VerifAI

Refusal-aware RAG system for proprietary documents that prioritizes grounding, confidence estimation, and observability over answer rate

Overview

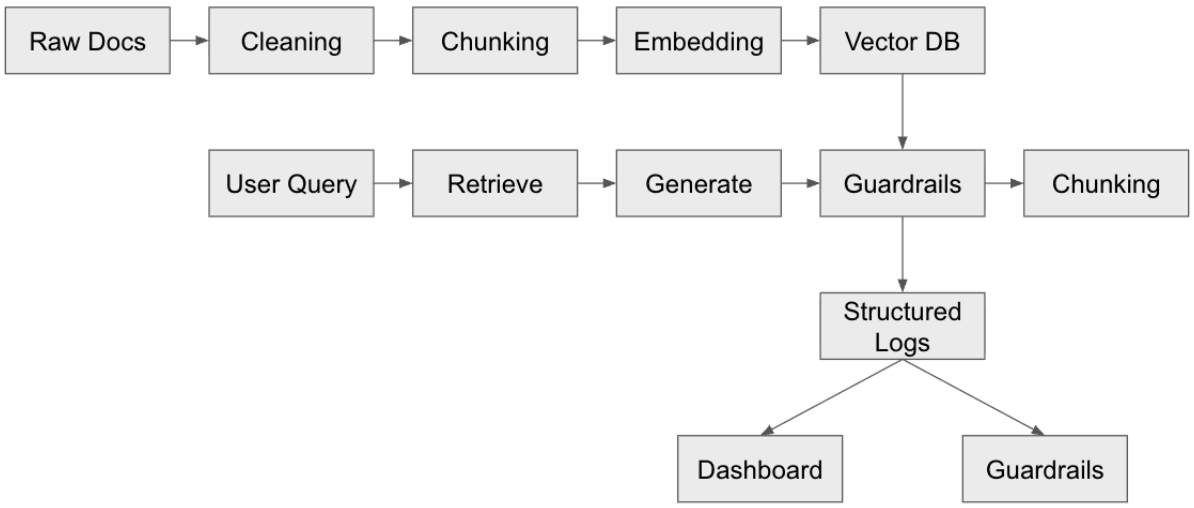
VerifiAI is a production-oriented RAG system designed specifically for proprietary and internal knowledge bases, such as company policies, onboarding documents, or internal manuals.

Unlike general-purpose chatbots, VerifiAI prioritizes:

1) Grounded Answers
2) Explicit Refusals
3) Measurable Confidence
4) Operational Observability.

This makes it suitable to be integrated into enterprise, compliance-sensitive, and internal AI deployment settings where hallucination is unacceptable.

High-Level Architecture



Design Philosophy

VerifAI improves upon the typical RAG app that focuses on purely answer quality by focussing on answer correctness, refusal correctness and observability.

Key Principles:

1) Refuses to answer queries for which there is insufficient information present in the vector DB to answer correctly
2) When uncertain on how to answer a query, this uncertainty is made explicit
3) Everything that could be of use to the user is logged

Document Processing Pipeline

The documents to be ingested into the vector DB are cleaned, chunked and audited.

Cleaning

Raw PDFs converted to text often contain broken lines, random line breaks, inconsistent spacing and headers split across lines.
The cleaning script normalizes text accordingly through:

1) Normalising line endings
2) Removing trailing whitespace
3) Merging broken paragraph lines
4) Preserving section headers and list items

This process allows chunk semantic coherence to be improved which in turn improves embedding quality.

Chunking

Documents in the form of normalized text files now are split into semantically meaningful chunks through structural chunking based on logical grouping.

Chunking Strategy

- Section headers are used as natural boundaries
- Very small sections are merged into adjacent chunks
- A minimum chunk size of 200 is maintained to enforce embedding quality. If a section is too small, it is absorbed into the previous chunk to ensure each chunk has an acceptable level of information density.

Each chunk stores the following metadata as well: 1) doc_id 2) chunk_id 3) source

Embedding-Based splitting was not used as it might fail to split sections which are similar and differ in semantic meaning only very slightly. Furthermore, structural chunking makes use of REGEX instead of having to run each sentence through an embedding model which improves processing speed and reduces cost.

Chunk Auditing

In order to enforce a high standard of quality for the data that is embedded, we audit the quality of chunks.

Chunks that are too short ⇒ weak semantic signal
Chunks that are too long ⇒ attention dilution (relevant information required to answer a question is buried within a large volume of irrelevant or less important context)
Chunks that are missing headers ⇒ poor interpretability

Vector Embedding

The model all-MiniLM-L6-v2 was chosen to embed the chunks into vector embeddings as it offers a good balance of strong performance and high efficiency. This makes it ideal for fast semantic search, clustering and information retrieval.

The chunks are then stored in ChromaDB with persistence enabled. This allows the stored data to be persisted automatically and loaded on start.

Retrieval

The retrieval layer is responsible for identifying the most relevant document chunks for a given query. VerifiAI uses dense vector retrieval with use of **SentenceTransformers** and **ChromaDB**.Each query is embedded using the same embedding model (all-MiniLM-L6-v2) that was used during ingestion, ensuring embedding space consistency. The query embedding is then compared against persisted document embeddings stored in ChromaDB, returning the top-k closest chunks along with their distances and metadata.

Distances returned by ChromaDB are converted into similarity scores using a simple inverse transformation. This makes similarity values more intuitive and directly usable downstream. The retriever returns a structured tuple of (chunk_text, similarity_score, metadata) rather than raw database results. This design keeps retrieval transparent and auditable, allowing later stages (confidence estimation, refusal logic, logging) to reason explicitly about retrieval strength instead of treating it as a black box.

Generation

VerifiAI enforces hard grounding through prompt design. The model is explicitly instructed to answer only using the provided context, cite supporting chunks, and produce an explicit refusal statement if the answer cannot be derived from the retrieved documents.
Context is assembled as a sequence of chunk blocks, each tagged with a stable chunk ID. These IDs are required to appear in citations within the answer, making it possible to trace every claim back to source text. This citation requirement is not only a correctness mechanism, but also a signal used later for confidence estimation.

The generation function returns structured output rather than raw text. This includes the generated answer, source metadata, the model used, and latency. By keeping generation structured, the system can cleanly separate generation concerns from guardrails, logging, and evaluation.

Guardrails

This includes both confidence estimation and refusal logic.

Confidence Estimation

Confidence estimation provides a quantitative measure of how trustworthy a generated answer is, expressed as a value between 0 and 1.

The confidence score combines four factors:

1) Retrieval coverage
    a) Coverage ensures enough context was retrieved
2) Similarity strength
    a) Similarity ensures the context is relevant
3) Answer length relative to context
    a) Length ensures the answer does not exceed what the context can support
4) Citation usage
    a) Citations ensure claims are explicitly grounded

In the case where the model outputs a refusal phrase or no retrieval occurs, confidence is set to 0.

Refusal Logic

Refusal logic determines whether the system should return an answer or intentionally refuse. Refusal is treated as a correct and desirable outcome when the system lacks sufficient grounding. The decision is based on three conditions:

1) Absence of retrieved chunks
2) Low overall confidence
3) Weak similarity among retrieved chunks

Structured Logging

Every query is logged as a single append-only JSON object. Logs capture the full lifecycle of a request: the query, retrieved chunk IDs and similarity scores, generation details, confidence, latency breakdown, cost placeholder, and final outcome. Logging is performed synchronously at the API boundary to ensure no request escapes observation.

The structured, schema-consistent format enables reliable downstream analysis, dashboarding, and auditing. Logs are designed to be human-readable while remaining machine-parsable, making them suitable for both debugging and operational monitoring.

Monitoring Dashboard

The Streamlit dashboard provides real-time visibility into system behavior using the structured logs. It surfaces high-level operational metrics such as refusal rate, error rate, average latency, and cost, along with breakdowns of retrieval and generation latency.

The dashboard also exposes refusal reasons and confidence distributions, allowing operators to identify whether refusals are caused by weak retrieval, low confidence, or systemic issues.

Evaluation

Offline evaluation scripts test the system against test cases with known expected behavior. These scripts measure grounding, refusal correctness and confidence behavior.

Additional audit scripts inspect chunk quality. This ensures retrieval failures can be traced back to ingestion issues rather than being misattributed to model behavior.

Design choices

Retrieval based confidence calculation instead of model-reported confidence	Calculating confidence from observed signals like retrieval coverage, similarity strength, citation usage and answer length ensures that the confidence reflects evidence quality.
Refusal is considered a successful outcome	Refusal is treated as correct behavior, not an error. Refusals are logged down with structured reasons as well. This allows teams to understand that their knowledge base has gaps, thus prioritising safety and correctness over answer rate.
Persistent vector storage with ChromaDB	ChromaDB was chosen for its lightweight local persistence, transparency, and fast iteration. Persistence ensures embeddings are built once and reused across restarts, mirroring production systems where ingestion and serving are decoupled.
Single embedding model for both ingestion and retrieval	Using the same embedding model (all-MiniLM-L6-v2) across the entire pipeline guarantees embedding space consistency. This avoids retrieval degradation caused by mismatched encoders and improves retrieval reliability.
Structural chunking instead of semantic chunking	Documents are chunked using section headers and merged heuristics instead of fixed token windows or embedding similarity.

Structured API outputs	Every internal component returns structured data (answer, sources, model_used, confidence, latency). This allows guardrails, logging, and evaluation to operate independently of the LLM, enabling system-level reasoning and debugging.

Target Audience

Designed for teams building RAG applications over internal, policy, legal, HR, or operational documents where accuracy, traceability, and refusal behavior matter more than conversational polish.