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

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

Suryadiputra Liawatimena[1,2]
[1]Computer Science Department, BINUS Graduate Program - Master of Computer Science, Bina Nusantara University, Jakarta, Indonesia, 11480
[2]Automotive & Robotics Program, Computer Engineering Department, BINUS ASO School of Engineering, Bina Nusantara University, Jakarta, Indonesia 11480

*Corresponding author. E-mail: 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
Real estate platforms still force users through rigid dropdown filters, yet most buyers think in fuzzy terms like "somewhere near shopping centre," "budget around 1 billion," or "at least 3 bedrooms." To bridge this gap, we built a Retrieval-Augmented Generation (RAG) chatbot exposed as a stateless HTTP/JSON API endpoint, with a simple web interface for testing and demonstration. We compared three retrieval strategies: Vector-only using ChromaDB, API-only with structured MySQL filters, and a Hybrid approach combining both. Using 30 gold-labelled questions, we measured Per-Constraint Accuracy (PCA), Constraint Pass Ratio (CPR), Strict Success Ratio, and question-level Precision, Recall, F1, and Accuracy derived from confusion matrix analysis. The Hybrid method outperformed both baselines, achieving 97.6% CPR, 96.6% strict success, and 100% question-level accuracy compared to API-only (73.3% accuracy) and Vector-only (56.7% accuracy). Hybrid also achieved perfect correct abstention (2/2 true negatives) when no matching properties exist. Our findings suggest that pairing structured database queries with semantic search handles both exact constraints and vague user intent effectively.

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

1 Introduction

Most real estate systems still require users to pre-select rigid filters such as location, price range, bedrooms, and property type. This workflow performs well for narrow, well-defined queries but struggles when user intents are fuzzy, for example "near a named landmark," "budget around 1 billion," or "minimum 3 bedrooms" [1]. Real estate agents often spend multiple rounds tweaking filter inputs, delaying answers and risking user drop-off [2]. We address this gap with a conversational assistant that captures natural-language intent while preserving transactional correctness through live, structured data.

Large Language Models (LLMs) enable fluent, multi-turn conversations but can drift from enterprise facts or hallucinate nonexistent listings. Retrieval-Augmented Generation (RAG) mitigates this problem by pairing text generation with targeted retrieval from authoritative data sources [3]. Dense passage retrieval methods [4] and late-interaction models [5] have demonstrated strong performance for open-domain question answering, while frameworks such as Fusion-in-Decoder aggregate evidence across multiple retrieved passages [6]. Recent surveys consolidate this landscape and highlight trade-offs between recall, latency, and provenance [7]. In property search, we make two key design choices: first, decouple the chat interface from inference by exposing a stateless HTTP/JSON API endpoint; second, compare three retrieval pipelines on the same gold-labeled questions using constraint-grounded metrics.

Conversational search formalizes properties of interactive information retrieval in a chat setting,

motivating dialogue-centric models that iteratively refine user information needs [8]. Surveys on conversational recommender systems emphasize multi-turn preference elicitation and dialogue strategies that reduce interaction friction compared to static filter forms [9][10]. Our work applies these principles to real estate by designing a domain-specific RAG assistant delivered as a response API endpoint that any front-end client can consume. Rather than building a dedicated chat UI, we expose a stateless endpoint that returns assistant replies along with optional diagnostics and metadata. A simple web interface accompanies the API for testing and demonstration purposes.

The system implements three retrieval strategies: Vector-only using semantic search with ChromaDB [11] , API-only using structured MySQL queries following text-to-JSON conversion [12], and Hybrid combining API filtering with semantic re-ranking. The Hybrid approach treats the live database API as the source of truth for transactional fields like price and availability, while using vector embeddings to expand recall for fuzzy location names and informal queries [3][4]. We orchestrate these pipelines using LangChain [13] with tool-augmented prompting. We evaluate all three pipelines using a constraint-based protocol with Per-Constraint Accuracy (PCA), Constraint Pass Ratio (CPR), Strict Success Ratio, and question-level Precision, Recall, F1, and Accuracy derived from confusion matrix analysis.

The rest of the paper is organized as follows. Section 2 reviews related work on chatbots, conversational IR, and RAG foundations. Section 3 describes the system architecture, data preparation, pipeline implementations, experimental setup, and evaluation metrics. Section 4 reports results and discussion. Section 5 concludes the paper with limitations and future work.

2 Related Works

This section reviews prior research relevant to our RAG-based property search assistant, covering chatbot adoption, conversational IR, RAG foundations, retrieval approaches, and evaluation methods.

2.1 *Chatbots Conversational Recommendation*

Empirical studies report that AI-enabled chatbots can improve perceived service quality in real estate, suggesting practical benefits for high-inquiry tasks such as property search [2]. Surveys on conversational recommender systems (CRS) emphasize multi-turn preference elicitation and dialogue strategies that reduce interaction friction compared to static filter forms [9] [10].These findings motivate natural-language interfaces that handle fuzzy, evolving user preferences while maintaining factual correctness.

2.2 *Conversational IR and RAG Foundations*

Foundational work in conversational search formalizes interactive IR properties and motivates dialogue-centric models that iteratively refine information needs [8]. The TREC CAsT track established methodology for conversational assistance research [14]. Retrieval-Augmented Generation (RAG) addresses LLM hallucination by grounding outputs in retrieved content [3]. Related approaches include REALM for retrieval-augmented pretraining [15], DPR for dense passage retrieval [4], Fusion-in-Decoder for evidence aggregation [6], and ColBERT for efficient late-interaction ranking [5]. Recent surveys discuss trade-offs between retrieval recall, latency, and provenance [7].

2.3 *Faceted Search and Its Limitations*

Classic HCI literature documents both strengths and friction of faceted filtering [1]. While robust for well-specified queries, facets impose overhead when intents are vague. A user searching for "a house near the university for my child" must manually decompose this into location, type, and price filters. Conversational retrieval elicits constraints naturally while structured back-ends enforce correctness [8].

2.4 *Hybrid Retrieval and Vector Search*

Efficient nearest-neighbor search underpins dense retrieval, with FAISS providing billion-scale similarity search [16][17]. Lightweight vector stores like ChromaDB and orchestration frameworks like LangChain ease RAG prototyping [11][13]. Recent work shows that combining sparse lexical methods

with dense semantic methods outperforms either alone [18][19][20]. Small embedding models with LLM re-ranking can match larger models, suggesting retriever-reader alignment matters more than embedding size [19].

2.5 *Natural Language to Database*

Converting natural language to structured queries has been studied extensively [12]. Our API-only pipeline generates JSON filter objects rather than raw SQL, reducing injection risks while maintaining interpretability.

2.6 *RAG Evaluation*

Evaluating RAG systems requires assessing both retrieval and generation quality. Retrieval metrics include Precision, Recall, MRR, and NDCG [21][22]. For task-oriented systems, constraint satisfaction measures whether results match user requirements. Selective classification principles suggest systems should abstain when confidence is low [23]. Our evaluation adopts constraint-based metrics combined with confusion matrix analysis.

2.7 *Research Gap*

Prior real estate chatbots typically use single retrieval modes with limited systematic comparisons. Our work addresses this by: (1) designing a channel-agnostic RAG assistant with stateless API architecture, (2) comparing three retrieval strategies on identical gold-labeled questions, (3) proposing constraint-based evaluation measuring both listing-level and question-level correctness, and (4) demonstrating that Hybrid retrieval achieves superior performance.

3 Research Methodology

This chapter describes how we built and evaluated the property search assistant. Figure 1 shows the high-level architecture.

3.1 *System Architecture*

The system consists of three main layers: a ReAct agent for reasoning and tool selection, a tool layer with six specialized functions for property search, and data sources including a live API and a vector database. Figure 1 illustrates the overall architecture.

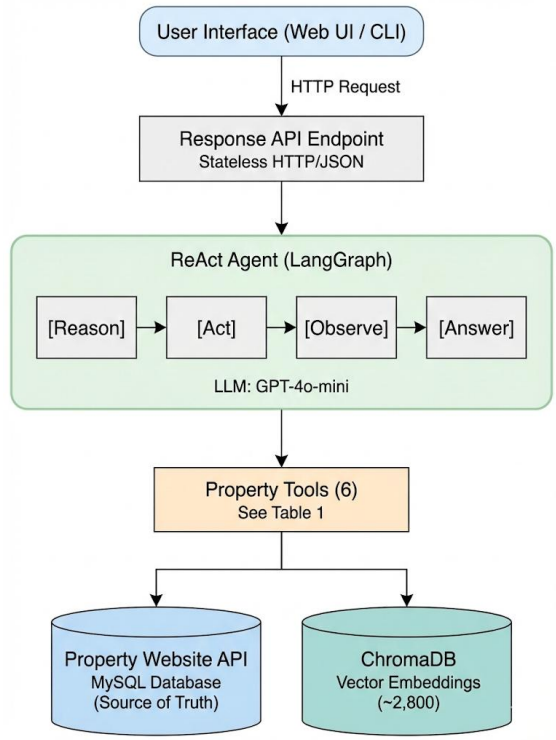

Figure 1: High Level Architecture

ReAct Agent. We implement a ReAct (Reasoning and Acting) agent using LangGraph that follows a reason-act-observe loop [13]. Given a user query, the LLM first reasons about what information is needed, selects appropriate tools, observes the results, and iterates until it can formulate a final answer. This architecture allows the agent to handle multi-step queries and combine results from multiple tools.

Tool Layer. The agent has access to six property tools for searching, retrieving, and locating listings. Table 1 describes each tool and its function.

Data Sources. The system queries two data sources: a live MySQL database via REST API serving as the source of truth for transactional data, and a ChromaDB vector store for semantic search and re-ranking. Table 2 provides details of each data source.

Table 1 : Agent Tools

Tool Name	Description
search_properties	Search listings with structured filters (location, price, bedrooms, type)
get_property_detail	Retrieve full property details by ID
get_property_by_number	Get details by result number (1-10) from previous search
geocode_location	Convert location names to geographic coordinates
search_nearby	Radius-based search from given coordinates
search_pois	Discover nearby points of interest (schools, malls, hospitals)

Table 2 : Data Sources

Data Sources	Description
Property Website API (MYSQL via REST)	Live property listings (~2,800 records), source of truth for price, availability, and attributes
ChromaDB (Vector Store)	Property embeddings using text-embedding-3-small (1536 dimensions) for semantic search and re-ranking

3.2 *Data Preparation*

Figure 2 illustrates the data preparation workflow, which consists of two parallel paths: offline vector index preparation and online API query processing.

Transactional Data. The MySQL database serves as the source of truth, containing property ID, price, bedrooms, bathrooms, building area, land area, location, property type, listing type, and availability status. We expose a POST endpoint accepting JSON filters and returning matching rows with exact constraint satisfaction.

Vector Index. For semantic search, we prepare embeddings offline. We extract each listing's key fields and normalize them into text snippets containing title, attributes, location, and description. Using OpenAI's text-embedding-3-small model (1536 dimensions), we generate embeddings and store them in ChromaDB. The index contains approximately 2,800 property listings and is refreshed periodically to stay synchronized with the live database.

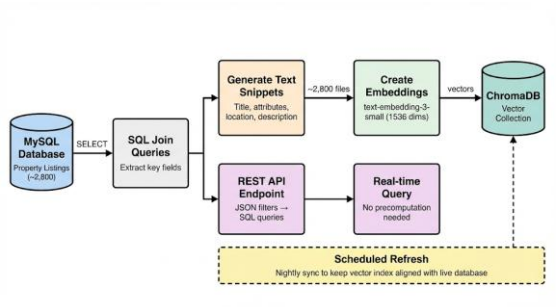

Figure 2: Data Preparation workflow

Gold Standard. We construct 30 gold-labeled questions reflecting typical buyer queries across several categories: simple location search, location with price constraints, location with specifications (bedrooms, floors), property type variations, context-dependent follow-ups, feature search, and nearby search. Each question includes expected constraints and expected result status (has_data or no_data). Table 3 shows example questions.

Table 3 : Gold Standard Question Categories

Category	Count	Example Question
location_simple	3	“carikan rumah dijual di daerah cemara”
location_price	3	“carikan rumah di cemara harga 1M an”
location_price_spec	3	"Rumah dijual ringroad dibawah 800jt 3 kamar"
property_type	3	"Apakah ada ruko disewakan di krakatau?"
context_followup	3	"Apakah masih ada pilihan lain?"
context_modify	2	"Pilihan lain, tapi yang 3 lantai?"
project_search	2	"Cari rumah di citraland bagya city"
feature_search	5	"Carikan rumah dengan fasilitas CCTV"
nearby_search	4	"Cari rumah dekat mall di medan"
no_data	2	"Apakah ada gudang di KIM?"
Total	**30**	

3.3 *Retrieval Pipelines*

The system implements three retrieval pipelines to handle diverse user queries. Each pipeline is optimized for different query types.

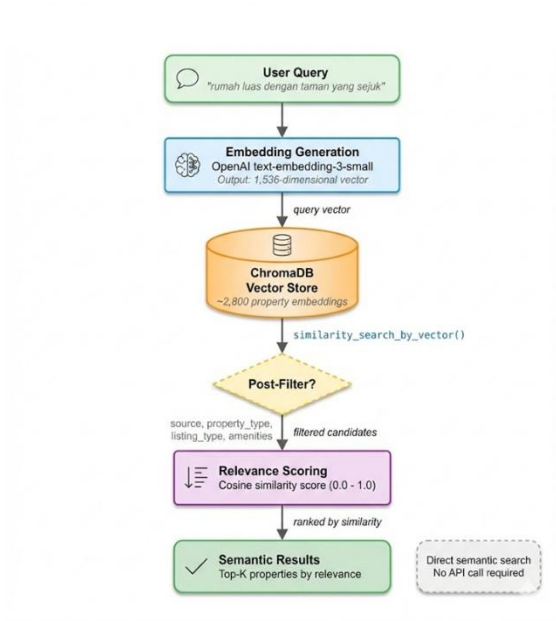

Figure 3a: Vector-Only Pipeline

The vector-only approach (Figure 3a) performs direct semantic search on ChromaDB without API calls. User queries are converted to embeddings using OpenAI's text-embedding-3-small model (1,536 dimensions), then matched against ~2,800 pre-indexed property vectors using cosine similarity. Optional post-filtering can be applied based on metadata fields (source, property_type, listing_type). This pipeline excels at understanding vague or descriptive queries such as "rumah luas dengan taman yang sejuk" (spacious house with a cool garden).

The API-only approach (Figure 3b) leverages structured database filtering. The LLM extracts search parameters from user queries (e.g., property type, price range, bedrooms, location) and constructs a JSON filter. This filter is sent to the MetaProperty REST API, which queries the MySQL database. This pipeline handles precise numerical queries effectively, such as "rumah 3 kamar harga 1 milyaran di Medan" (3-bedroom house around 1 billion in Medan).

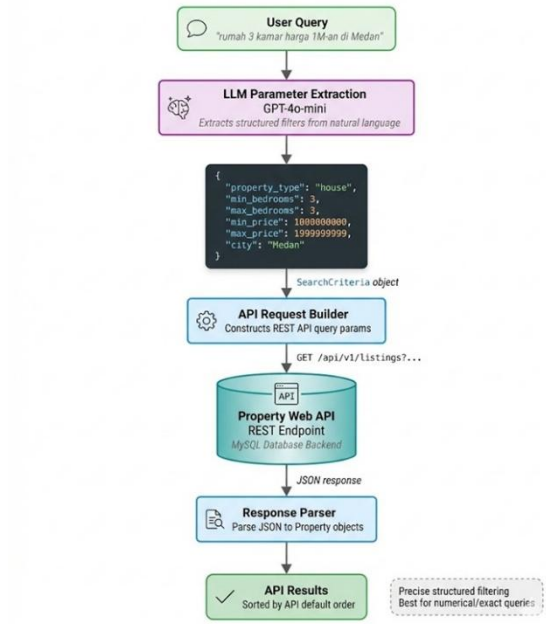

Figure 3b: API-Only Pipeline

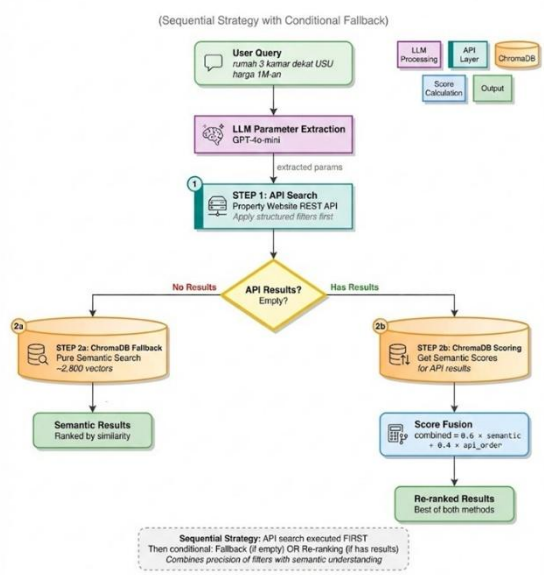

Figure 3c: Hybrid Pipeline (SEQUENTIAL with Fallback)

The hybrid approach (Figure 3c) combines both methods using a sequential strategy with fallback. The process follows these steps: (1) **API**

Search First: The system performs an API search with LLM-extracted parameters. (2) **Conditional Branching : -** If API returns no results, the system falls back to pure ChromaDB semantic search. - If API returns results, the system re-ranks them using ChromaDB semantic scores. (3) **Score Fusion :** For re-ranking, the combined score is calculated as: score = 0.6 × semantic_score + 0.4 × api_position_score. This sequential approach ensures that structured filters are applied first for precision, while semantic understanding provides either fallback coverage or relevance-based re-ranking.

3.4 *Experimental Setup*

All three retrieval pipelines were evaluated on the same 30 gold-labelled questions under identical conditions to ensure fair comparison. The evaluation followed a sequential conversation protocol where questions were processed in order, maintaining conversation context for follow-up queries.

Evaluation Protocol. Each pipeline processed the complete question set independently using the same maximum result limit (max_items = 10) and identical LLM parameters. The evaluation script executed queries sequentially, recorded all agent responses, extracted property listings from the output, and computed constraint-based metrics against the gold standard annotations.

Conversation Context. For context-dependent questions in the context_followup and context_modify categories, the system-maintained conversation history from previous turns within the same evaluation session. This design reflects real-world usage patterns where users iteratively refine their search criteria based on initial results. Each evaluation session used a unique thread_id to isolate conversation state.

Ground Truth Verification. For each property returned by the agent, we verified constraint satisfaction against the live MySQL database via the Property website API. This approach ensures that evaluation reflects actual data accuracy rather than potentially stale information from the vector index. The API serves as the authoritative source of truth for transactional fields including price, availability status, and property specifications.

3.5 *Implementation Details*

3.5.1 Hardware

The system was deployed on a cloud Virtual Private Server (VPS) with 4 vCPU cores, 8GB RAM, and 100GB SSD storage. Both the MySQL database (via Property Website API) and ChromaDB vector store were hosted on the same infrastructure to minimize network latency during hybrid retrieval operations.

3.5.2 Software Stack

The implementation uses Python 3.11 as the primary runtime environment. Table 4 lists the key software dependencies and their versions.

Table 4 : Software Dependencies

Component	Version	Purpose
LangChain	0.3.x	Agent orchestration
LangGraph	0.2.x	ReAct agent state management
ChromaDB	0.5.x	Vector storage
OpenAI API	-	LLM (GPT-4o-mini) & embeddings
FastAPI	0.110.x	HTTP/JSON endpoint
SQLAlchemy	2.0.x	Database ORM

3.5.3 Hyperparameters

Table 5 : System Hyperparameters

Parameter	Value	Description
LLM Model	GPT-4o-mini	Inference model
Temperature	0	Deterministic output
Embedding Model	text-embedding-3-small	1536 dimensions
Vector Search k	limit × 2	Pre-fetch for re-ranking
Similarity Threshold	0.35	Minimum cosine similarity
Max Results	10	Properties per query
CPR Threshold (T)	0.60	Query success threshold
Semantic Weight (α)	0.60	Hybrid score fusion
API Position Weight (β)	0.40	Hybrid score fusion

Table 5 summarizes the key hyperparameters used across all experiments.

The LLM temperature of 0 ensures reproducible results across evaluation runs. The CPR threshold T=0.60 means that with K=10 results, at least 6 of 10 listings must satisfy all applicable constraints for a query to be considered successful. The hybrid score fusion formula combines semantic relevance with API ordering: score = α × semantic_score + β × api_position_score.

3.6 *Metrics and Evaluation*

This study evaluates each chatbot reply against a gold standard prepared per question. A reply falls into one of two paths: (1) a listing replies where the bot returns one or more property items, or (2) a no-result reply where the bot explicitly states no matching properties exist. 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 6 : Metric definitions

Metric	Definition
PCA (Per-Constraint Accuracy)	Ratio of satisfied constraints for one listing
Strict Success	A listing where all constraints are satisfied (PCA=1)
Strict Success Ratio	Ratio of Strict Success listings across all results
CPR (Constraint Pass Ratio)	Fraction of Strict Success listings per query response
Mean CPR	Average CPR across all queries
Query Success Rate	Percentage of queries achieving CPR ≥ T
Confusion Matrix Metrics	Question-level Precision, Recall, F1, Accuracy

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 [21], [22]:

$$\mathbf{1}_{i,c} = \begin{cases} 1 & \text{if listing } i \text{ satifies constraint c} \\ 0 & \text{otherwise} \end{cases}$$

for each $c \in C$

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

$$\mathrm{PCA}_i = \frac{1}{|C|} \sum_{c \in C} 1_{i,c}$$

Strict Success (binary) for listing *i* :

$$\mathrm{Strict}_i = \begin{cases} 1 & \text{if } PCA_i = 1 \text{ (all constraints true)} \\ 0 & \text{otherwise} \end{cases}$$

3.6.2 Answer-level correctness (CPR)

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

CPR

$$\mathrm{CPR} = \frac{1}{K} \sum_{i=1}^{K} \mathrm{Strict}_i$$

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 [23].
We tract a No-Result Score per such question :

No Result Score

$$= \begin{cases} 1 & \text{if bot says no result and API also returns empty} \\ 0 & \text{otherwise} \end{cases}$$

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 $\geq T$, 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 :

$$\text{CM Precision} = \frac{TP}{TP + FP}$$

$$\text{CM Recall} = \frac{TP}{TP + FN}$$

$$\text{CM F1} = \frac{2.Prec.Rec}{Prec + Rec}$$

$$\text{CM Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

3.7 *Agent Flow and Safety Rules*

The system implements a ReAct (Reasoning and Acting) agent using LangGraph, which orchestrates tool selection through a reason-act-observe loop [13]. Given a user query, the LLM first reasons about required information, selects appropriate tools from the six available functions (Table 1), observes results, and iterates until formulating a final response. LangGraph manages the agent state and conversation flow through a directed graph structure, enabling complex multi-step interactions.

Tool Selection Strategy. The agent follows a routing strategy based on query characteristics:
(1) Queries with explicit numeric constraints (price, bedrooms) prioritize the API search tool. (2) Queries with vague locations or descriptive terms trigger vector search fallback. (3) Complex queries may invoke multiple tools sequentially.

Safety Rules. Three key safety principles govern system behaviour (1) **Source of Truth** : Price, availability, and property specifications must come from the live API. Vector snippets provide contextual information but cannot override API facts. (2) **Correct Abstention**: If constraints conflict or required data is missing, the system prefers to abstain rather than return incorrect results. This follows selective classification principles where systems should decline when confidence is low [23]. (3) **User Isolation**: Each conversation session maintains isolated state through unique thread identifiers, preventing cross-user data leakage.

3.8 *Reproducibility*

All configuration parameters are stored in version-controlled files including API endpoints, authentication keys, timeout values, ChromaDB collection names, embedding model specifications, and LLM parameters. Each evaluation run records: Runner version and timestamp, Prompt template hash, Vector index snapshot identifier, Per-request logs with tool calls, latencies, and token counts

The evaluator produces structured output including per-query CSV records and aggregated metrics with confusion matrix analysis. We use LLM temperature 0 to ensure deterministic outputs across runs. The complete implementation, evaluation scripts, and gold-labelled questions are available in a public repository.

3.9 *Limitations*

This study acknowledges several methodological constraints:
1. **Gold Set Size** : The evaluation uses 30 questions spanning 12 categories. While carefully constructed to cover diverse query types, this limits statistical power for fine-grained subgroup analysis.
2. **Single Market** : Evaluation data comes from the Medan, Indonesia real estate market. Results may vary in other regions with different naming conventions, price ranges, or property characteristics.
3. **Index Freshness** : The vector index is refreshed periodically but may lag behind the live database. While final verification uses API data, temporary

recall reduction can occur under high inventory churn.

4. **No User Study** : Evaluation focuses on objective constraint satisfaction metrics. Perceived usefulness, trust, and user satisfaction require separate usability studies with human participants.

4 Results and Discussion

4.1 *Results*

Table 7 summarizes the evaluation results across three retrieval pipelines: Vector-only, API-only, and Hybrid.

Table 7 : Summary of evaluation results

Metric	Vector	API	Hybrid	Δ Hybrid vs API
Mean CPR	55.33%	73.35%	97.61%	+24.26%
Strict Success Ratio	33.04%	72.62%	96.62%	+24.00%
Query Success Ratio	50.00%	73.33%	100.00%	+26.67%
Precision	100.00%	100.00%	100.00%	0
Recall	53.57%	71.43%	100.00%	+28.57%
F1	66.77%	83.33%	100.00%	+16.67%
Accuracy	56.67%	73.33%	100.00%	+26.67%

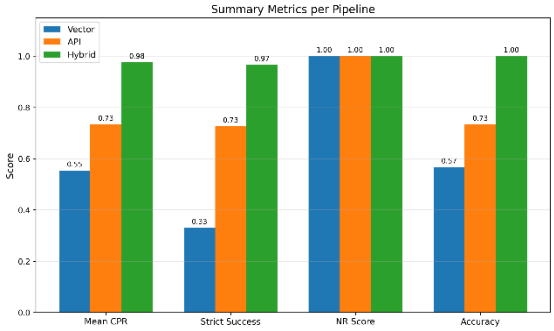

Figure 4a: Summary Metrics per Pipeline

Results reveal a clear performance hierarchy: **Hybrid > API > Vector**. Figure 4a visualizes this comparison across four key metrics: Mean CPR, Strict Success Ratio, No-Result Score, and Accuracy. The green bars (Hybrid) consistently reach or approach 1.0 across all metrics, while Vector (blue) shows the lowest performance, particularly on Mean CPR (0.55) and Accuracy (0.57).

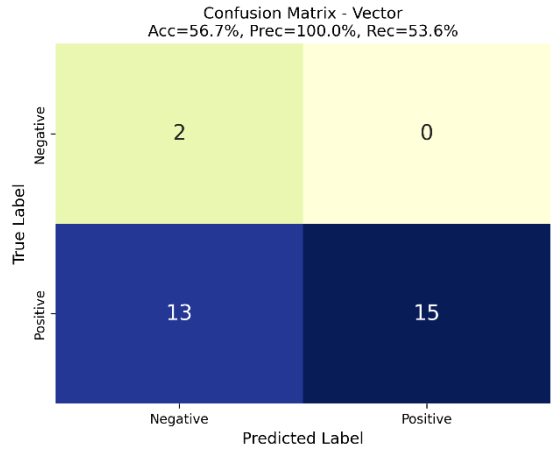

Figure 4b: Confusion Metrix – Vector

Vector-only achieves 56.67% accuracy with 13 false negatives (Figure 4b). The pipeline excels at location matching (PCA=94.20%) but fails on transactional constraints—price accuracy is only 43.75% and property_type only 52.17%. This confirms that semantic search alone cannot reliably enforce structured constraints.

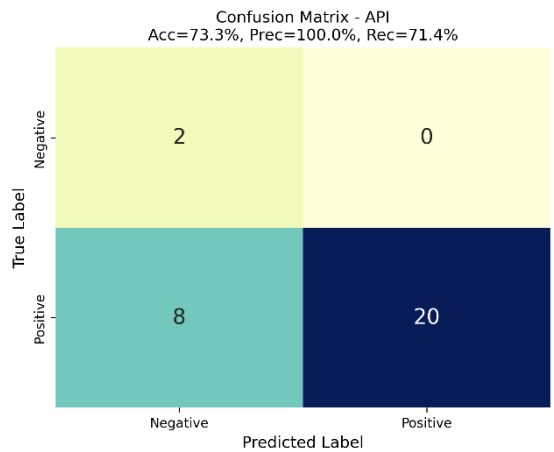

Figure 4c: Confusion Metrix – API

API-only improves significantly over Vector with 73.33% accuracy and F1=83.33% (Figure 4c). The pipeline achieves perfect constraint accuracy on structured fields (property_type, listing_type, price, bedrooms at 100%). However, API-only fails entirely on feature_search (0% success) and

nearby_search (0% success) categories, producing 8 false negatives.

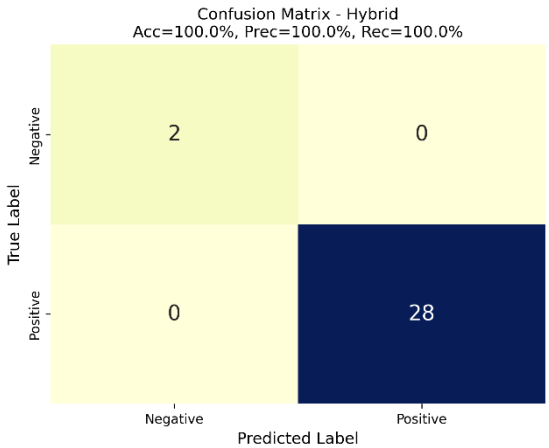

Figure 4d: Confusion Metrix – Hybrid

Hybrid achieves perfect question-level performance with 100% accuracy, 100% recall, and F1=1.0 (Figure 4d). The pipeline successfully answers all 28 positive queries (TP=28, FN=0) and correctly rejects both negative queries (TN=2, FP=0). Critically, Hybrid succeeds on feature_search (100% Query Success Rate, 90% Mean CPR) and nearby_search (100% Query Success Rate, 95% Mean CPR) where both baselines fail completely.

Table 8 presents the Per-Constraint Accuracy (PCA) breakdown for each pipeline.

Table 8 : Per-Constraint Accuracy by Pipeline

Constraint	Vector	API	Hybrid
Property_type	52.17%	100%	100%
Listing_type	83.05%	100%	100%
Location	94.02%	98.26%	98.82%
Price	43.75%	100%	100%
Bedrooms	52.94%	100%	100%
Floors	66.67%	100%	75%

The per-constraint breakdown reveals complementary strengths. Vector excels at location understanding (94.20%) through semantic matching of area names and landmarks. API guarantees transactional correctness for numeric fields. Hybrid inherits both advantages, with a minor trade-off on floors (75% vs API's 100%) due to occasional semantic interference.

Figure 4 presents the evaluation visualizations: Figure 4a shows the summary metrics comparison as a bar chart, while Figures 4b-4d display the confusion matrices for Vector, API, and Hybrid respectively.

All three methods achieve 100% precision (no false positives), indicating conservative behavior—when results are returned, they satisfy constraints. The key differentiator is recall: Hybrid achieves perfect recall (100%) while Vector misses 13 and API misses 8 answerable queries. The false negatives in API primarily come from feature_search (5 queries) and nearby_search (3 queries) categories that require semantic understanding beyond structured filters.

4.2 *Discussion*

Vector Strengths and Weaknesses. Vector retrieval excels at semantic understanding—matching synonyms, spelling variants, and informal location references. Its high location PCA (94.20%) demonstrates effective area name recognition. However, reliance on embedded snippets without database verification leads to poor transactional accuracy. Price constraints fail at 43.75% because embedded price strings may be outdated or incorrectly parsed.

API Strengths and Weaknesses. API retrieval guarantees transactional correctness by querying the live database with structured filters. The perfect PCA scores on property_type, listing_type, price, and bedrooms (all 100%) confirm this reliability. However, API cannot interpret semantic queries about features ("rumah dengan CCTV") or proximity ("dekat mall") because these concepts are not encoded as structured database fields.

Why Hybrid Succeeds. The Hybrid approach combines both retrieval paradigms through sequential execution with semantic re-ranking:
(1) **API-First for Accuracy**: Structured constraints (price, bedrooms, property_type) are handled by API queries, ensuring transactional correctness. (2) **Semantic Fallback for Coverage**: When API returns empty results (e.g., feature queries), ChromaDB semantic search provides fallback coverage by matching descriptive terms in property listings. (3) **Score Fusion for Ranking**: When API returns results, semantic scores re-rank candidates to

prioritize relevance: score = 0.6 × semantic + 0.4 × api_position.

This architecture enables Hybrid to handle feature_search queries (CCTV, WiFi, parking) by matching these terms in property descriptions, while API and Vector alone cannot address such queries.

Practical Implications. For practitioners deploying property search assistants: (1) **Hybrid is essential** : Neither API nor Vector alone provides comprehensive coverage. The 100% success rate demonstrates that combined semantic-structured retrieval is necessary for diverse query types. (2) **Feature and proximity queries require semantic components**: The 0% API success on these categories confirms that structured databases alone cannot address natural language queries about amenities and relative locations. (3) **Transactional fields need API grounding** : Vector-only systems should not be trusted for price-sensitive applications, as demonstrated by the 43.75% price accuracy.

5 Conclusion & Future Works

5.1 *Conclusion*

This study presented a channel-agnostic RAG assistant for real estate property search, comparing three retrieval pipelines—Vector-only (semantic search), API-only (structured query), and Hybrid (combined approach)—on 30 gold-labeled questions using constraint-based evaluation.

The key finding is that **Hybrid achieves perfect question-level performance** with 100% accuracy, 100% recall, and F1=1.0, significantly outperforming API-only (73.33% accuracy, F1=83.33%) and Vector-only (56.67% accuracy, F1=69.77%). At the constraint level, Hybrid maintains 97.61% Mean CPR compared to 73.35% for API and 55.33% for Vector.

Critically, Hybrid successfully handles feature-based queries (100% success, 90% CPR) and proximity-based queries (100% success, 95% CPR) where both API and Vector fail entirely (0% success). This demonstrates that combining semantic and structured retrieval provides comprehensive coverage that neither approach achieves alone.

The constraint-based evaluation protocol developed in this study—measuring Per-Constraint Accuracy, Constraint Pass Ratio, and confusion matrix metrics—provides a reproducible framework for assessing property search systems. Unlike traditional IR metrics (MRR, NDCG), this approach explicitly captures constraint satisfaction essential for transactional domains.

5.2 *Future Works*

Several directions merit future investigation:

1. **Dynamic Routing**: Develop a learned policy to automatically select Vector, API, or Hybrid retrieval based on query characteristics, optimizing for both accuracy and latency.
2. **Expanded Evaluation**: Scale the gold set to 100-200 questions with multiple annotators and inter-annotator agreement metrics (Cohen's κ) for robust statistical analysis.
3. **User Study**: Conduct usability evaluation measuring System Usability Scale (SUS), task completion time, and qualitative user feedback.
4. **Multi-turn Dialogue**: Extend the system to handle clarification requests and sophisticated preference refinement across conversation turns.

Acknowledgments

I am deeply grateful to my advisor, Suryadiputra Liawatimena, and the thesis committee at BINUS University for their guidance, constructive feedback, and steady encouragement throughout this research. Their probing questions sharpened the methodology and helped clarify the evaluation design. Special thanks go to PT. Meta Properti Indonesia for curating property listings, maintaining the live API, and supporting 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 unwavering 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 API endpoints, evaluation scripts, and pipeline configurations, is available in public GitHub repo at: github.com/ripinlibinus/rag-property-assistant. The repository includes setup instructions, environment specifications, and example configuration files. The 30-question gold-labelled evaluation set, along with constraint annotations, is provided in the repository under data/gold/questions_v2.json. The evaluation scripts are located in scripts/evaluate_v2.py and src/evaluation/. Property listings (2,857 records) and their vector embeddings are stored in data/chromadb/properties/. In addition, the repository provides database schema and seed data that can be imported via Laravel migrations to reconstruct the database used in our experiments.

References

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

[2] 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.

[3] P. Lewis *et al.*, “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks”, Accessed: Nov. 02, 2025. [Online]. Available: https://github.com/huggingface/transformers/blob/master/

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[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] 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;PAGE:STRING:ARTICLE/CHAPTER.

[10] 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.

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

[12] 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.

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

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

[15] 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.

[16] 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.

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

[18] A. Abdallah, J. Mozafari, B. Piryani, M. Ali, and A. Jatowt, "From Retrieval to Generation: Comparing Different Approaches," Feb. 2025, Accessed: Jan. 27, 2026. [Online]. Available: https://arxiv.org/pdf/2502.20245

[19] A. Rao, H. Alipour, and N. Pendar, "Rethinking Hybrid Retrieval: When Small Embeddings and LLM Re-ranking Beat Bigger Models," May 2025, Accessed: Jan. 27, 2026. [Online]. Available: https://arxiv.org/pdf/2506.00049

[20] N. N. Doan, A. Härmä, R. Celebi, and V. Gottardo, "A Hybrid Retrieval Approach for Advancing Retrieval-Augmented Generation Systems," 2024. Accessed: Jan. 27, 2026. [Online]. Available: https://aclanthology.org/2024.icnlsp-1.41/

[21] 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.

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

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