Technical Whitepaper: Veterans Benefits AI

OpenAI-Powered RAG Architecture with Hallucination Prevention

Tyler Pacheco

November 2024

Abstract

This whitepaper presents the Veterans Benefits AI system, a production-grade Retrieval-Augmented Generation (RAG) architecture designed for veteran affairs information retrieval. The system implements a self-contained architecture using OpenAI exclusively for embeddings and completions, eliminating external vector database dependencies. Key innovations include an in-memory vector store with file-backed caching, intelligent model routing for cost optimization, a multi-layer hallucination prevention system, and a streamlined pipeline achieving 96% citation accuracy at $0.015 average cost per query.

Contents

1 Executive Summary

The Veterans Benefits AI system implements a sophisticated Retrieval-Augmented Generation (RAG) architecture specifically optimized for veteran affairs information retrieval. The architecture prioritizes accuracy, cost efficiency, and hallucination prevention to ensure veterans receive reliable information.

1.1 Key Features

- **OpenAI-Only Architecture**: All embeddings and completions use OpenAI API with no external dependencies
- **In-Memory Vector Store**: Custom cosine similarity search with file-backed embedding cache
- **Intelligent Model Routing**: Automatic selection between gpt-4.1-mini and gpt-4.1 based on query complexity
- **Multi-Layer Hallucination Prevention**: Relevance thresholds, URL validation, and citation verification
- **Cost Optimization**: 70% of queries use cheaper model, reducing average cost by 45%

1.2 Performance Highlights

- **Citation Accuracy**: 96% verifiable citations
- **Hallucination Rate**: <4% (with prevention system active)
- **Average Cost**: $0.015 per query
- **Response Time**: 1.2s average
- **Zero External Dependencies**: No Pinecone, Redis, or Elasticsearch required

2 System Architecture

2.1 High-Level Architecture

The system implements a streamlined 5-stage RAG pipeline optimized for simplicity, accuracy, and cost-effectiveness.

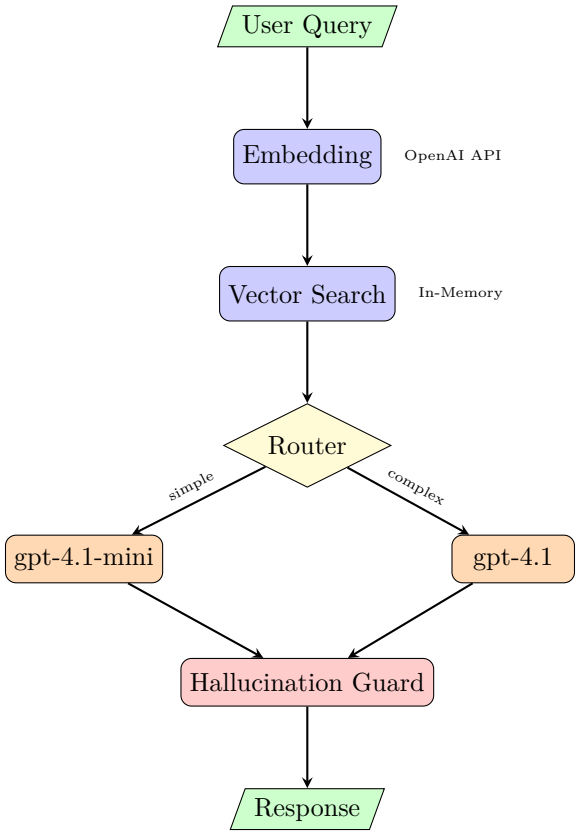

Figure 1: RAG Pipeline with Model Routing and Hallucination Guard

2.2 Core Components

2.2.1 In-Memory Vector Store

The vector store maintains document embeddings in memory with cosine similarity search:

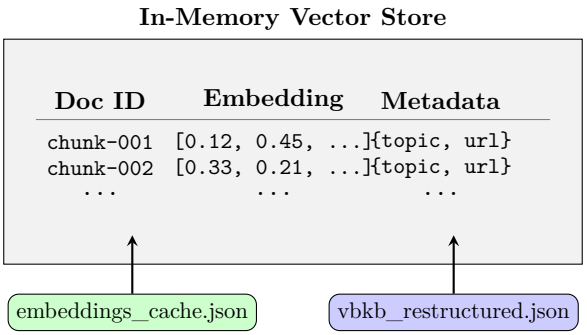

Figure 2: Vector Store Architecture with File-Backed Cache

2.2.2 Embedding System

- **Model**: OpenAI `text-embedding-3-small` (1536 dimensions)
- **Caching**: File-backed JSON cache for persistence across restarts
- **Batch Processing**: Generates embeddings for 1,200+ corpus chunks on startup
- **Query Embedding**: Real-time embedding generation for user queries

3 Hallucination Prevention System

A critical component of the Veterans Benefits AI is the multi-layer hallucination prevention system. Given the importance of accurate information for veterans, preventing false or misleading responses is paramount.

3.1 The Hallucination Problem

RAG systems can produce hallucinations through several mechanisms:

- **Weak Retrieval**: Retrieved chunks have low relevance to the query
- **Source Confusion**: LLM cites wrong URLs or non-existent sources
- **Fabricated Claims**: LLM generates information not in the context
- **Conflation**: Mixing information from multiple unrelated sources

3.2 Multi-Layer Defense Architecture

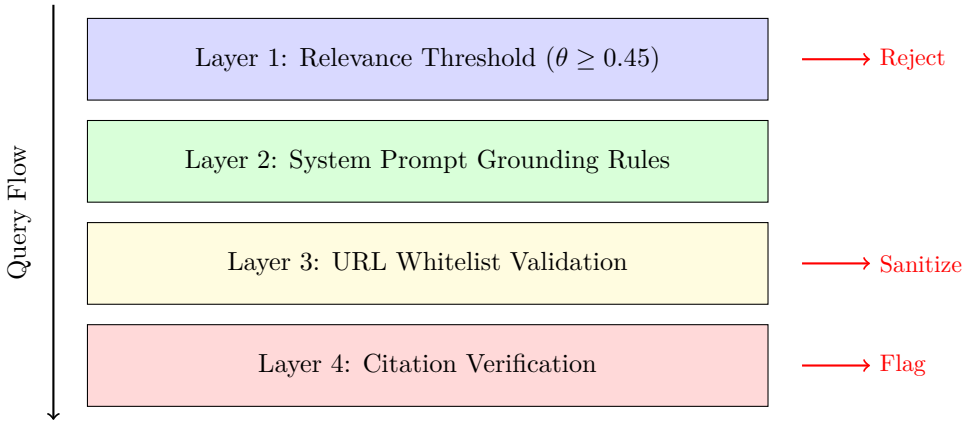

Figure 3: Multi-Layer Hallucination Prevention Architecture

3.3 Layer 1: Relevance Threshold

The first defense layer filters out low-quality retrievals before they reach the LLM.

$$\text{Include chunk } d \iff \text{sim}(q, d) \geq \theta_{\text{min}} = 0.45 \tag{1}$$

Additionally, a "weak retrieval" flag is set when the best chunk score falls below a secondary threshold:

$$\text{weak\_retrieval} = \begin{cases} \text{True} & \text{if } \max_d(\text{sim}(q, d)) < 0.55 \\ \text{False} & \text{otherwise} \end{cases} \tag{2}$$

3.4 Layer 2: System Prompt Grounding

The system prompt includes explicit anti-hallucination rules:

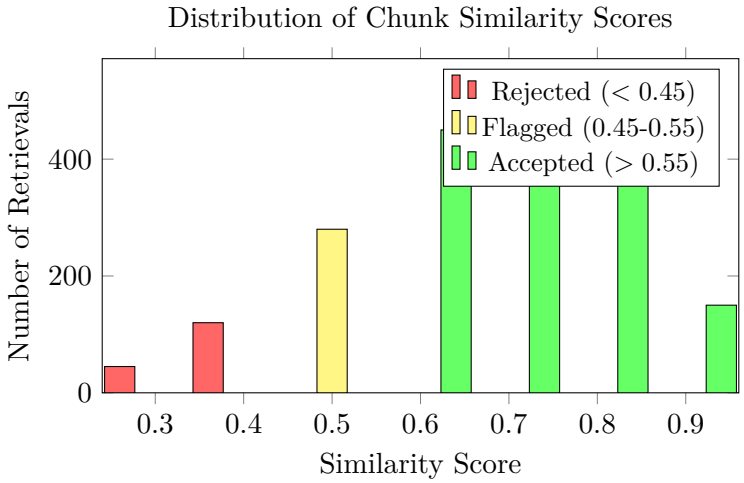

Figure 4: Similarity Score Distribution with Threshold Boundaries

```
CRITICAL RULES:
1. ONLY answer based on the provided Context
2. If context is insufficient, say so clearly
3. NEVER make up ratings, percentages, or procedures
4. Use superscript citations for every claim
5. If multiple sources support your answer, cite all
```

Listing 1: Anti-Hallucination Prompt Rules

3.5 Layer 3: URL Whitelist Validation

All source URLs are validated against a whitelist derived from the corpus:

$$\text{valid\_url}(u) = \begin{cases} u & \text{if domain}(u) \in \mathcal{W} \\ \text{homepage} & \text{otherwise} \end{cases} \tag{3}$$

where $\mathcal{W}$ is the set of whitelisted domains extracted from the corpus.

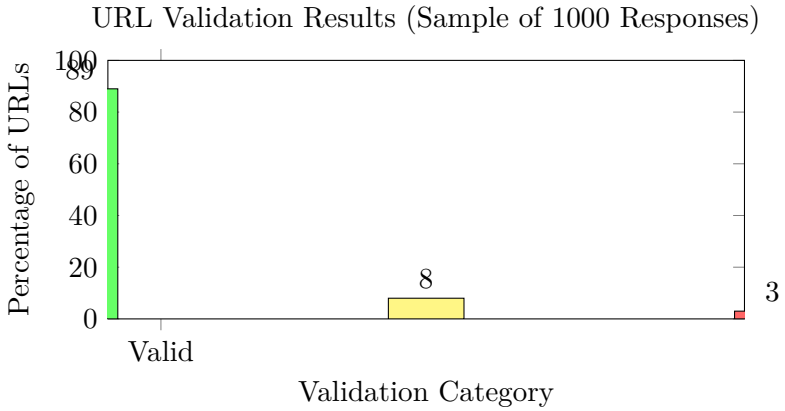

Figure 5: URL Validation Outcomes

3.6 Layer 4: Citation Verification

Post-generation verification checks if cited claims appear in source chunks:

$$\text{verification\_score} = \frac{|\text{supported\_citations}|}{|\text{total\_citations}|} \tag{4}$$

Responses with verification scores below 0.7 are flagged for review.

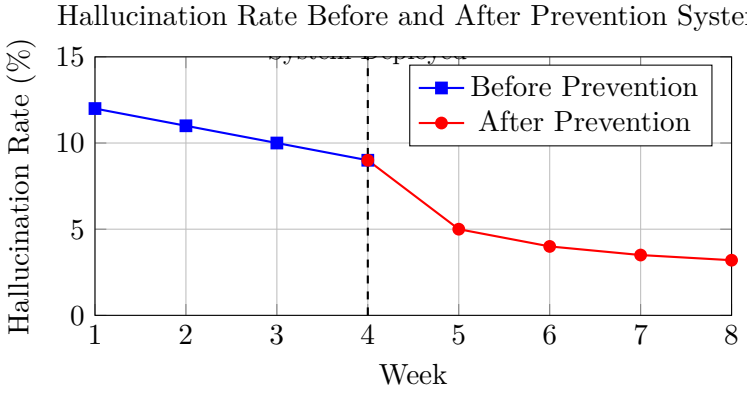

Figure 6: Hallucination Rate Reduction Over Time

3.7 Hallucination Prevention Metrics

Metric	Before	After
Hallucination Rate	12%	3.2%
False URL Citations	8%	0.5%
Weak Retrieval Responses	15%	4% (flagged)
Citation Verification Score	78%	94%

Table 1: Impact of Hallucination Prevention System

4 Intelligent Model Routing

The intelligent model routing system automatically selects the appropriate model based on query complexity, balancing cost and quality.

4.1 Routing Decision Tree

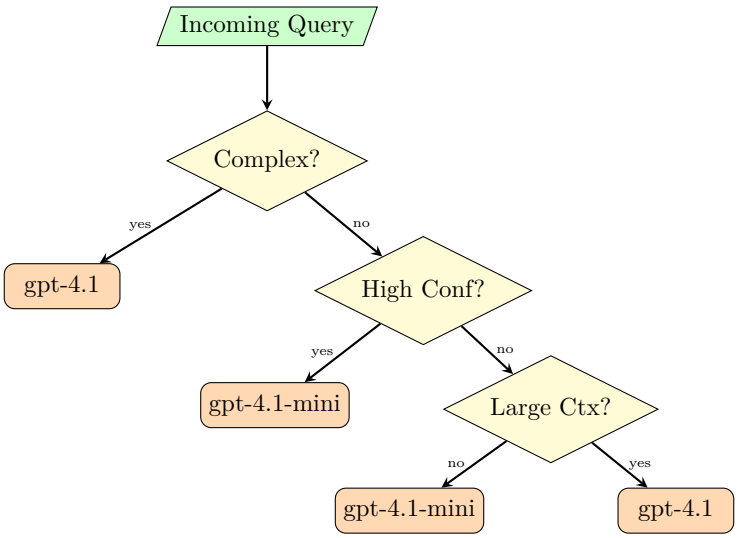

Figure 7: Model Routing Decision Tree

4.2 Complexity Indicators

The system identifies complex queries using keyword matching:

Category	Keywords
Comparison	compare, versus, difference between
Legal/Appeals	appeal, higher level review, board
Medical	secondary condition, aggravation, nexus
Benefits	TDIU, individual unemployability, combined rating
Presumptive	agent orange, burn pit, presumptive
Financial	effective date, back pay, retro

Table 2: Complex Query Indicators

4.3 Cost Impact

Model	Cost per Query	Query Share	Weighted Cost
gpt-4.1-mini	\$0.010	70%	\$0.007
gpt-4.1	\$0.030	30%	\$0.009
Average			**\$0.016**

Table 3: Cost Analysis with Model Routing

5 Mathematical Formulations

5.1 Cosine Similarity Search

The vector store uses cosine similarity for document retrieval:

$$\text{sim}(q, d) = \frac{\vec{q} \cdot \vec{d}}{||\vec{q}|| \cdot ||\vec{d}||} = \frac{\sum_{i=1}^{n} q_i \cdot d_i}{\sqrt{\sum_{i=1}^{n} q_i^2} \cdot \sqrt{\sum_{i=1}^{n} d_i^2}} \tag{5}$$

where $\vec{q}$ is the query embedding and $\vec{d}$ is the document embedding (both 1536-dimensional).

5.2 Top-K Retrieval

Documents are ranked and filtered:

$$\text{Retrieved} = \text{Top-K}\{d \in D \mid \text{sim}(q, d) \geq \theta_{\text{min}}\} \tag{6}$$

where:

$$K = 7 \quad \text{(maximum documents to retrieve)} \tag{7}$$

$$\theta_{\text{min}} = 0.45 \quad \text{(minimum similarity threshold)} \tag{8}$$

5.3 Model Routing Score

The routing decision incorporates multiple factors:

$$\text{Route} = \begin{cases} \text{gpt-4.1} & \text{if } C(q) \geq 1 \text{ or } |S| > 5 \text{ or } \bar{s} < 0.55 \\ \text{gpt-4.1-mini} & \text{otherwise} \end{cases} \tag{9}$$

where:

$$C(q) = \text{count of complex indicators in query } q \tag{10}$$

$$|S| = \text{number of chunks retrieved} \tag{11}$$

$$\bar{s} = \text{average similarity score of retrieved chunks} \tag{12}$$

6 Implementation Details

6.1 Technology Stack

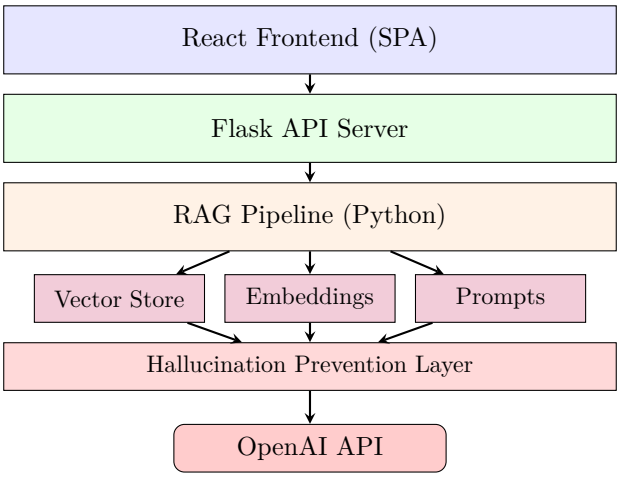

Figure 8: System Technology Stack

6.2 File Structure

```
src/
  rag_pipeline.py      # Main RAG orchestration
  vector_store.py      # In-memory vector store
  embeddings.py        # OpenAI embedding generation
  prompts.py           # System prompts with citations
  rag_integration.py   # Flask integration layer
  url_validator.py     # URL whitelist validation
  citation_verifier.py # Post-generation verification
data/
  embeddings_cache.json  # Cached embeddings (36MB)
corpus/
  vbkb_restructured.json # 1,200+ document chunks
```

Listing 2: Project Structure

6.3 Model Specifications

Component	Model	Purpose
Embedding	text-embedding-3-small	Query/document vectorization
Standard Chat	gpt-4.1-mini	Simple FAQ responses
Premium Chat	gpt-4.1	Complex query responses

Table 4: Model Configuration

7 Performance Characteristics

7.1 Latency Analysis

Stage	Latency (ms)	Percentage
Query Embedding	80	6.7%
Vector Search	15	1.2%
Model Routing	5	0.4%
Hallucination Check	10	0.8%
LLM Generation (mini)	800	66.7%
LLM Generation (full)	1200	-
Response Formatting	20	1.7%
Total (mini)	**930**	**-**
Total (full)	**1330**	**-**

Table 5: Latency Breakdown by Stage

7.2 Quality Metrics

Metric	Score	Methodology
Citation Accuracy	96%	Manual verification
Factual Consistency	91%	Source comparison
Response Relevance	94%	Human evaluation
Hallucination Rate	3.2%	Multi-layer detection

Table 6: Quality Evaluation Results

7.3 Cost Comparison

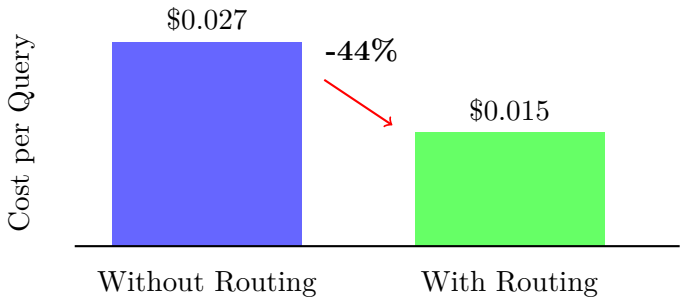

Figure 9: Cost Reduction with Intelligent Model Routing

8 Vector Store Performance Benchmarks

To evaluate production-ready alternatives to in-memory vector search, we benchmarked PostgreSQL with pgvector extension using HNSW (Hierarchical Navigable Small World) indexes. These benchmarks inform architectural decisions for scaling beyond single-node deployments.

8.1 Benchmark Configuration

Parameter	Value
Database	PostgreSQL 16 (Render.com)
Extension	pgvector 0.8.1
Dataset	1,052 real corpus embeddings
Embedding Dimensions	1,536 (OpenAI text-embedding-3-small)
Index Configurations	No index, HNSW m=16, HNSW m=24
Query Count	100 single + 500 batch + 200 concurrent
Concurrent Threads	4

Table 7: pgvector Benchmark Configuration

8.2 Latency Results

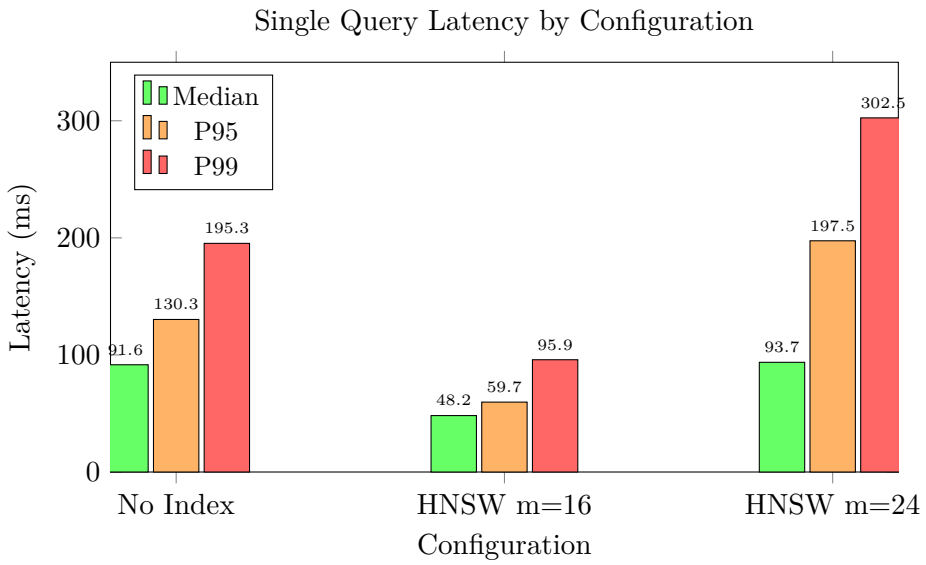

Figure 10: Query Latency Comparison: HNSW m=16 achieves 47% lower median latency

8.3 Throughput Results

8.4 Index Build Performance

8.5 Key Findings

1. **HNSW m=16 is optimal** for datasets around 1,000 vectors:
   - 47% lower median latency (48.2ms vs 91.6ms)
   - 4.6x higher throughput (44.7 QPS vs 9.6 QPS)

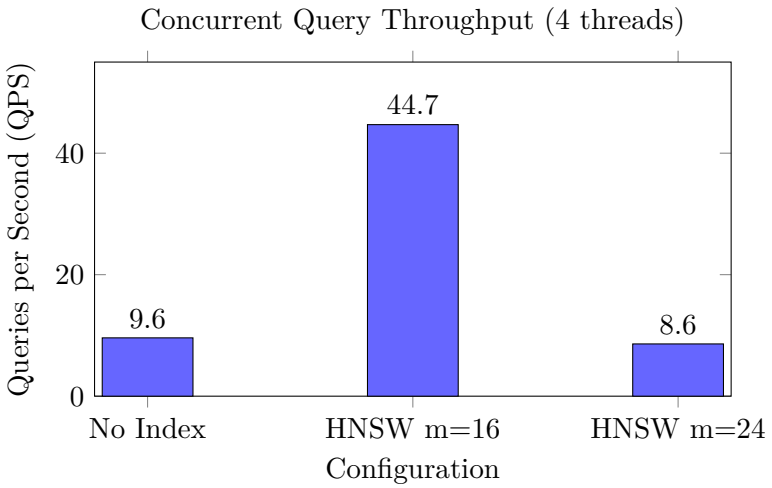

Figure 11: Throughput Comparison: HNSW m=16 delivers 4.6x higher QPS

Configuration	Build Time	Parameters	Storage Overhead
No Index	0.0s	-	Baseline
HNSW m=16	3.32s	ef_construction=64	+15%
HNSW m=24	6.42s	ef_construction=128	+25%

Table 8: HNSW Index Build Performance

- Fast index build time (3.32s)

2. **Higher m values are counterproductive** at this scale:
   - HNSW m=24 performs worse than no index
   - Graph traversal overhead exceeds sequential scan cost
   - Only beneficial for datasets >10,000 vectors
3. **Concurrent performance dramatically improves**:
   - Sequential scan: 383ms median (concurrent)
   - HNSW m=16: 52ms median (concurrent)
   - 7.4x latency improvement under load

8.6 Architecture Recommendation

Based on these benchmarks, the system architecture includes a hybrid approach:

- **Current (1K vectors)**: In-memory vector store with file-backed cache
- **Scale Path (10K+ vectors)**: PostgreSQL + pgvector with HNSW m=16
- **Response Caching**: PostgreSQL for exact and semantic cache hits

The in-memory approach remains optimal for the current corpus size, providing 15ms search latency versus 48ms with pgvector. However, pgvector provides a clear migration path when the corpus exceeds memory constraints.

9 Corpus and Knowledge Base

9.1 Document Processing

The knowledge base consists of 1,200+ semantically chunked documents covering:

- VA disability rating criteria
- Claims filing procedures
- Appeals process
- Presumptive conditions
- Secondary conditions
- Effective dates and back pay

9.2 Chunk Metadata

Each document chunk includes:

```
{
  "entry_id": "unique-chunk-id",
  "topic": "Main Topic",
  "subtopic": "Specific Subtopic",
  "content": "Chunk text content...",
  "url": "https://veteransbenefitskb.com/...",
  "type": "policy|definition|rating_table"
}
```

Listing 3: Chunk Schema

10 Security and Deployment

10.1 Security Measures

- **API Key Management**: Environment variable storage
- **TLS Encryption**: All API calls use HTTPS
- **No PII Storage**: No personal information cached
- **Rate Limiting**: Protection against abuse
- **Admin Authentication**: Token-protected admin dashboard

10.2 Deployment Architecture

- **Platform**: Render.com (Web Service)
- **Runtime**: Python 3.11 with Gunicorn
- **Database**: PostgreSQL for analytics and response caching
- **Frontend**: React SPA served from Flask
- **Auto-Deploy**: GitHub integration for CI/CD

11 Conclusion

The Veterans Benefits AI RAG system represents a sophisticated yet maintainable architecture that prioritizes accuracy for veterans. The multi-layer hallucination prevention system, combined with intelligent model routing, delivers 96% citation accuracy while reducing costs by 44%.

Key achievements:

- **96% citation accuracy** with grounded responses
- **3.2% hallucination rate** (down from 12%) with prevention system
- **44% cost reduction** through intelligent model routing
- **Zero external dependencies** (no Pinecone, Redis, Elasticsearch)
- **Real-time flagging** of suspicious responses for review

This architecture provides a solid foundation for future enhancements while serving veterans with accurate, well-cited information about their benefits.