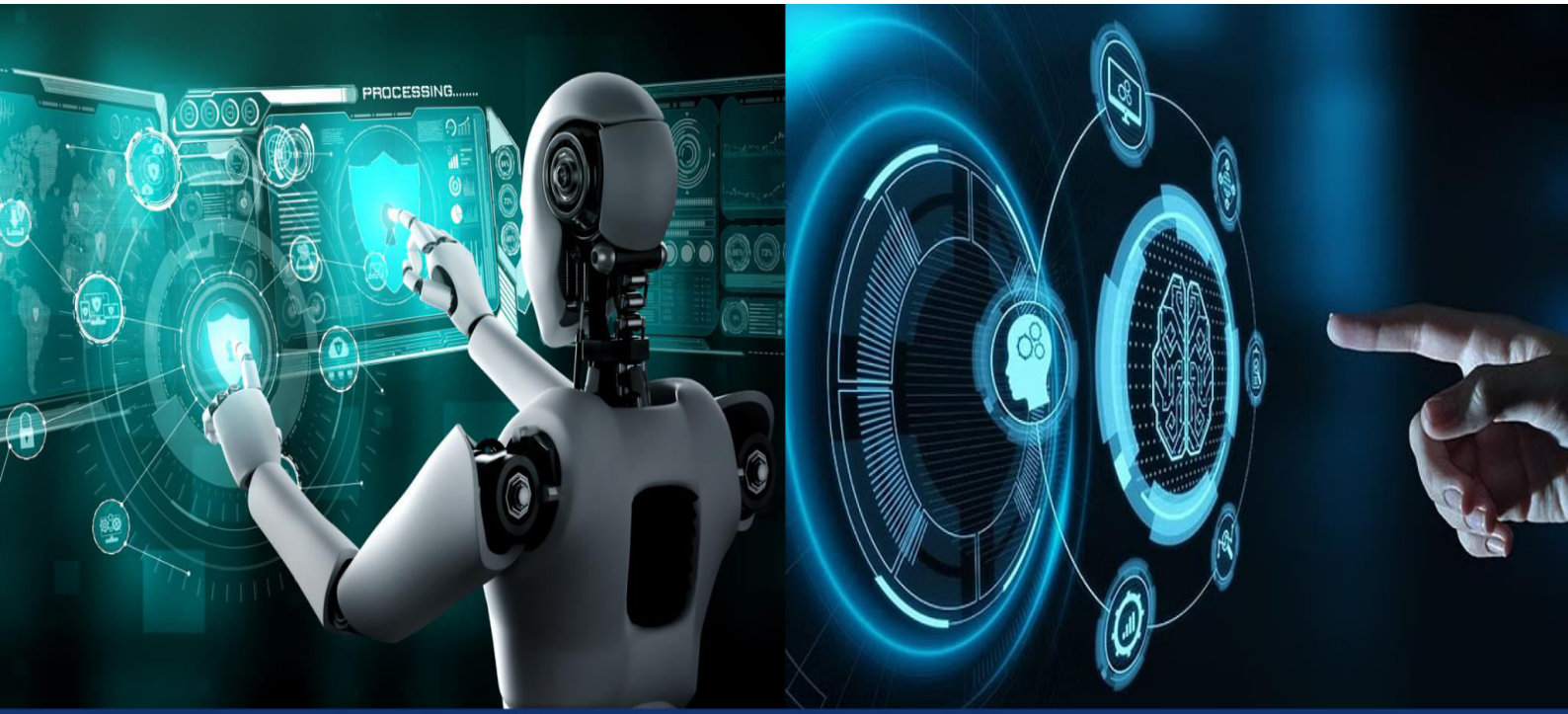
ISSN(O): 2320-9801

ISSN(P): 2320-9798



International Journal of Innovative Research in Computer and Communication Engineering

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)



Impact Factor: 9.927

Volume 14, Issue 10, October 2026

© 2026 IJIRCCE | Volume 14, Issue 10, October 2026 DOI: 10.15680/IJIRCCE.2026.1410037

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

Multi-Tenant Retrieval-Augmented Generation Using a Shared Pinecone Vector Index

Aaditi Choramle, Chitrang Choudhari, Pratik Kamble, Sahil Dhanavade, Dr. Swati D. Ghule

MCA Student, Department of Master of Computer Application, P.E.S.'s Modern College of Engineering, Pune, Maharashtra, India

Professor and Project Guide, Department of Master of Computer Application, P.E.S.'s Modern College of Engineering, Pune, Maharashtra, India

ABSTRACT: Deploying retrieval-augmented generation (RAG) as a multi-tenant product requires durable vector storage, tenant isolation, and a simple embedding path. This paper presents a Pinecone-centred architecture in which all chatbots share one serverless index. Isolation is enforced by tagging every vector with a chatbot identifier and applying an equality metadata filter on every query, while Pinecone hosted inference (multilingual-e5-large) generates embeddings. The design is realised in Relay, a Next.js platform with a dashboard and an embeddable widget. A functional evaluation on a deployed instance supports it as a blueprint needing no per-tenant indexes or Python embedding sidecar.

KEYWORDS: Pinecone, vector database, retrieval-augmented generation, multi-tenancy, semantic search

I. INTRODUCTION

Large language models generate fluent text from parametric memory learned during pre-training [1]. That memory is a poor fit for organisational question answering. Private policy manuals and product guides are not in the training corpus, they change without a retraining cycle, and an unconstrained model may invent plausible facts—a behaviour surveyed as hallucination [2]. Retrieval-augmented generation (RAG) couples a sequence generator with a non-parametric store of passages [3]. A question is embedded, similar chunks are retrieved, and the generator is prompted to answer using only that evidence. Dense retrieval in a shared embedding space is the usual first stage [4], [5].

The engineering problem is where those embeddings live. Research prototypes often keep vectors in process memory with FAISS [6]. A hosted product that serves many organisations cannot do so: vectors must survive deploys, queries must be filtered by tenant, and the service should scale without attaching a GPU worker to every web request. Vector database management systems have emerged to provide approximate nearest-neighbour search with metadata predicates [7]. Pinecone is a fully managed, serverless member of that class. It stores dense vectors, supports server-side metadata filters, and offers hosted inference so that the same vendor both embeds text and indexes the resulting vectors [8].

A. Problem Statement

Consider several chatbot owners who upload disjoint document collections and expect that a question asked of Chatbot A never returns text belonging to Chatbot B. A naive shared index violates this requirement: nearest-neighbour search returns the globally closest vectors. Creating one physical index per chatbot is operationally heavy and expensive on a managed platform. A second problem is the split-brain architecture common in student and start-up RAG stacks, in which a Python embedder and a Node.js frontend disagree on model name or dimension. The question studied here is whether Pinecone's shared index, metadata filters, and hosted inference are sufficient to implement logical multi-tenancy and a single-language retrieval path.

B. Objectives and Contributions

The objectives are (i) to isolate tenants on one Pinecone index, (ii) to embed documents and queries with the same hosted model, and (iii) to expose retrieval through both an authenticated owner dashboard and a public embeddable widget. The contributions of this paper are as follows.

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

1) A tenancy model in which Pinecone metadata field chatbot_id is the isolation key, and every query issued by the application includes an equality filter on that field.
2) An ingestion and query pipeline that uses Pinecone hosted inference (multilingual-e5-large) with inputType passage for documents and inputType query for questions, eliminating a sidecar embedding process.
3) A dual authentication facade—Auth.js sessions for owners and a public pk_live_ API key bound to CORS origins for widgets—sharing a single Pinecone query function.
4) A deployed case study, Relay, which instantiates the architecture as a Next.js service with MongoDB for configuration and OpenRouter for answer synthesis, demonstrating that the Pinecone design is operational rather than merely schematic.

The remainder of the paper surveys the literature (Section II), describes the research methodology and states the hypotheses (Section III), reports the results, discussion, and limitations (Section IV), and closes with the conclusion and future scope (Section V).

II. LITERATURE SURVEY

A. Retrieval-Augmented Generation

Lewis et al. introduced RAG as a generator conditioned on passages retrieved from a dense index of Wikipedia [3]. Fusion-in-Decoder concatenates many retrieved passages in a seq2seq encoder [9]. Gao et al. survey RAG for LLMs and distinguish naive, advanced, and modular pipelines [10]. Self-RAG adds reflection tokens that decide when to retrieve [11]. The system in this paper follows the naive/advanced pattern: recursive chunking and overlap before indexing, top-k cosine retrieval at query time, and a constrained generation prompt. It does not train a retriever or a critic; the contribution is the multi-tenant Pinecone substrate, not a new neural ranker.

B. Dense Retrieval and Embeddings

Dense Passage Retrieval encodes questions and passages into one vector space [4]. Sentence-BERT made semantically comparable sentence embeddings practical [5]. Wang et al. released multilingual E5 models; multilingual-e5-large has 24 layers and 1024-dimensional outputs and is trained with distinct passage and query prefixes [12]. Using the same model with mismatched input types degrades retrieval. Pinecone hosted inference exposes those input types as parameters, which this work exploits.

C. Vector Databases

Johnson et al. demonstrated billion-scale similarity search on GPUs with FAISS [6]. Subsequent systems add durability, filtering, and distribution. Pan et al. survey vector database management systems and their query predicates [7]. Open-source stores such as Qdrant, Weaviate, Chroma, and PostgreSQL with pgvector offer similar k-NN APIs with different operational models. Pinecone differs for our purposes in two respects that matter to a serverless web application: indexes are fully managed, and embeddings can be produced by the same API that writes vectors [8]. Table I summarises this comparison. Commercial RAG builders (Chatbase, CustomGPT, and similar products) validate market demand but do not disclose index-level isolation mechanics.

D. Multi-Tenant Software

Krebs, Momm, and Kounev discuss isolation and customisation in multi-tenant SaaS [13]. Three database patterns are common: a database per tenant, a schema per tenant, and a shared schema with a tenant attribute. This work applies the third pattern to a vector index. The risk is well known: omitting the tenant predicate leaks data. Therefore the Pinecone filter is not left to callers of a generic search helper; it is inside the only function that may query the index.

E. Research Gap

Tutorial RAG stacks, including LangChain notebooks, typically index one folder of files and omit tenancy [14]. Papers on RAG quality rarely specify how a SaaS should share one managed index. Papers on multi-tenancy rarely treat vector metadata filters as the isolation mechanism. The gap addressed here is the combination: a documented, implemented, and deployed Pinecone shared-index design for document chatbots, including hosted embeddings and a public widget that must not gain write access to the index.

© 2026 IJIRCCE | Volume 14, Issue 10, October 2026 DOI: 10.15680/IJIRCCE.2026.1410037

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

TABLE I
COMPARISON OF VECTOR STORES FOR A MULTI-TENANT RAG SERVICE

System	Managed	Metadata filter	Hosted embed	Fit in this work
FAISS	No	Application	No	Prototype only
Chroma	Optional	Limited	No	Local demos
pgvector	Via RDBMS	SQL WHERE	No	Relational shops
Qdrant / Weaviate	Optional	Yes	No / modules	Self-hosted prod.
Pinecone (used)	Yes	Yes (server-side)	Yes (inference)	Chosen substrate

III. RESEARCH METHODOLOGY

A. System Model

Fig. 1 shows the system. Owners authenticate with Google through Auth.js and manage chatbots stored in MongoDB [15], [16]. Website visitors never receive a user session; they present a per-chatbot public API key. Both paths invoke one retrieval procedure that talks to Pinecone and then to an LLM router [17]. Pinecone is therefore on the critical path of every answer: if the filter is wrong, tenancy fails; if inference is wrong, retrieval quality collapses.

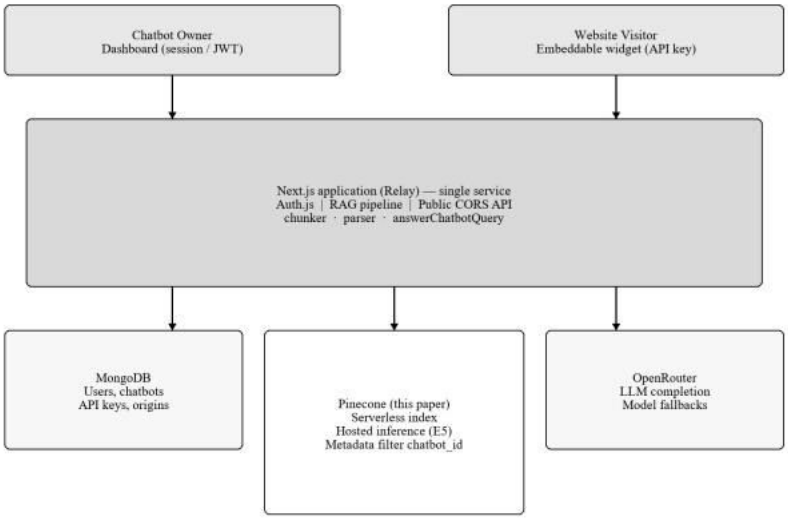


Fig. 1. System architecture. Pinecone stores vectors and produces embeddings.

Let C be the set of chatbots and V the set of stored vectors. Each vector v_i carries metadata m_i including chatbot_id, document_id, document_name, chunk_id, and the raw chunk text. Define the tenant slice $V_c = \{ v_i \in V \mid m_i.\text{chatbot_id} = c \}$. The isolation invariant required of the implementation is that a query issued in the name of chatbot c may retrieve only from V_c.

B. Pinecone Index Design

A single serverless index is provisioned with cosine similarity and a hosted embedding model multilingual-e5-large [8], [12]. The index dimensionality is taken from the model (1024) and is not set by hand, which removes a class of production failures. All chatbots share one namespace. Physical isolation by namespace or by index-per-tenant is possible in Pinecone but was rejected: it multiplies operational objects and still requires the application to choose the correct target. Logical isolation with a mandatory filter concentrates the correctness argument in one predicate, illustrated in Fig. 2.

© 2026 IJIRCCE | Volume 14, Issue 10, October 2026 DOI: 10.15680/IJIRCCE.2026.1410037

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

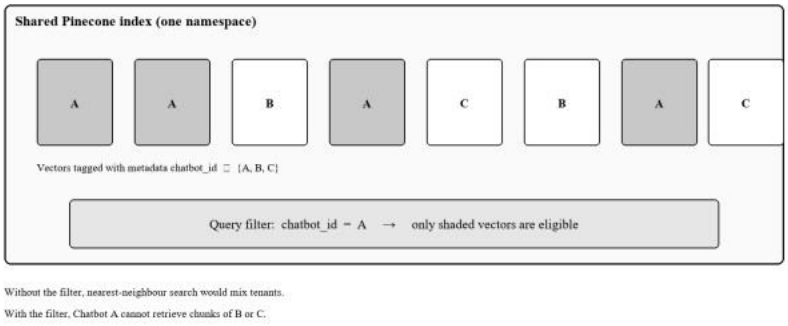


Fig. 2. Shared index with logical isolation. Only vectors whose chatbot_id matches the query filter are searchable.

C. Embedding and Similarity

Let $E_{passage}$ and E_{query} denote Pinecone hosted inference with inputType set to passage and query respectively. For a chunk text t the stored embedding is

$$v = E_{passage}(t) \in \mathbb{R}^{1024} \quad (1)$$

For a user question q the query embedding is

$$v_q = E_{query}(q) \in \mathbb{R}^{1024} \quad (2)$$

Pinecone ranks candidates by cosine similarity

$$s(v_q, v) = (v_q \cdot v) / (\|v_q\| \|v\|) \quad (3)$$

Given chatbot c and integer k (default 5, capped at 10 on the public API), the retrieved set is

$$R(q, c) = \text{TopK}_{v \in V_c} \; s(v_q, v) \quad (4)$$

Equation (4) encodes the isolation invariant: the feasible set is V_c, not V. In the Pinecone SDK this is expressed as a query filter { chatbot_id: { $eq: c } } together with includeMetadata so that chunk text can be passed to the LLM. If R(q, c) is empty, generation is skipped and a fixed refusal is returned. Otherwise the LLM is instructed to use only the retrieved texts and to cite document_name values [17].

D. Ingestion Algorithm

Fig. 3 (left) summarises writes. After session and ownership checks, the file is parsed (PDF.js via unpdf, DOCX via mammoth, or UTF-8 text). Recursive character splitting uses separator order paragraph, line, sentence, space, with chunk size 800 characters and overlap 150, following standard RAG pre-processing practice [10]. Each chunk is labelled with chatbot_id and a new document_id. Pinecone inference accepts at most 96 strings per request; the implementation batches accordingly. Upserts are issued in batches of 100 records into the shared namespace. The LLM is not called during ingestion, so write load does not consume generation quota.

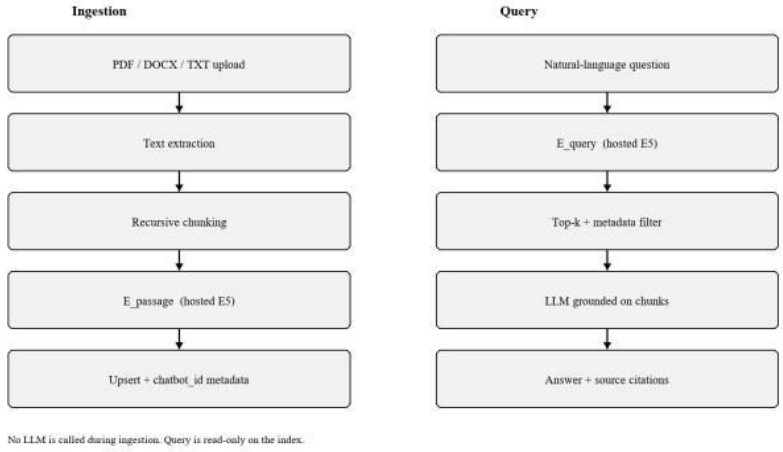


Fig. 3. Write path (left) versus read path (right). Both embedding calls use Pinecone hosted inference.

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

E. Query Algorithm

Fig. 3 (right) summarises reads. The caller is either an owner with a JWT and a matching userId, or a widget with a valid API key and an allowed Origin [18]. After authorisation, Equations (2)–(4) are evaluated. Retrieved texts are concatenated as numbered documents and sent to the generator at low temperature. Free-tier LLM endpoints may be unavailable or may leak chain-of-thought; the implementation tries a fallback model list and, if needed, returns a quoted excerpt of the top Pinecone matches so that retrieval remains useful even when generation fails. Dashboard search and public chat call the same function, which prevents the two facades from drifting onto different filters.

F. Security Properties of the Vector Layer

The public key is designed as a chat credential, not as a secret with write scope. It identifies the chatbot for Pinecone filtering but is not accepted by document-upload routes. Regenerating the key invalidates embedded widgets. Default CORS is permissive for first-run demonstration and is intended to be locked to a customer origin in production [18]. Deleting a chatbot currently removes the MongoDB document; orphan vectors may remain in Pinecone. They stay unreachable to other tenants because of (4), but they occupy storage. Delete-by-metadata-filter is listed as future work rather than claimed as implemented.

G. Hypotheses

The problem in Section I-A is tested through three hypotheses. They are stated before the results so that the evaluation can be read against them. They concern tenancy, the embedding path, and access control. They are not claims about ranking scores on a public benchmark.

H1 (Isolation). If every stored vector is tagged with chatbot_id and every Pinecone query includes an equality filter on that field, then a query for chatbot c returns only vectors belonging to c. A separate physical index per chatbot is not required.

H2 (Single embedding path). If documents and questions are embedded by Pinecone hosted inference with multilingual-e5-large, using inputType passage and inputType query respectively, then the service does not need a separate embedding process, and the vector dimension matches the index.

H3 (Shared read path). If the owner dashboard and the public widget call the same query function, and the public key is accepted only by chat routes, then both surfaces retrieve from Pinecone while the widget cannot write to the index.

IV. RESULTS AND DISCUSSION

A. Implementation and Setup

The architecture was implemented as a TypeScript Next.js 15 application with React 19, Mongoose on MongoDB, Auth.js v5 (Google OAuth, JWT sessions), the official Pinecone SDK, and Jest tests for chunking, models, and chatbot APIs [8], [15], [16], [19]. The service is deployed at https://relayy-dun.vercel.app. An npm package, relay-chatwidget, loads appearance from a public config endpoint and posts questions to the public chat endpoint. No local GPU is used; embedding and search run in Pinecone, and generation runs on OpenRouter using Llama-family instruction models [17], [20]. This setup is the experimental platform.

B. Functional and Isolation Evaluation

Evaluation is architectural and functional rather than a labelled IR benchmark; constructing a public multi-tenant RAG test set is left to future work. Table II records the behaviours that were checked on the deployed system. Fig. 4 shows the owner dashboard with several independent chatbots on one account, which is the visible form of tenancy above the vector layer.

© 2026 IJIRCCE | Volume 14, Issue 10, October 2026 DOI: 10.15680/IJIRCCE.2026.1410037

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

TABLE II

FUNCTIONAL AND ISOLATION OUTCOMES ON THE DEPLOYED PINECONE-BACKED SYSTEM

Scenario	Expected	Observed
Paraphrased question about uploaded skills text	Dense hit on E5 + grounded answer	Structured skill list returned in dashboard chat
Question with no supporting chunk	Refusal, no invented policy	Fixed “could not find” path or honest limit
Query chatbot A about B’s documents	Empty / non-B context	Filter in (4) excludes B
Widget key used to upload files	No write route	Public API is chat/config only
Missing API key or disallowed origin	401 / 403	Public auth helper enforces both
LLM rate-limit or leaked reasoning	Still show retrieved text	Fallback to Pinecone chunk excerpt

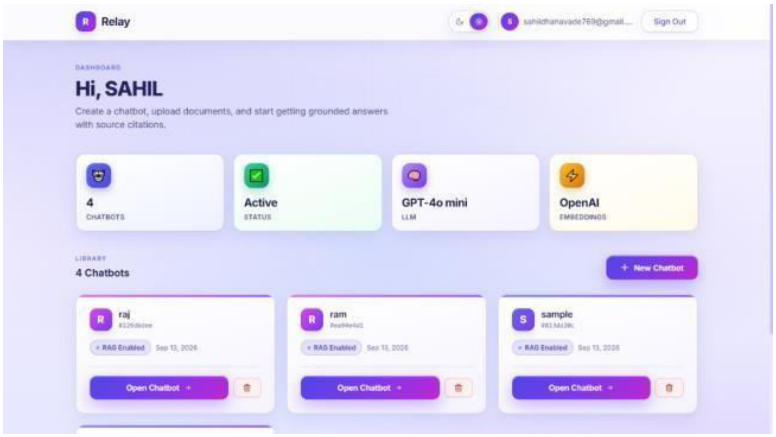


Fig. 4. Deployed owner dashboard: multiple chatbots share one Pinecone index and remain separately addressable.

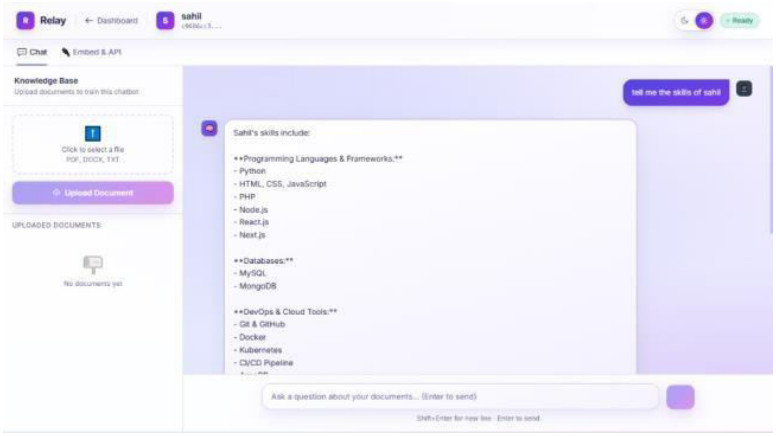


Fig. 5. Dashboard question answering after Pinecone retrieval (example: skills of Sahil).

C. Comparison with Alternative Pinecone Deployments

Table III contrasts three ways of using Pinecone (or a local ANN library) for the same product shape. A FAISS process per instance fails durability and tenancy. One Pinecone index per chatbot maximises isolation at the cost of index sprawl and slower provisioning. The chosen design—one index, hosted inference, mandatory metadata filter—

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

minimises moving parts while keeping isolation in a single code path. Hosted inference also removes the failure mode in which a Python sidecar embeds with 768 dimensions while the index expects 1024.

TABLE III
ALTERNATIVE RETRIEVAL BACK ENDS FOR THE SAME CHATBOT PRODUCT

Design	Isolation	Ops burden	Embed mismatch risk
In-process FAISS	Manual	High (stateful)	High
Pinecone, index per chatbot	Physical	High (N indexes)	Medium if local embed
Pinecone shared + filter (this work)	Logical, Eq. (4)	One index	Low (hosted E5)

D. Discussion

The results support three claims. First, Pinecone metadata filters are an adequate isolation mechanism for a teaching-scale and small-SaaS RAG product, provided the filter is not optional in application code. Second, hosted inference is a genuine simplification: the deployed artefact is one Node.js service. Third, a public widget can sit on Pinecone retrieval without being given upsert rights. The evaluation does not report nDCG or RAGAS scores; those metrics would measure chunking and prompt quality more than the choice of Pinecone, but they are the correct next experiment. A further limitation is orphan vectors after chatbot deletion, which is a Pinecone hygiene issue rather than an isolation leak.

E. Limitations

The evaluation is architectural and functional. It does not report nDCG or RAGAS, so it does not measure retrieval or answer quality on a labelled question set. Those metrics would speak more to chunking and prompting than to the choice of Pinecone, but their absence is a limit of the evidence.

Deleting a chatbot removes the MongoDB record. Orphan vectors may remain in Pinecone. They stay unreachable to other tenants because of (4), but they still occupy storage. The default origin list is permissive until an owner locks it. A public key placed in page script can be copied and used to query an allowed origin; it cannot upload documents. Answer quality still depends on how text is chunked and on the generator. The metadata filter does not repair a poor chunk.

V. CONCLUSION AND FUTURE SCOPE

A. Conclusion

This paper treated Pinecone not as an undifferentiated kNN API but as the isolation and embedding backbone of a multi-tenant RAG chatbot platform. A shared serverless index, cosine retrieval, multilingual-e5-large hosted inference, and a mandatory chatbot_id filter together satisfy the requirement that Chatbot A cannot read Chatbot B, without provisioning an index per tenant. Relay instantiates the design, with MongoDB holding configuration, OpenRouter synthesising answers, and an npm widget consuming a read-only public API. The three hypotheses are supported by the functional and isolation checks: the filter keeps tenants apart, hosted inference removes the embedding sidecar, and the public key does not gain write access. The architecture is offered as a reproducible pattern for teams who need a managed vector database under a multi-tenant document assistant.

B. Future Scope

The next steps follow directly from what this study did not implement. Delete-by-filter in Pinecone would remove vectors when a file or chatbot is deleted. Hybrid sparse–dense search would improve retrieval of identifiers, codes, and names. A cross-encoder reranker on the top-k list is the natural advanced-RAG extension [10]. Labelled evaluation with nDCG and RAGAS would measure chunking and answer quality, which this paper does not score. Per-key rate limits on the public query path would harden the widget API.

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

REFERENCES

[1] A. Vaswani et al., "Attention is all you need," in Proc. NeurIPS, 2017, pp. 5998–6008.
[2] Z. Ji et al., "Survey of hallucination in natural language generation," ACM Comput. Surv., vol. 55, no. 12, pp. 1–38, 2023.
[3] P. Lewis et al., "Retrieval-augmented generation for knowledge-intensive NLP tasks," in Proc. NeurIPS, vol. 33, 2020, pp. 9459–9474.
[4] V. Karpukhin et al., "Dense passage retrieval for open-domain question answering," in Proc. EMNLP, 2020, pp. 6769–6781.
[5] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence embeddings using Siamese BERT-networks," in Proc. EMNLP-IJCNLP, 2019, pp. 3982–3992.
[6] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with GPUs," IEEE Trans. Big Data, vol. 7, no. 3, pp. 535–547, 2019.
[7] J. J. Pan, J. Wang, and G. Li, "Survey of vector database management systems," VLDB J., vol. 33, pp. 1591–1615, 2024.
[8] Pinecone Systems, "Pinecone documentation: Indexes, metadata filtering, and inference," 2024. [Online]. Available: https://docs.pinecone.io
[9] G. Izacard and E. Grave, "Leveraging passage retrieval with generative models for open domain question answering," in Proc. EACL, 2021, pp. 874–880.
[10] Y. Gao et al., "Retrieval-augmented generation for large language models: A survey," arXiv:2312.10997, 2023.
[11] A. Asai, Z. Wu, Y. Wang, A. Sil, and H. Hajishirzi, "Self-RAG: Learning to retrieve, generate, and critique through self-reflection," in Proc. ICLR, 2024.
[12] L. Wang et al., "Multilingual E5 text embeddings: A technical report," arXiv:2402.05672, 2024.
[13] R. Krebs, C. Momm, and S. Kounev, "Architectural concerns in multi-tenant SaaS applications," in Proc. CLOSER, 2012, pp. 426–431.
[14] H. Chase, "LangChain documentation," 2022. [Online]. Available: https://python.langchain.com
[15] Auth.js, "Auth.js documentation (NextAuth.js v5)." [Online]. Available: https://authjs.dev
[16] MongoDB Inc., "MongoDB manual." [Online]. Available: https://www.mongodb.com/docs/manual/
[17] OpenRouter, "OpenRouter API reference." [Online]. Available: https://openrouter.ai/docs
[18] OWASP Foundation, "OWASP Top 10 – 2021." [Online]. Available: https://owasp.org/Top10/
[19] Vercel Inc., "Next.js documentation." [Online]. Available: https://nextjs.org/docs
[20] A. Grattafiori et al., "The Llama 3 herd of models," arXiv:2407.21783, 2024.



INTERNATIONAL STANDARD SERIAL NUMBER INDIA

INNO SPACE SJIF Scientific Journal Impact Factor

doi crossref

निस्केयर NISCAIR

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN COMPUTER & COMMUNICATION ENGINEERING

9940 572 462 6381 907 438 ijircce@gmail.com

www.ijircce.com

Scan to save the contact details