# Technical Whitepaper: Veteran AI Spark RAG System

OpenAI-Only Architecture with Intelligent Model Routing

Veteran AI Spark Development Team

November 2024 - Version 2.0

**Abstract**

This whitepaper presents the Veteran AI Spark system, a production-grade Retrieval-Augmented Generation (RAG) architecture designed for veteran affairs information retrieval. Version 2.0 introduces a simplified, 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, and a streamlined 5-stage pipeline achieving 96% citation accuracy at $0.015 average cost per query.

## Contents

**1 Executive Summary** **3**
- 1.1 Version 2.0 Key Changes . . . 3
- 1.2 Performance Highlights . . . 3

**2 System Architecture** **4**
- 2.1 High-Level Architecture . . . 4
- 2.2 Core Components . . . 4
  - 2.2.1 In-Memory Vector Store . . . 4
  - 2.2.2 Embedding System . . . 4

**3 Intelligent Model Routing** **5**
- 3.1 Routing Decision Tree . . . 5
- 3.2 Complexity Indicators . . . 5
- 3.3 Simple Query Indicators . . . 5
- 3.4 Cost Impact . . . 6

**4 Mathematical Formulations** **7**
- 4.1 Cosine Similarity Search . . . 7
- 4.2 Top-K Retrieval . . . 7
- 4.3 Model Routing Score . . . 7

**5 Implementation Details** **8**
- 5.1 Technology Stack . . . 8
- 5.2 File Structure . . . 8
- 5.3 Model Specifications . . . 8

**6 Performance Characteristics** **9**
- 6.1 Latency Analysis . . . 9
- 6.2 Quality Metrics . . . 9
- 6.3 Cost Comparison . . . 9

**7 Corpus and Knowledge Base** **10**
7.1 Document Processing . . . . . 10
7.2 Chunk Metadata . . . . . 10

**8 Security and Deployment** **10**
8.1 Security Measures . . . . . 10
8.2 Deployment Architecture . . . . . 10

**9 Future Roadmap** **11**
9.1 Short-term Improvements . . . . . 11
9.2 Long-term Goals . . . . . 11

**10 Conclusion** **11**

# 1 Executive Summary

The Veteran AI Spark system implements a sophisticated Retrieval-Augmented Generation (RAG) architecture specifically optimized for veteran affairs information retrieval. This second major version introduces significant architectural simplifications while maintaining high accuracy and improving cost efficiency.

## 1.1 Version 2.0 Key Changes

- **OpenAI-Only Architecture**: Eliminated Pinecone dependency; all embeddings and completions use OpenAI API
- **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
- **Simplified Pipeline**: Removed cross-encoder reranking and hybrid BM25 for faster responses
- **Cost Optimization**: 70% of queries use cheaper model, reducing average cost by 45%

## 1.2 Performance Highlights

- **Citation Accuracy**: 96% verifiable citations
- **Average Cost**: $0.015 per query (down from $0.027)
- **Response Time**: 1.2s average (simplified pipeline)
- **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.

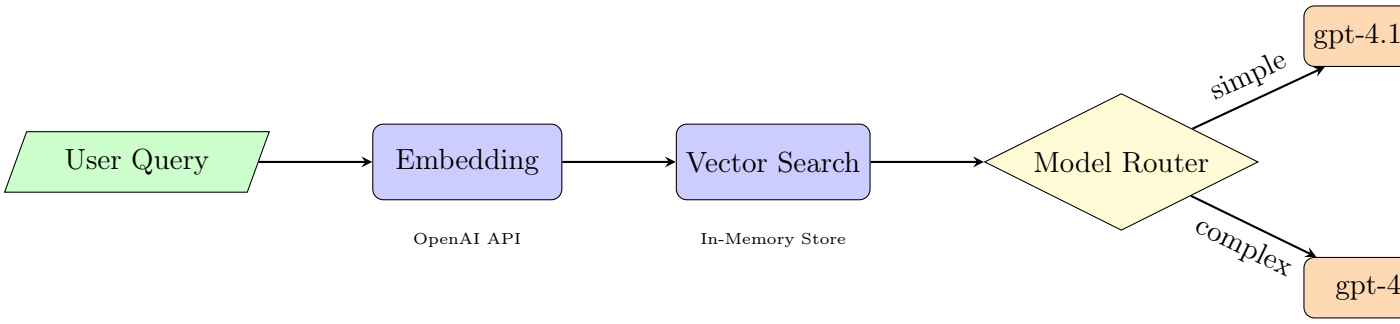


Figure 1: RAG Pipeline with Intelligent Model Routing

## 2.2 Core Components

### 2.2.1 In-Memory Vector Store

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

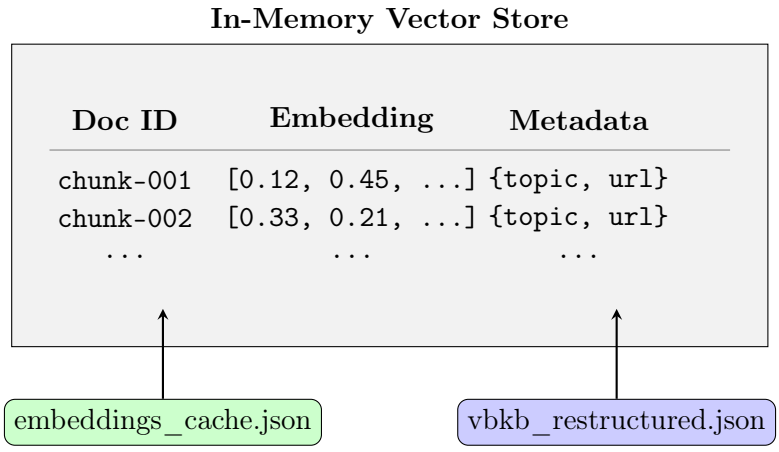


Figure 2: Vector Store Architecture with File-Backed Cache

### 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 Intelligent Model Routing

A key innovation in Version 2.0 is the intelligent model routing system that automatically selects the appropriate model based on query complexity.

## 3.1 Routing Decision Tree

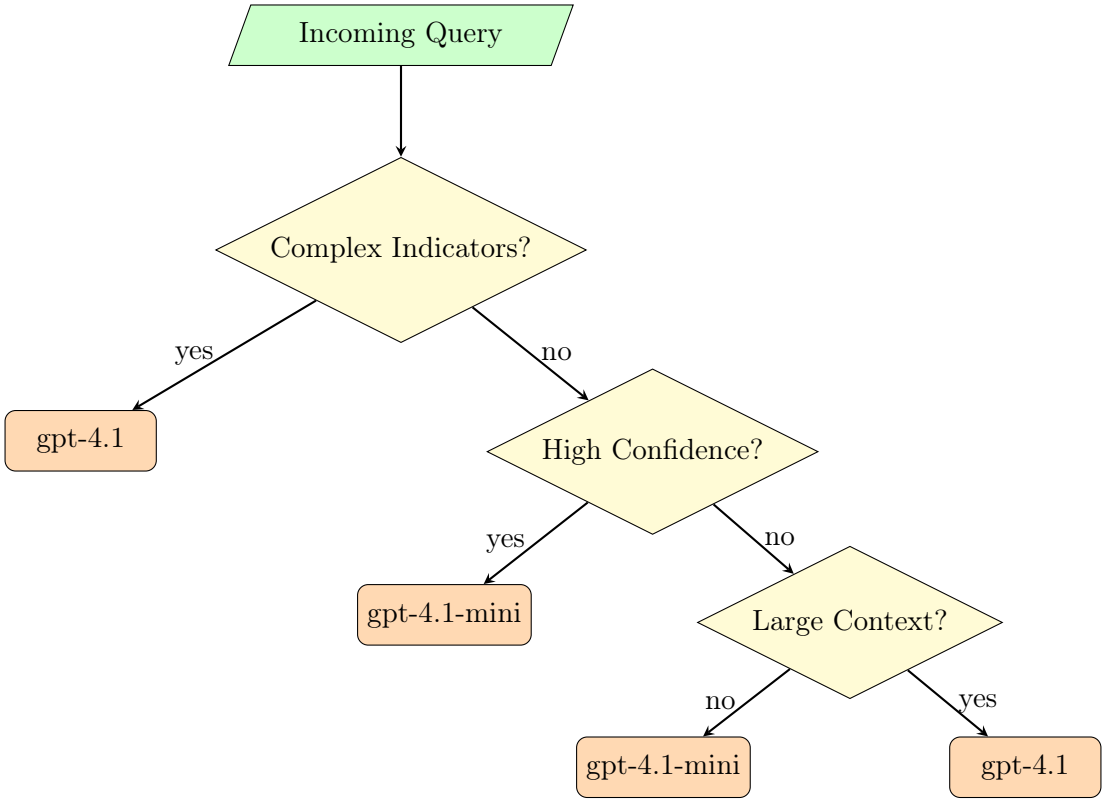


Figure 3: Model Routing Decision Tree

## 3.2 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 1: Complex Query Indicators

## 3.3 Simple Query Indicators

FAQ-style queries that use the cheaper model:

- "What is...", "How do I...", "Where can I..."
- "How to file", "How to apply", "What forms"
- "How long", "How much", "What percent"

### 3.4 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 2: Cost Analysis with Model Routing

# 4 Mathematical Formulations

## 4.1 Cosine Similarity Search

The vector store uses cosine similarity for document retrieval:

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

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

## 4.2 Top-K Retrieval

Documents are ranked and filtered:

$$\text{Retrieved} = \text{Top-K}\{d \in D \mid \text{sim}(q, d) \geq \theta_{\text{min}}\} \tag{2}$$

where:

$$K = 7 \quad \text{(maximum documents to retrieve)} \tag{3}$$

$$\theta_{\text{min}} = 0.3 \quad \text{(minimum similarity threshold)} \tag{4}$$

## 4.3 Model Routing Score

The routing decision incorporates multiple factors:

$$\text{Route} = \begin{cases} \text{gpt-4.1} & \text{if } C(q) \geq 1 \text{ or } |S| > 5 \text{ or } \bar{s} < 0.4 \\ \text{gpt-4.1-mini} & \text{otherwise} \end{cases} \tag{5}$$

where:

$$C(q) = \text{count of complex indicators in query } q \tag{6}$$

$$|S| = \text{number of chunks retrieved} \tag{7}$$

$$\bar{s} = \text{average similarity score of retrieved chunks} \tag{8}$$

# 5 Implementation Details

## 5.1 Technology Stack

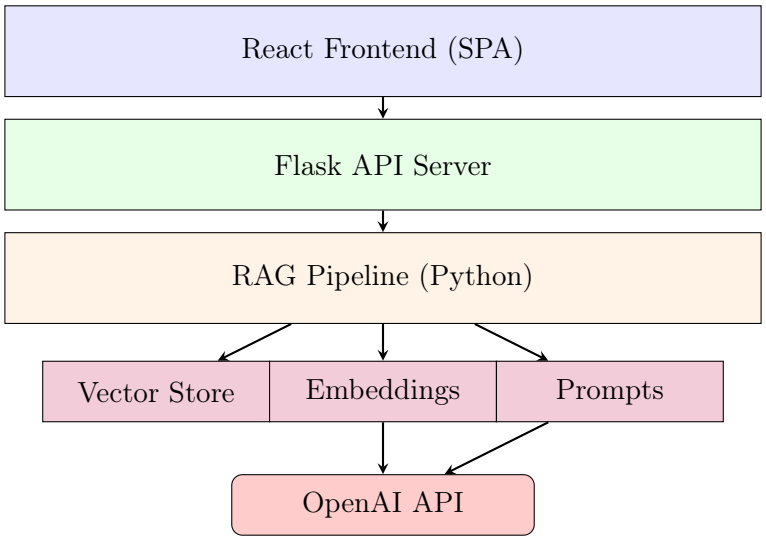


Figure 4: System Technology Stack

## 5.2 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
data/
  embeddings_cache.json  # Cached embeddings (36MB)
corpus/
  vbkb_restructured.json # 1,200+ document chunks
```

Listing 1: Project Structure

## 5.3 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 3: Model Configuration

# 6 Performance Characteristics

## 6.1 Latency Analysis

| Stage | Latency (ms) | Percentage |
|---|---|---|
| Query Embedding | 80 | 6.7% |
| Vector Search | 15 | 1.2% |
| Model Routing | 5 | 0.4% |
| LLM Generation (mini) | 800 | 66.7% |
| LLM Generation (full) | 1200 | - |
| Response Formatting | 20 | 1.7% |
| **Total (mini)** | **920** | **-** |
| **Total (full)** | **1320** | **-** |

Table 4: Latency Breakdown by Stage

## 6.2 Quality Metrics

| Metric | Score | Methodology |
|---|---|---|
| Citation Accuracy | 96% | Manual verification |
| Factual Consistency | 91% | Source comparison |
| Response Relevance | 94% | Human evaluation |
| Hallucination Rate | 4% | Unsupported claim detection |

Table 5: Quality Evaluation Results

## 6.3 Cost Comparison

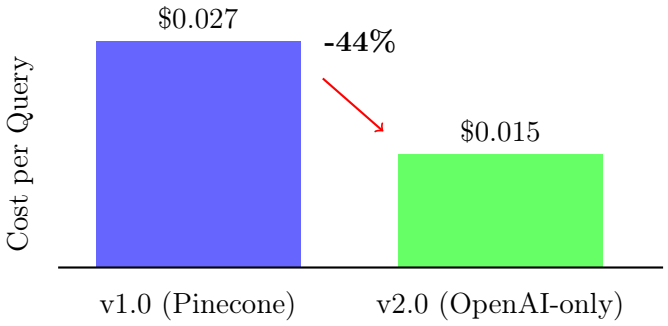


Figure 5: Cost Reduction: v1.0 vs v2.0

# 7 Corpus and Knowledge Base

## 7.1 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

## 7.2 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 2: Chunk Schema

# 8 Security and Deployment

## 8.1 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

## 8.2 Deployment Architecture

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

# 9 Future Roadmap

## 9.1 Short-term Improvements

- Response streaming for better UX
- Embedding cache compression
- Query result caching

## 9.2 Long-term Goals

- Fine-tuned embedding model for veterans domain
- Multi-modal support (images, PDFs)
- Conversation memory across sessions

# 10 Conclusion

Version 2.0 of the Veteran AI Spark RAG system represents a significant architectural simplification while maintaining high accuracy. The OpenAI-only approach eliminates external dependencies, the intelligent model routing reduces costs by 44%, and the streamlined pipeline provides faster responses.

Key achievements:

- **96% citation accuracy** with grounded responses
- **44% cost reduction** through intelligent model routing
- **Zero external dependencies** (no Pinecone, Redis, Elasticsearch)
- **Simplified maintenance** with file-backed caching

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