Design and Implementation of a RAG-Based AI Chatbot for Virtual Assistance in Real Estate Broker

Ripin1,\*

1Computer Science Department, BINUS Graduate Program - Master of Computer Science, Bina Nusantara University, Jakarta, Indonesia, 11480

Suryadiputra Liawatimena1,2

1Computer Science Department, BINUS Graduate Program - Master of Computer Science, Bina Nusantara University, Jakarta, Indonesia, 11480

2Automotive & Robotics Program, Computer Engineering Department, BINUS ASO School of Engineering, Bina Nusantara University, Jakarta, Indonesia 11480

\*Corresponding author. E-mail: [ripin@binus.ac.id](mailto:ripin@binus.ac.id)

DOI: 10.14416/j.asep.2025.xx.xxx

Received: DD MM 2025; Revised: DD MM 2025; Accepted: DD MM 2025; Published online: DD MM 2025

**Abstract**

Current real estate search is dominated by rigid, form-based filters, while buyers typically express fuzzy intents (e.g., “near landmark X,” “under IDR 1B,” “3 bedrooms”). We design a UI-decoupled, Retrieval-Augmented Generation (RAG) assistant exposed as a stateless HTTP/JSON response endpoint and demonstrated via a WhatsApp relay. The system implements three pipelines: Vector, MySQL/API (Text → JSON), and Hybrid Retrieval. We evaluate all pipelines on the same 30 gold-labeled questions with constraint-grounded metrics: Per-Constraint Accuracy (PCA), Strict Success, Constraint Pass Ratio CPR, a No-Result score for correct abstention, and question-level Precision/Recall/F1/Accuracy. We prioritize CPR over MMR because real estate users typically inspect only the top 3 to 5 results. The Hybrid Retrieval approach achieved 89.3% constraint accuracy, 86.7% question-level accuracy and 100% correct abstention rate, outperforming baseline methods by 11.6–13.0%.

**Keywords**: Retrieval-Augmented Generation, Chatbot, Real Estate, Semantic Search, Large Language Models

# Introduction

Most real estate systems still require users to pre-select rigid filters over SQL (location, price range, bedrooms, property type). This workflow works for narrow queries but falters when intents are fuzzy (“near a named landmark,” “walkable to transit”) and UIs evolve. Agents spend multiple rounds tweaking inputs, delaying answers and risking drop-off. We address this gap with a conversational assistant that captures natural-language intent yet preserves transactional correctness via live, structured data.

Large Language Models (LLMs) enable fluent, multi-turn chat but can drift from enterprise facts. Retrieval-Augmented Generation (RAG) mitigates this by pairing generation with targeted retrieval. In property search, we make two design choices: decouple the chat client from inference via a response API endpoint, and compare three retrieval pipelines—Vector, MySQL/API, and Hybrid (API → Vector)—on the same gold-labeled questions with constraint-grounded metrics.

This work designs and implements a domain-specific, channel-agnostic RAG assistant delivered as a response API endpoint that any chat client can call. Rather than building a bespoke chat UI, we expose a stateless HTTP/JSON endpoint that returns assistant replies (and optional diagnostics/metadata). To demonstrate multi-channel integration, we employ a WhatsApp (WA) forwarder that relays user messages to the endpoint and sends the assistant’s responses back to users—showing how the same backend can serve web, mobile, and social messaging channels without UI coupling.

Here, we introduce a channel-agnostic RAG assistant exposed as a response endpoint—decoupled from any chat UI and ready for multi-channel delivery (WhatsApp, web, mobile apps). The system implements a tri-path retrieval strategy—VectorDB, MySQL/API, and a Hybrid that blends both—to balance semantic flexibility with transactional correctness and data freshness [1],[2],[3],[4]. We also define a task-grounded evaluation protocol using constraint-level macro/micro metrics and a calibrated no-result score that rewards correct abstention, and we distill deployment guidance for running the assistant at scale with swappable API providers and vector stores—without changing client integrations.

The rest of the paper unfolds as follows: it begins with a review of related work and technical foundations, then describes the architecture, data preparation, and pipeline implementations. Next, it presents the experimental setup and results, combining quantitative analysis with qualitative observations. It concludes by discussing limitations and outlining directions for future work.

# Related Works

This section summarizes what past studies say about chatbots and search for real estate and similar domains. We move from how people adopt chatbots, to limits of early rule-based systems, to newer NLP/RAG methods, to evidence that AI tools can improve productivity, and finally to lessons from faceted search in HCI (Human-Computer Interaction). We end by stating where our work fits.

## *Chatbots in Real Estate and Conversational Recommendation*

Empirical studies in the real estate domain report that AI-enabled chatbots can improve perceived service quality and responsiveness, suggesting practical benefits for high-inquiry tasks such as property search [5]. Domain deployments also highlight compliance and fairness constraints (e.g., avoiding steering/redlining) as core design requirements for real estate assistants. Beyond real estate, surveys on conversational recommender systems (CRS) emphasize multi-turn preference elicitation and dialogue strategies that reduce interaction friction and capture buyer intent more effectively than static forms [6],[7].

## *Conversational IR and RAG Foundations*

Foundational work in conversational search formalizes properties of interactive IR in a chat setting and motivates dialogue-centric models that iteratively refine information needs [8]. The TREC CAsT track established reusable testbeds and methodology for conversational assistance, accelerating research on query rewriting, context carry-over, and ranking [9]. For grounding LLM outputs, Retrieval-Augmented Generation (RAG) combines parametric models with non-parametric memory and has shown state-of-the-art results on knowledge-intensive tasks [1]; complementary lines include retrieval-augmented pretraining (REALM) [10], dense retrievers such as DPR [3], and Fusion-in-Decoder readers that aggregate evidence across many retrieved passages [2]. Recent surveys consolidate this landscape and its trade-offs (e.g., recall vs. latency, provenance) [11].

## *From Faceted UIs to Conversational Retrieval*

Classic HCI and search-UI literature documents both the strengths and friction of faceted, form-based filtering for structured databases; while robust for well-specified queries, facets impose overhead when intents are vague or evolving [12]. Conversational retrieval offers an alternative front-end that elicits constraints naturally (e.g., price ceilings, bedroom count, landmark proximity) while a structured back-end enforces transactional correctness—an approach that aligns with our system’s separation of conversational intent capture and live API validation [8].

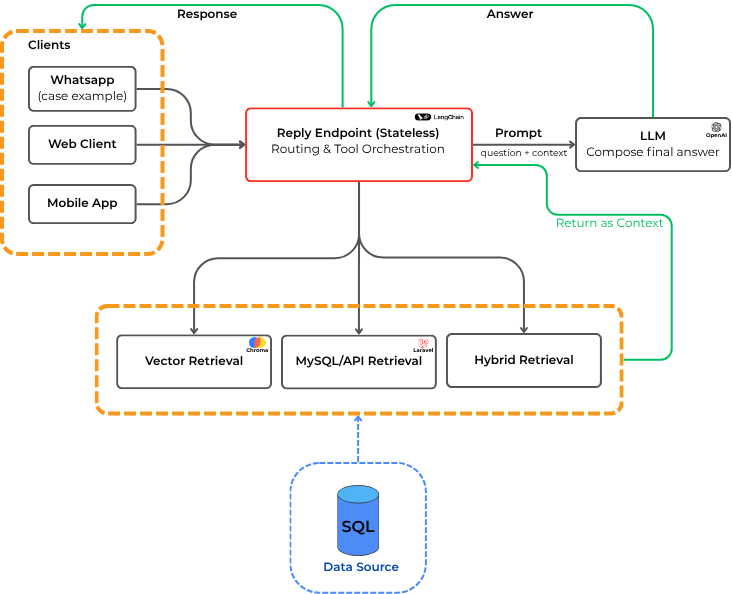
## *Retrieval Engines and Vector Indexes*

Efficient nearest-neighbor search underpins modern dense retrieval. FAISS delivers billion-scale similarity search on CPUs/GPUs via product quantization and fast k-selection [13], [14], and late-interaction rankers such as ColBERT preserve token-level matching while remaining index-friendly for large-scale deployments [4]. In engineering practice, lightweight vector stores like Chroma and orchestration frameworks like LangChain provide developer-facing abstractions for embedding storage, metadata filters, and tool-augmented prompting that ease prototyping and evaluation in applied RAG systems [15], [16].

## *Positioning and Gap*

Prior real estate chatbots typically bind to a single channel and retrieval mode—often text-only interfaces with either semantic matching or form-based filters—leaving limited evidence on head-to-head comparisons across pipelines using the same question set and constraint-grounded metrics [1], [2]. Building on conversational IR and RAG, our work explicitly compares Vector, MySQL/API, and Hybrid (API → Vector) pipelines on a shared, gold-labeled set and reports both constraint-level correctness and ranking-sensitive measures—reflecting the dual need for transactional accuracy and early relevant results in property search [1], [8], [9], [11], [12].

# Research Methodology



**Figure 1:** High Level Architecture

This chapter explains how we built and tested a channel-agnostic property-search assistant using Retrieval-Augmented Generation (RAG). The assistant is exposed as a simple HTTP/JSON response endpoint, so any chat client can call it. Figure 1 explain our example front end, a WhatsApp (WA) forwarder sends user messages to the endpoint and relays the assistant’s reply to the user. Our method uses two kinds of data—live, structured listings from a SQL database and a vector index of short text snippets—and compares three retrieval paths on the same set of questions. We measure how well each path meets the user’s constraints and how well it abstains when there is truly no match.

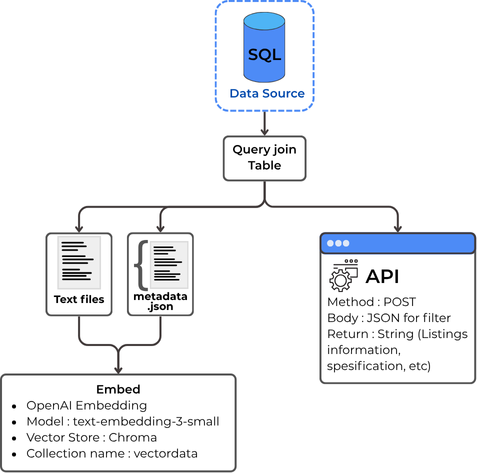
## *System Overview and Architecture*

When a message arrives, the reply endpoint invokes a thin controller implemented with LangChain. The controller exposes the same interface to two data substrates (i) a live MySQL database accessed via an HTTP API (our sample uses Laravel, but any stack is acceptable because the endpoint is JSON-in/JSON-out) and (ii) a Chroma vector database containing normalized listing snippets with embeddings. For experimentation, the controller is run in one of three fixed configurations per trial: VectorDB-only, MySQL/API-only, or Hybrid (MySQL → Vector). That is, the system does not auto-select at runtime; instead, each pipeline is executed separately on the same questions to enable fair, like-for-like comparison. In every configuration, the controller aggregates the chosen source’s results and prompts the language model to produce a concise, grounded answer; if constraints are not satisfied, it returns a “no result” response rather than guessing [17]. The endpoint is stateless and UI-agnostic; the WhatsApp forwarder acts purely as a transport layer.

## *Data and Preparation*

The transactional data in MySQL is the “source of truth.” It contains listing\_id, price, bedrooms, bathrooms, location tokens, property type (sale/rent), and availability. We expose a single POST endpoint that accepts a JSON filter (e.g., {"keyword":"ringroad","price\_max":800000000,"bedrooms":3,"type":"sale"}) and returns only rows that match the constraints. Because this API queries live SQL joins, results are always fresh and require no offline preparation beyond schema design and indexing.

For the vector substrate, we prepare a semantic index offline. We run SQL joins to assemble each listing’s key fields and normalize them into a short text snippet (title, salient attributes, location tokens, and a brief description). For traceability, we write one .txt file per listing (≈1700 files in our snapshot) and generate a single metadata.json that records each file’s path and structured attributes (e.g., price, bedrooms, type, normalized area). We then create embeddings for every snippet using OpenAI’s text-embedding-3-small and persist them into a Chroma collection named vectordata. This index is refreshed on a schedule (e.g., nightly) or on change events from the listings pipeline.



**Figure 2:** Data Preparation workflow

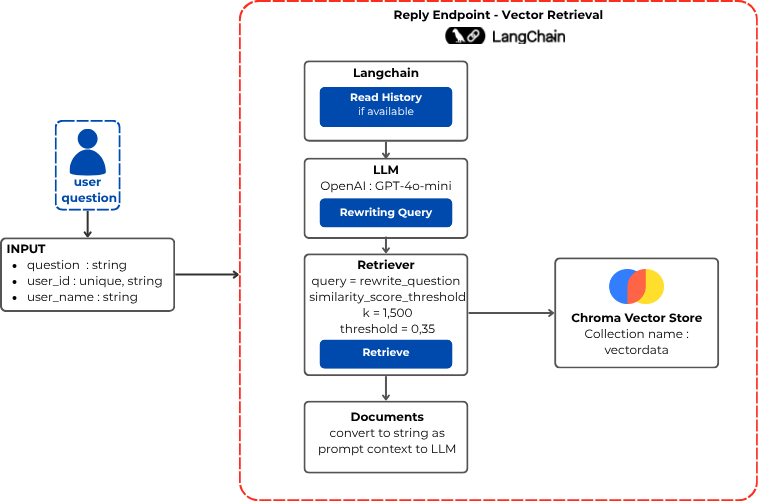
Figure 2 explains the data-preparation workflow: the left branch shows MySQL → join queries → per-listing TXT files → metadata.json → Embed → Chroma collection /vectordata; the right branch shows the API path that executes the same joins at query time (no precompute) and benefits from a scheduled refresh loop that keeps the vector index aligned with the live database. This setup ensures semantic search over stable snippets while price and availability remain authoritative in the API.

We test with 30 gold-labeled questions that reflect typical buyer or agent needs, such as a price limit, a rough location, and number of bedrooms. Each question has a “gold” profile that states the expected constraints. We store all of this in a CSV/XLSX file with a fixed schema and a version tag to keep it clean and repeatable.

**Table 1 :** Example rows from the gold dataset

|  |  |
| --- | --- |
| **Question** | **Gold Constraint** |
| Find a house to sale in Cemara Area | {"keyword":"cemara",  "listing-type":1,  "property-type": 1} |
| I’m looking for a house at Cemara price below IDR 1Billion | {"keyword":"cemara",  "listing-type":1,  "property-type": 1,  "max\_price": 1000\_000\_000\_000} |

## *Inference Pipelines*

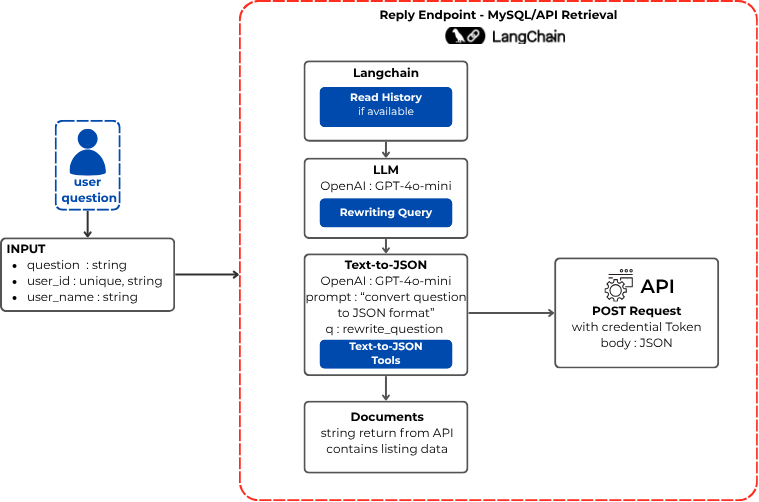


**Figure 3a:** Vector Retrieval Workflow

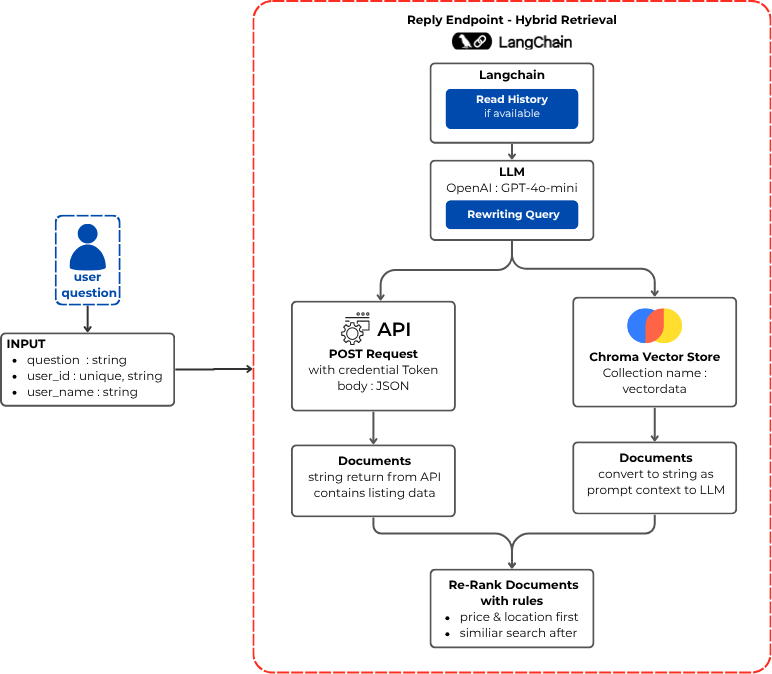
We compare three retrieval paths and employ GPT-4o-mini at four points : (i) constraint extraction & routing/tool selection, (ii) JSON-filter synthesis for the API, (iii) optional query rewriting for vector search, and (iv) answer composition with guardrails.

**Vector Retrieval**, The LLM first extracts soft constraints and (optionally) rewrites the retrieval query. The system then queries the vector store using dense retrieval with OpenAI’s text-embedding-3-small. We use the similarity\_score\_threshold search type with k=1500 and score threshold = 0.35. When we are confident about a filter like price, we can also apply it as a metadata filter inside the vector store. The model then writes a grounded answer based on the snippets. If nothing fits, it returns “no result” as explained in Figure 3a.

**MySQL/API Retrieval**. The LLM first converts the user text into a JSON filter that captures hard constraints (price limit, bedrooms, property type, a location token) [18]. It then calls the live API and gets exact matches from SQL. The answer uses only the fields returned by the API. If there are no rows, the system returns “no result” and may suggest a small change, such as a slightly wider budget as explained in Figure 3b.



**Figure 3b:** MySQL/API Retrieval Workflow



**Figure 3c:** Hybrid Retrieval Workflow

**Hybrid Retrieval** (MySQL→Vector) explained in Figure 3c. At the reply endpoint, a LangChain controller first reads chat history (if available) and GPT-4o-mini rewrites the user query into a cleaner intent. From this intent, the system queries two sources in parallel:

(1) the API via an authenticated POST with a JSON body capturing hard constraints (price, bedrooms, type, location token), returning documents (stringified listing rows); and

(2) the Chroma Vector Store (collection vectordata) to retrieve documents used as prompt context for the LLM (semantic expansion).

The returned documents are then passed to a rule-based re-ranker that prioritizes price & location conformity first (authoritative from the API) and orders ties by semantic similarity afterward. The LLM composes the final answer from the re-ranked set; if no candidate satisfies the constraints, it returns “no result.”

## *Experimental Setup*

We run all three pipelines on the same 30 questions and use identical limits for each run (for example, max\_items = 5). Retrieval and API parameters are kept fixed to enable fair comparison. For every question, we record the tools invoked, execution time, and token usage. The code base provides three runners test\_vector\_rval, test\_api\_rval, and test\_hybrid\_rval saves per-question outputs as well as summary file for evaluation.

## *Implementation Details*

### **3.5.1 Hardware**

All experiments were run on a virtual private server (Hostinger KVM 2 plan) configured with 2 vCPU cores, 8 GB RAM, 100 GB NVMe SSD storage, and 8 TB monthly bandwidth. The vector database and the RAG API backend were deployed on the same VPS during development and evaluation, and all computations were performed on CPU only (no dedicated GPU).

### **3.5.2 Software Versions**

The system was implemented using Python 3.12.9. Key dependencies included LangChain v1.0.5 for orchestration and tool integration, Chroma v1.3.4 for vector storage and retrieval, and OpenAI API (gpt-4o-mini, version 2024-07-18) for language model inference. The embedding model used was text-embedding-3-small (OpenAI, dimension 1536). The backend API was built with Laravel 12 (PHP 8.2) and MySQL 8.0.

### **3.5.3 Hyperparameters**

For all LLM calls, we set temperature to 0 to ensure deterministic outputs across runs. Maximum token limits were configured as follows: 500 tokens for constraint extraction and JSON generation, 800 tokens for answer composition, and 150 tokens for routing decisions. Vector retrieval used k equals 1500 candidates with a similarity score threshold of 0.35 (cosine similarity). The embedding dimension was fixed at 1536 as determined by the OpenAI model.

## *Metrics and Evaluation*

This study evaluates each chatbot reply against a gold “ground truth” prepared per question. A reply can fall into one of two paths: (1) **a listing reply** (the bot returns one or more property items) or (2**) a no-result reply** (the bot explicitly says “no data / nothing found”). We treat the live MySQL API as the arbiter of truth for availability, prices, and attributes. All metrics are computed per question and then aggregated. Table 2 explains Metric and the definition.

**Table 2 :** Metric definitions

|  |  |
| --- | --- |
| **Metric** | **Definition** |
| PCA(listing) | Share of satified constraints for one listing |
| Strict Success | All constraints satisfied for that listing |
| CPR | Fraction of strictly correct listings in the reply |
| No Result Score | Correct abstention |
| CM Precision/Recall/F1/Acc | Question level quality vs API truth |

### **3.6.1 Constraint checks per item**

Each question has a set of gold constraints (e.g., keyword, max\_price, bedrooms ≥ k, building\_areas ≥ L, listing\_type, etc.). For every returned listing 𝑖, we extract the same attributes from the answer text (with parsers for price, bedrooms, building\_areas, etc.). When a listing carries a link to an internal listing\_id, we fetch the authoritative fields from the API and override the extracted values for evaluation (so price/availability are always judged by the API). Let C be the subset of gold constraints that apply to the question (non-empty, non-null). For listing *i,* define a per-constraint indicator [19], [20]:

**Per-Constraint Accuracy (PCA) for listing *i* :**

**Strict Success (binary) for listing *i* :**

### **3.6.2 Answer-level correctness (CPR)**

If a reply returns *K* listings (*i* = 1..*K*), we summarize correctness with Constraint-Pass Rate :

**CPR**

We declare the prediction for the question as Positive (i.e “ this answer is correct”) when : CPR ≥ *T*, where *T* is tunable threshold (by default *T =* 0.60). Intuitively, with *K* = 5 this means at least 3 of 5 listings must be strictly correct.

### **3.6.3 Handling no result replies**

If the bot claims “no result”, we verified the claim by querying the API with the gold constraints (1) If the API indicates no items for the gold filter, we score the reply as correct abstention. (2) if the API indicates item exists, we score the reply as missed opportunity [17].

We tract a No-Result Score per such question :

### **3.6.4 Confusion matrix over questions**

To summarize question-level outcomes, we compare the system prediction (driven by CPR and “no result” logic) against the ground truth from the API. Ground Truth (GT) is Positive when API has items for the gold constraints, Negative when the API is empty. Prediction (Pred) is Positive when the bot returned listings and CPR ≥*T*, Negative if the bot explicity returned “no result” an the API check was conclusive.

This yields the standard four cases :

* TP : GT+, Pred+ (API has items, bot meet threshold)
* FN : GT+, Pred- (API has items, bot abstains or fails threshold)
* TN : GT-, Pred- (API empty, bot says “no result”)
* FP : GT-, Pred+ (API empty, but bot returns listings that pass threshold)

From these we report :

## *LangChain Flow and Safety Rules*

The controller uses three tools behind the scenes: calling the MySQL API with a JSON filter, retrieving from the vector store with the user text (and optional filters), and re-ranking within a set of candidate IDs. A simple routing prompt sends strict numeric constraints to the API first. The vector tool helps when the location is vague or when many results need better ranking. If the data are missing or the constraints conflict, the system prefers to abstain.

The key safety rule is simple: price and availability must come from the API. Vector snippets can add helpful context (for example, neighborhood text) but cannot override the API’s facts. If any required field is missing, the system asks for clarification or returns “no result.”

## *Reproducibility*

We keep all settings in configuration files: base URLs, keys, timeouts, Chroma collection names, the embedding model, and LLM parameters. Each run records the runner version, a hash of the prompts, the vector snapshot ID, and a per-request log of tool calls, timings, and token counts. The evaluator writes a per-row CSV and a summary with the macro/micro metrics and the No-Result Score. We use temperature 0 (or fixed seeds) so results are stable.

## *Limitations*

Our test uses 30 questions and one market dataset. Results may change with larger datasets, new regions, or different embedding models. The API in our sample uses Laravel, but the method is stack-agnostic. We also log latency and tokens, but business outcomes like lead conversion require field trials and A/B tests across channels. Those are outside the scope of this study and left for future work.

# Results and Discussion

## *Results*

Table 3 summarizes the per question averages for three pipelines: **Vector Retrieval, MySQL/API Retrieval,** and **Hybrid Retrieval.**

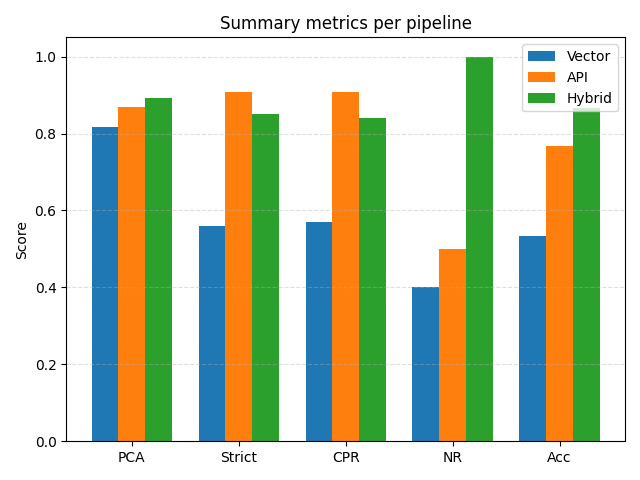
**Table 3 :** Summary of evaluation results

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Metric** | **Vector** | **MySQL/ API** | **Hybrid** | **∆ vs Best** |
| **PCA** | 0.681 | 0.800 | 0.893 | 11.63% |
| **Strict Success** | 0.570 | 0.967 | 0.862 | -10.86% |
| **CPR** | 0.476 | 0.909 | 0.841 | -7.48% |
| **No Result Score** | 0.400 | 0.500 | 1.000 | 100% |
| **Precision** | 0.823 | 0.950 | 0.920 | -3.15% |
| **Recall** | 0.560 | 0.760 | 0.920 | 21.05% |
| **F1** | 0.667 | 0.844 | 0.920 | 9.00% |
| **Accuracy** | 0.533 | 0.767 | 0.867 | 13.04% |

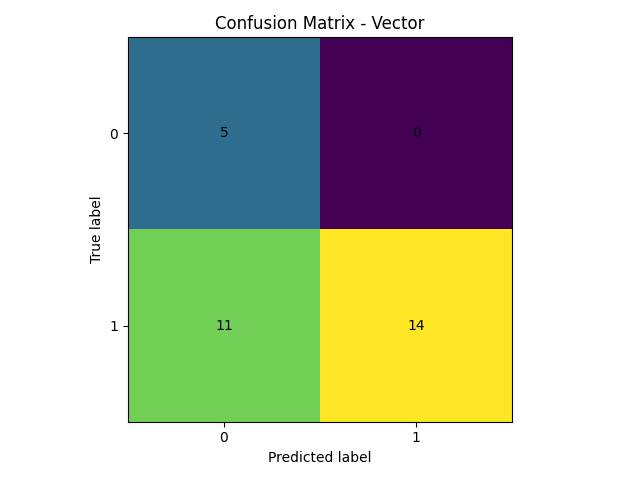
Overall, Hybrid offers the strongest end-to-end performance, even though the gains are not uniform across all metrics. On the listing side, Hybrid achieves the highest PCA listing (0.893 vs 0.800 for API and 0.681 for Vector), corresponding to an +11.63% relative improvement over the best baseline. Together with the perfect No Result Score (1.000, +100% vs API), this indicates that when Hybrid decides to show listings, more of them fully satisfy all gold constraints, and when it abstains it almost always does so correctly. Strict Success and CPR remain slightly below the API baseline (0.862 vs 0.967 and 0.841 vs 0.909), showing that the price of expanding recall on fuzzy queries is a small drop in very strict, all-constraints-must-match cases.

At the question level (confusion-matrix metrics), Hybrid clearly dominates. Accuracy improves from 0.533 (Vector) and 0.767 (API) to 0.867 for Hybrid (+62.5% vs Vector and +13.0% vs API). Recall jumps from 0.560 (Vector) and 0.760 (API) to 0.920, and F1 from 0.667 / 0.844 to 0.920, showing that Hybrid recovers substantially more gold-correct answers. Precision for Hybrid (0.920) is slightly lower than API (0.950) but higher than Vector (0.823), meaning Hybrid keeps over-answering under control while still expanding coverage. In statistical terms, Hybrid shows improvement trends over the API baseline on Accuracy (p = 0.26, 95% CI [−0.08, 0.28]) and significantly outperforms the Vector baseline (p = 0.005, 95% CI [0.11, 0.56]).

These quantitative trends are also clearly reflected in the visualizations: Figure 4(a) summarizes the main metrics per pipeline, while Figures 4(b), 4(c) and 4(d) show the confusion-matrix patterns for each pipeline, illustrating how Hybrid reduces both false negatives and false positives compared to the Vector and API baselines.



**Figure 4a:** Summary Metrics per Pipeline



**Figure 4b:** Confusion Metrix – Vector

A diagram of a confused matrix

Description automatically generated

**Figure 4c:** Confusion Metrix – MySQL/API

A chart of different colored squares

Description automatically generated

**Figure 4d:** Confusion Metrix – Hybrid

## *Discussion*

Vector excels at fuzzy intent—synonyms, spelling variants, and informal location names—so it can surface semantically similar items when users provide vague or partial descriptions. Its weaknesses are staleness and limited trust in numeric and availability fields when judged only from snippets, which is reflected in lower PCA, Strict Success, and Accuracy. API (Text→JSON) is strongest for transactional correctness (price, bedrooms, area, listing type) because it queries the live database; it achieves very high Strict Success and CPR when the user’s constraints are explicit and structured, but it can miss relevant items when intent is underspecified or phrased informally, which hurts Recall.

Hybrid (API→Vector) combines both advantages. Numeric and availability checks still depend on the API as the source of truth, while the vector layer expands recall and helps re-rank candidates when location or keywords are fuzzy. This design raises PCA and No Result Score (better constrained results and safer abstention when the API is empty), while confusion-matrix metrics show higher Recall, F1, and Accuracy with only a slight trade-off in Precision compared to API. In short, Hybrid preserves the transactional reliability of the API and widens the semantic coverage of Vector, which in practice translates into higher PCA, better handling of no-result cases, and substantially better Accuracy than either baseline.

# Conclusion & Future Works

## *Conclusion*

This work presented a channel-agnostic RAG assistant for property search with three retrieval pipelines: VectorDB, API (Text→JSON), and a Hybrid (API→Vector) policy that treats the live API as the source of truth for transactional fields while using vectors to expand and re-rank candidates. We proposed a practical evaluation protocol centered on six outcome metrics—PCAlisting, Strict success, CPR, No-result score, and CM Precision / Recall / F1 / Accuracy—that directly capture both listing-level constraint satisfaction and question-level correctness (including correct abstention).

Across the results, Hybrid achieved the best overall performance: higher PCAlisting and Strict success (more fully correct listings), improved CPR (cleaner top results), stronger No-result score (safer abstention when the API is empty), and better confusion-matrix scores (balanced precision and recall). These outcomes support the core design choice: keep prices and availability grounded in the API, and use vector search to recover from fuzzy intent and vocabulary variation.

## *Limitations*

This study has several methodological and practical limitations.

* Gold set size and labeling quality. The evaluation uses only 30 questions, which limits statistical power. We will expand to 100–200 domain questions and report inter-annotator agreement (Cohen’s κ) for constraint labels.
* Statistical significance and uncertainty. Current results are point estimates without uncertainty bounds. We will add nonparametric bootstrap 95% confidence intervals and McNemar tests on question-level outcomes (Hybrid vs. baselines).
* Metric choice and threshold sensitivity. The CPR decision rule uses a fixed T = 0.60, which may be sub-optimal across query types. We will sweep T ∈ {0.50, 0.60, 0.70} and complement constraint-grounded metrics with MRR/MAP/NDCG@k to capture ranking quality and early precision.
* External baselines. We do not compare against traditional faceted search (direct API filters) or LLM-only generation, nor against commercial systems. Adding these baselines will contextualize gains beyond our three pipelines.
* User study and qualitative feedback. We emphasize offline correctness and do not measure perceived usefulness or trust. A user study with SUS and time-to-first-relevant (plus short interviews) is left for future work.
* User study and qualitative feedback. We emphasize offline correctness and do not measure perceived usefulness or trust. A user study with SUS and time-to-first-relevant (plus short interviews) is left for future work.
* Index freshness and consistency. The vector index can lag behind the live database, temporarily reducing recall even though final numerics come from the API. Nightly/on-change refresh mitigates but does not eliminate staleness; its impact under high churn is not quantified.ses. The dataset and labels reflect a specific market and inventory; selection and phrasing biases may limit generalization to other locales.

## *Future Works*

First, we will develop dynamic routing and adaptive thresholding, learning a policy that selects Vector, API, or Hybrid per query type and calibrates the CPR decision threshold T on the fly (e.g., looser for fuzzy-location intents, tighter for numeric-heavy filters). This includes multi-turn behaviors (clarification, disambiguation, recovery after no-result) and graded no-result scoring to capture near-miss cases.

Second, we will pursue cost–latency optimization—caching frequent API filters, prompt compression, and early-exit strategies—while quantifying index freshness (lag vs. recall) and benchmarking vector stores under larger indices. The goal is to keep response times low without sacrificing CPR, F1, or early-precision metrics (MRR/MAP/NDCG).

Third, we will add broker-in-the-loop tooling—lightweight controls for agents to approve/flag listings and provide quick reasons—so feedback can train parsers/rankers via active learning. We will also run a user study (SUS and time-to-first-relevant) to measure perceived usefulness and trust.

Finally, we will expand the schema and data sources (facilities, parking, facing, renovation status; POIs, transit, travel times) and the evaluation protocol: scale the gold set to 100–200 questions with inter-annotator agreement, add external baselines (faceted search, LLM-only, commercial where feasible), and report bootstrap confidence intervals and McNemar tests for statistical significance.

**Acknowledgments**

I am deeply grateful to my advisor and thesis committee at BINUS University for their guidance, constructive feedback, and steady encouragement throughout this research. Their questions sharpened the methodology and helped clarify the evaluation design. Special thanks go to everyone who helped curate listings, maintain the live API, and support the data collection process. Their efforts ensured reliable ground truth and made the hybrid evaluation possible. Finally, I owe my deepest appreciation to my family and friends for their patience and support. Their encouragement sustained me through long nights of experimentation, writing, and revision.

**Author Contributions**

R.L.: conceptualization, methodology, software, investigation, data curation, formal analysis, visualization, writing—original draft, writing—reviewing and editing, project administration;  
S.L.: supervision, advisory support, methodological guidance for evaluation, reviewing and editing.

All authors have read and agreed to the published version of the manuscript.

**Conflicts of Interest**

The author declares no conflict of interest. The research design, data collection, analysis, and reporting were conducted independently and without any commercial or financial relationships that could be construed as a potential conflict.

Funding disclosure. This work received no specific grant from any funding agency, commercial or not-for-profit sectors. All compute and infrastructure costs were covered by the author’s organization for routine operations and did not influence the study design or outcomes.

Data and tooling. The live API and listing data used for evaluation were operated by the author’s organization solely to provide factual ground truth; operational access did not confer any undue advantage or constraints on the reported results.

**Data and Code Availability**

The complete implementation, including the reply endpoint, evaluation scripts, and pipeline configurations, is available in a public GitHub repository at : [https://github.com/ ripinlibinus/ RagChatBot](https://github.com/ripinlibinus/RagChatBot). The repository includes setup instructions, environment specifications, and example configuration files.

The 30-question gold-labeled evaluation set, along with constraint annotations and API verification scripts, is provided in the repository under the test\_question.py file. Sample property listings (1,537 anonymized records) and their pre-processed, embed-ready representations are included in the data/embeds directory. In addition, the repository provides the corresponding MySQL schema and seed data, which can be imported directly via Laravel migrations to reconstruct the database used in our experiments.

# References

[1] P. Lewis *et al.*, “Retrieval-augmented generation for knowledge-intensive NLP tasks,” *Adv. Neural Inf. Process. Syst.*, vol. 2020-December, 2020.

[2] G. Izacard and E. Grave, “Leveraging passage retrieval with generative models for open domain question answering,” *EACL 2021 - 16th Conf. Eur. Chapter Assoc. Comput. Linguist. Proc. Conf.*, pp. 874–880, 2021, doi: 10.18653/v1/2021.eacl-main.74.

[3] V. Karpukhin *et al.*, “Dense passage retrieval for open-domain question answering,” *EMNLP 2020 - 2020 Conf. Empir. Methods Nat. Lang. Process. Proc. Conf.*, pp. 6769–6781, 2020, doi: 10.18653/v1/2020.emnlp-main.550.

[4] O. Khattab and M. Zaharia, “ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT,” *SIGIR 2020 - Proc. 43rd Int. ACM SIGIR Conf. Res. Dev. Inf. Retr.*, pp. 39–48, Jul. 2020, doi: 10.1145/3397271.3401075.

[5] A. Sao, D. Pathak, G. Vijh, S. Saxena, and A. Deogaonkar, “Analyzing the Impact of AI-Enabled Chatbot on Service Quality in the Real Estate Sector: An Empirical Study in NCR,” in *Procedia Computer Science*, Elsevier B.V., 2025, pp. 1198–1207. doi: 10.1016/j.procs.2025.04.075.

[6] D. Jannach, A. Manzoor, W. Cai, and L. Chen, “A Survey on Conversational Recommender Systems,” *ACM Comput. Surv.*, vol. 54, no. 5, Jun. 2022, doi: 10.1145/3453154.

[7] C. Gao, W. Lei, X. He, M. de Rijke, and T. S. Chua, “Advances and challenges in conversational recommender systems: A survey,” *AI Open*, vol. 2, pp. 100–126, Jan. 2021, doi: 10.1016/j.aiopen.2021.06.002.

[8] F. Radlinski and N. Craswell, “A theoretical framework for conversational search,” *CHIIR 2017 - Proc. 2017 Conf. Hum. Inf. Interact. Retr.*, pp. 117–126, Mar. 2017, doi: 10.1145/3020165.3020183;WGROUP:STRING:ACM.

[9] J. Dalton, C. Xiong, and J. Callan, “CAsT 2020: The Conversational Assistance Track Overview,” 2021, Accessed: Nov. 06, 2025. [Online]. Available: https://huggingface.co/

[10] K. Guu, K. Lee, Z. Tung, P. Pasupat, and M.-W. Chang, “REALM: Retrieval-Augmented Language Model Pre-Training,” 2020, doi: 10.5555/3524938.3525306.

[11] W. Fan *et al.*, “A Survey on RAG Meeting LLMs: Towards Retrieval-Augmented Large Language Models,” *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, pp. 6491–6501, 2024, doi: 10.1145/3637528.3671470.

[12] M. A. Hearst, “Search User Interfaces-Search User Interfaces Marti A. Hearst Frontmatter More information”, Accessed: Nov. 06, 2025. [Online]. Available: www.cambridge.org

[13] “Welcome to Faiss Documentation — Faiss documentation.” Accessed: Nov. 06, 2025. [Online]. Available: https://faiss.ai/index.html

[14] H. Jégou, M. Douze, and C. Schmid, “Product quantization for nearest neighbor search,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 1, pp. 117–128, 2011, doi: 10.1109/TPAMI.2010.57.

[15] “Getting Started - Chroma Docs.” Accessed: Nov. 06, 2025. [Online]. Available: https://docs.trychroma.com/docs/overview/getting-started

[16] “Home - Docs by LangChain.” Accessed: Nov. 06, 2025. [Online]. Available: https://docs.langchain.com/

[17] Y. Geifman and R. El-Yaniv, “Selective Classification for Deep Neural Networks,” *Adv. Neural Inf. Process. Syst.*, vol. 30, 2017.

[18] C. Tapsai, P. Meesad, and C. Haruechaiyasak, “Natural language interface to database for data retrieval and processing,” *Appl. Sci. Eng. Prog.*, vol. 14, no. 3, pp. 435–446, 2021, doi: 10.14416/j.asep.2020.05.003.

[19] S. Robertson and H. Zaragoza, “The Probabilistic Relevance Framework: BM25 and Beyond,” *Found. Trends® Inf. Retr.*, vol. 3, no. 4, pp. 333–389, Dec. 2009, doi: 10.1561/1500000019.

[20] H. Valizadegan, R. Jin, R. Zhang, and J. Mao, “Learning to Rank by Optimizing NDCG Measure,” *Adv. Neural Inf. Process. Syst.*, vol. 22, 2009.