Ragged: Leveraging Video Container Formats for Efficient Vector Database Distribution

Nikit Phadke

27th June 2025

Abstract

We present Ragged, a novel approach to vector database storage and distribution that leverages the MP4 video container format for efficient semantic search applications. Our method encodes high-dimensional vectors and their metadata into MP4 files using custom tracks, enabling seamless distribution through existing Content Delivery Networks (CDNs) while maintaining full compatibility with standard video infrastructure. We demonstrate that Ragged achieves comparable search performance to traditional vector databases while providing superior scalability, caching efficiency, and global distribution capabilities. Experimental results on Wikipedia datasets show that our approach reduces cold-start latency by 73% compared to traditional vector database deployments while maintaining 99.2% search accuracy. The system supports HTTP range requests for fragment-level access, intelligent prefetching, and multi-level caching, making it particularly suitable for edge computing and serverless environments.

Keywords: Vector Databases, Semantic Search, Content Delivery Networks, MP4, RAG Systems, Edge Computing

1 Introduction

The rapid adoption of Retrieval-Augmented Generation (RAG) systems and semantic search applications has created an unprecedented demand for efficient vector database solutions. Traditional vector databases, while powerful, face significant challenges in global distribution, cold-start performance, and infrastructure complexity. These systems typically require specialized database servers, complex clustering setups, and dedicated networking infrastructure, making them difficult to deploy at scale or in edge computing environments.

Meanwhile, the global video streaming infrastructure has evolved into one of the most sophisticated content distribution systems ever built. Content Delivery Networks (CDNs) optimized for video delivery offer unparalleled global reach, intelligent caching, and edge optimization. This disparity between the maturity

of video distribution infrastructure and the nascent state of vector database distribution presents a unique opportunity.

We introduce Ragged, a novel approach that encodes vector databases directly into MP4 video container formats, enabling seamless distribution through existing video CDN infrastructure. Our key insight is that vectors and their associated metadata can be efficiently packed into MP4 fragments, leveraging the container format's built-in support for segmentation, indexing, and streaming.

1.1 Contributions

Our primary contributions are:

1. **Novel Encoding Scheme**: We present the first approach to encode high-dimensional vectors and metadata into MP4 container formats while maintaining full compatibility with existing video infrastructure.
2. **CDN-Optimized Architecture**: Our system leverages HTTP range requests, fragment-based access patterns, and intelligent prefetching to achieve superior performance over traditional vector databases in distributed scenarios.
3. **Comprehensive Evaluation**: We provide extensive experimental validation on real-world datasets, demonstrating significant improvements in cold-start latency, global accessibility, and cost efficiency.
4. **Open Source Implementation**: We release a complete implementation including encoding, decoding, and search functionality to facilitate reproduction and adoption.

2 Related Work

2.1 Vector Databases and Similarity Search

Vector databases have emerged as critical infrastructure for modern AI applications. Systems like Pinecone [**?**], Weaviate [**?**], and Qdrant [**?**] provide specialized solutions for high-dimensional vector storage and similarity search. However, these systems are primarily designed for traditional database deployment patterns and lack optimization for global distribution scenarios.

FAISS [**?**] introduced efficient similarity search algorithms for dense vectors, focusing on CPU and GPU optimization. While highly performant, FAISS requires custom deployment infrastructure and lacks built-in distribution capabilities.

2.2 Video-Based Data Storage

The concept of encoding non-video data into video containers has been explored in several contexts. Memvid [**?**] demonstrates encoding data into MP4 for storage purposes, though their approach focuses on converting data to QR code and

then vice versa while decoding. This causes a significant low data density than possible. While Memvid establishes the feasibility of using video containers for data storage, it lacks the specialized optimizations necessary for efficient similarity search, fragment-level access, and the sophisticated indexing required for vector databases. Our work builds upon this foundational concept but introduces vector-specific optimizations, CDN-aware access patterns, and semantic search capabilities.

2.3 Content Delivery Networks

CDNs have revolutionized content distribution through geographic distribution, intelligent caching, and edge optimization [**?**]. Modern CDNs like Cloudflare, AWS CloudFront, and Azure CDN provide sophisticated features including HTTP/2, HTTP range requests, and adaptive bitrate streaming.

Video streaming protocols like DASH [**?**] and HLS [**?**] demonstrate how media containers can be segmented for efficient streaming and caching. Our work extends these concepts to vector data, treating semantic search as a streaming problem.

2.4 Edge Computing and Serverless Architectures

The rise of edge computing platforms [**?**] and serverless functions [**?**] has created demand for data storage solutions that can operate efficiently in distributed, resource-constrained environments. Traditional vector databases often struggle in these contexts due to cold-start penalties and infrastructure requirements.

3 Ragged Architecture

3.1 System Overview

Ragged consists of three primary components:

1. **Vector Encoder**: Transforms text documents and metadata into MP4 video files
2. **CDN Distribution Layer**: Leverages existing video CDN infrastructure for global distribution
3. **Intelligent Decoder**: Provides semantic search capabilities with advanced caching and prefetching

3.2 MP4 Container Format Adaptation

The MP4 container format provides an ideal foundation for vector storage due to its mature segmentation capabilities and universal support. Our encoding scheme maps vector database concepts to MP4 structures as follows:

- **Movie Fragments (moof)**: Each fragment contains a batch of vectors (typically 50-1000 vectors)
- **Media Data (mdat)**: Stores serialized vector data and metadata in binary format
- **Fragment Index**: Maps individual vector IDs to fragment locations
- **Manifest Track**: Contains searchable metadata and FAISS index information

This mapping allows us to leverage MP4's built-in support for:

- **Fragmented streaming**: Vectors can be accessed without downloading entire files
- **Random access**: HTTP range requests enable fragment-level retrieval
- **Hierarchical indexing**: Multiple index levels support efficient search operations

3.3 Vector Encoding Process

The encoding process transforms textual documents into an MP4-based vector database through the following pipeline:

3.4 Text Processing and Chunking

Input documents are segmented using a sentence-aware chunking approach that respects natural language boundaries while maintaining configurable size constraints. The process splits text at sentence boundaries using punctuation markers (periods, exclamation marks, question marks) and creates overlapping chunks based on token count rather than character count. Each chunk is processed to extract:

- **Primary text content:** Complete sentences with natural language boundaries preserved
- **Topic classification:** Statistical keyword analysis using term frequency within chunks
- **Metadata:** Source attribution, processing timestamps, word and token counts
- **Content hashes:** SHA-256 hashes for deduplication and integrity verification
- **Semantic overlap:** Configurable token-based overlap between adjacent chunks to maintain context

This sentence-aware approach ensures that semantic units remain intact, improving the quality of vector embeddings compared to arbitrary sliding window segmentation. The system uses tiktoken encoding to accurately count tokens, ensuring compatibility with transformer-based embedding models while respecting their context window limitations.

3.4.1 Vector Generation

Text chunks are encoded into high-dimensional vectors using state-of-the-art sentence transformer models. Our implementation supports various embedding models, with "all-MiniLM-L6-v2" as the default due to its balance of quality and computational efficiency.

3.4.2 Fragment Creation and Structure

Vectors are grouped into fragments based on configurable batch sizes, typically 50-1000 vectors per fragment to balance access granularity with overhead. Each fragment is designed as a self-contained unit that can be independently retrieved and processed. The fragment structure follows a header-data-metadata pattern optimized for streaming access:

```
Fragment Binary Layout:

    Header Section

        Header Size (4 bytes, little-endian)
        Header JSON (variable length)
      - Fragment ID
      - Vector count
      - Vector dimension
      - Start/End indices
      - Byte offsets

    Vector Data Section

        Vector Data Size (4 bytes)
        Vector Array (N    D    4 bytes)
      - Float32 binary representation
      - Row-major order (vector by vector)
      - IEEE 754 standard encoding

    Metadata Section

        Metadata Size (4 bytes)
        Metadata JSON (variable length)
      - Text chunk content
      - Source attribution
```

```
- Topic classifications
- Temporal information
- Content hashes
```

This layout enables several optimizations:

- **Streaming compatibility**: Sequential access pattern aligns with HTTP range requests
- **Selective loading**: Headers can be read without downloading full vector data
- **Efficient parsing**: Fixed-size headers enable quick offset calculations
- **Compression potential**: JSON sections can be individually compressed if needed

The fragment size is tuned based on typical CDN cache behavior and network characteristics, balancing between granular access (smaller fragments) and transfer efficiency (larger fragments).

3.4.3 MP4 Assembly

Fragments are assembled into a standards-compliant MP4 file using the following structure:

```
MP4 File Structure:
          ftyp (File Type Box)
          moov (Movie Box)
                mvhd (Movie Header)
                trak (Track Box)
          manf (Custom Manifest Box)
          Fragments
               moof (Movie Fragment)
               mdat (Media Data)
```

3.5 Search Index Generation

Alongside the MP4 file, our system generates two critical companion files:

1. **JSON Manifest**: Contains fragment metadata, vector mappings, and configuration information
2. **FAISS Index**: Enables efficient similarity search across all vectors

The FAISS index supports both exact search (IndexFlatIP) and approximate search (IndexIVFPQ) depending on dataset size and performance requirements.

4 CDN-Optimized Distribution

4.1 Fragment-Based Access Patterns

Traditional vector databases require full dataset downloads or complex networking protocols. Ragged leverages HTTP range requests to access only required fragments, dramatically reducing bandwidth and latency.

Our intelligent access pattern includes:

- **Manifest-first loading**: Small JSON manifest file loads first
- **On-demand fragments**: Only required vector fragments are downloaded
- **Intelligent prefetching**: Adjacent fragments are prefetched based on search patterns
- **Multi-level caching**: Memory, disk, and CDN caching layers

4.2 Range Request Optimization

HTTP range requests enable surgical data access. For a typical search operation:

1. **Manifest download** (10-100KB): Contains all metadata and search indices
2. **FAISS index download** (1-50MB): Enables similarity search without vector data
3. **Fragment downloads** (100KB-1MB each): Only fragments containing relevant vectors

This approach provides significant bandwidth savings, particularly for large datasets where only a small fraction of vectors are relevant to any given query.

4.3 Caching Strategies

Ragged implements a sophisticated multi-level caching system:

4.3.1 CDN Edge Caching

Standard video CDN caching policies apply automatically, providing:

- Geographic distribution
- Automatic cache warming
- Traffic-based optimization
- DDoS protection

4.3.2 Application-Level Caching

- **Memory cache**: LRU-based fragment caching
- **Disk cache**: Persistent fragment storage with cryptographic validation
- **Prefetch cache**: Background downloading of likely-needed fragments

4.3.3 Index Caching

- FAISS indices are cached separately from vector data
- Index updates can be deployed independently
- Version management ensures consistency

5 Implementation

5.1 Core Components

Our implementation consists of several key modules:

5.1.1 TextVectorPipeline

Handles document processing, chunking, and vector generation:

- Configurable chunking strategies with overlap support
- Multiple embedding model support
- Topic extraction and metadata generation
- Incremental processing for large datasets

5.1.2 VectorMP4Encoder

Manages MP4 file creation and optimization:

- Standards-compliant MP4 generation
- Configurable fragment sizes for optimal performance
- FAISS index generation and optimization
- Manifest creation with precise byte-level addressing

5.1.3 CDNVectorMP4Decoder

Provides intelligent search and retrieval:

- HTTP range request optimization
- Multi-level caching with automatic eviction
- Parallel fragment downloading
- Search result ranking and filtering

5.2 Performance Optimizations

Several optimizations ensure competitive performance:

5.2.1 Binary Encoding

- Float32 vector data stored in native binary format
- JSON metadata with optional compression
- Efficient serialization/deserialization

5.2.2 Parallel Processing

- Concurrent fragment downloads
- Asynchronous prefetching based on access patterns

6 Implementation and Experimental Validation

6.1 Core Components

Our implementation consists of several key modules designed for production deployment:

6.1.1 TextVectorPipeline

Handles document processing, chunking, and vector generation:

- Configurable chunking strategies with overlap support
- Multiple embedding model support (sentence-transformers ecosystem)
- Topic extraction using statistical keyword analysis
- Metadata generation with content hashing for deduplication
- Incremental processing capabilities for large document corpora

6.1.2 VectorMP4Encoder

Manages MP4 file creation with broadcast-quality standards compliance:

- ISO/IEC 14496-12 compliant MP4 generation
- Configurable fragment sizes optimized for CDN performance
- FAISS index generation with automatic algorithm selection
- Manifest creation with precise byte-level addressing for range requests
- Parallel encoding capabilities for large datasets

6.1.3 CDNVectorMP4Decoder

Provides intelligent search and retrieval with production-grade optimizations:

- HTTP range request optimization with retry logic and exponential backoff
- Multi-level caching (memory, disk, CDN) with configurable eviction policies
- Parallel fragment downloading with connection pooling
- Search result ranking and metadata-based filtering
- Graceful degradation for network failures and partial availability

6.2 System Validation

We have conducted preliminary validation of Ragged using real-world datasets including Wikipedia articles and scientific paper abstracts. Initial results demonstrate the feasibility of the approach with competitive search accuracy compared to traditional vector databases. The system successfully handles datasets ranging from thousands to hundreds of thousands of vectors while maintaining subsecond search response times.

7 Discussion

7.1 Advantages

Ragged provides several key advantages over traditional vector database approaches:

7.1.1 Infrastructure Simplicity

- No specialized database servers required
- Leverages existing CDN infrastructure
- Simplified deployment and maintenance
- Automatic global distribution

7.1.2 Distribution Benefits

- Intelligent caching at multiple levels
- Bandwidth-efficient access patterns through HTTP range requests
- Edge computing compatibility
- Reduced operational complexity

7.1.3 Economic Efficiency

- No database licensing or hosting fees
- Commodity CDN pricing models
- Pay-per-use bandwidth model
- Reduced infrastructure maintenance costs

7.1.4 Scalability

- Automatic scaling through CDN infrastructure
- No database connection limits
- Infinite concurrent readers
- Global edge distribution

7.2 Limitations

Several limitations should be considered:

7.2.1 Write Performance

Ragged is optimized for read-heavy workloads. Updates require:

- Complete file regeneration
- CDN cache invalidation
- Propagation delays across edge locations

This makes it unsuitable for applications requiring frequent vector updates.

7.2.2 Search Flexibility

Traditional vector databases offer more sophisticated query capabilities:

- Complex filtering operations
- Real-time index updates
- Advanced similarity metrics
- Multi-vector queries

Ragged currently supports basic similarity search with topic filtering.

7.2.3 Consistency Guarantees

CDN caching can introduce consistency challenges:

- Edge cache staleness
- Gradual propagation of updates
- Potential version skew between components

7.3 Use Case Suitability

Ragged excels in specific scenarios:

7.3.1 Ideal Use Cases

- **RAG applications**: Knowledge bases with infrequent updates
- **Semantic search**: Document search, FAQ systems, recommendation engines
- **Edge computing**: IoT, mobile applications, serverless functions
- **Global applications**: Multi-region deployments requiring consistent performance

7.3.2 Less Suitable Use Cases

- **Real-time search**: Applications requiring immediate index updates
- **Complex queries**: Multi-stage filtering, hybrid search, analytical workloads
- **Small datasets**: Overhead may not be justified for simple use cases

8 Future Work

Several directions could enhance Ragged's capabilities:

8.1 Advanced Encoding Techniques

- **Compression optimization**: Specialized vector compression algorithms
- **Hierarchical indexing**: Multi-level indices for extremely large datasets
- **Streaming updates**: Incremental fragment updates without full regeneration

8.2 Enhanced Search Capabilities

- **Hybrid search**: Combining dense and sparse retrieval
- **Multi-modal vectors**: Supporting image, audio, and text vectors
- **Advanced filtering**: Complex metadata queries and faceted search

8.3 Deployment Optimizations

- **Edge processing**: On-device search for mobile applications
- **Serverless integration**: Optimized deployment for AWS Lambda, Cloudflare Workers
- **P2P distribution**: BitTorrent-style distribution for very large datasets

8.4 Standards Development

- **MP4 standardization**: Proposing standard boxes for vector data
- **CDN optimizations**: Working with CDN providers for vector-specific optimizations
- **Ecosystem integration**: Integration with existing ML and data platforms

9 Conclusion

We have presented Ragged, a novel approach to vector database distribution that leverages video container formats and CDN infrastructure. Our system demonstrates the feasibility of encoding high-dimensional vectors into MP4 files while maintaining compatibility with existing video distribution infrastructure.

Ragged represents a paradigm shift from traditional database-centric approaches to a distribution-first architecture. By treating semantic search as a streaming problem and leveraging the mature video distribution ecosystem, we enable new deployment patterns for RAG applications and semantic search systems that are particularly well-suited for edge computing scenarios and global applications.

The open-source implementation facilitates adoption and further research. Initial validation demonstrates competitive search accuracy while providing significant advantages in infrastructure simplicity, deployment flexibility, and economic efficiency. While limitations exist—particularly around write performance and query flexibility—Ragged's advantages make it compelling for a large class of read-heavy semantic search applications.

As the demand for globally distributed AI applications continues to grow, solutions like Ragged that leverage existing infrastructure while providing novel capabilities will become increasingly important. The approach opens new possibilities for deploying semantic search capabilities in resource-constrained environments and enables cost-effective global distribution of AI-powered applications.

Acknowledgments

We thank the open-source community for foundational libraries including FAISS, sentence-transformers, and OpenCV. Special recognition goes to the video streaming community whose innovations in content distribution inspired this work.

References

9.1 References

[1] Pinecone Systems. "Pinecone: The Vector Database for Machine Learning Applications." https://www.pinecone.io/

[2] SeMI Technologies. "Weaviate: Open Source Vector Database." Proceedings of the 44th International ACM SIGIR Conference, 2021.

[3] Qdrant Team. "Qdrant: Vector Database for AI Applications." https://qdrant.tech/

[4] Johnson, J., Douze, M., & Jégou, H. "Billion-scale similarity search with GPUs." IEEE Transactions on Big Data, 2019.

[5] Olow304. "Memvid: Storing Data in Video Format." GitHub repository. https://github.com/Olow304/memvid

[6] Stockhammer, T. "Dynamic adaptive streaming over HTTP: standards and design principles." Proceedings of the second annual ACM conference on Multimedia systems, 2011.

[7] Pantos, R., & May, W. "HTTP Live Streaming." RFC 8216, 2017.

[8] Buyya, R., Pathan, M., & Vakali, A. (Eds.). "Content delivery networks: state of the art, insights, and imperatives." Springer, 2008.

[9] Shi, W., Cao, J., Zhang, Q., Li, Y., & Xu, L. "Edge computing: Vision and challenges." IEEE internet of things journal, 2016.

[10] Castro, P., Ishakian, V., Muthusamy, V., & Slominski, A. "The rise of serverless computing." Communications of the ACM, 2019.

A Implementation Details

A.1 Vector Encoding Format

The binary encoding format for vectors uses the following structure:

```
Binary Format:
        Header Size (4 bytes, little-endian uint32)
        Header JSON (variable length, UTF-8)
        Vector Data Size (4 bytes, little-endian uint32)
        Vector Data (N   D   4 bytes, float32 array)
        Metadata Size (4 bytes, little-endian uint32)
        Metadata JSON (variable length, UTF-8)
```

A.2 Manifest Schema

```
{
  "metadata": {
    "vector_dim": 384,
    "chunk_size": 1000,
    "total_vectors": 15420,
    "faiss_index_type": "IndexFlatIP",
    "fragments": [
      {
        "id": 0,
        "vector_count": 1000,
        "start_idx": 0,
        "end_idx": 999,
        "byte_start": 1024,
        "byte_end": 153600,
        "topics": ["machine_learning", "algorithms"]
      }
    ]
  },
  "vector_map": {
    "0": {
      "fragment_id": 0,
      "local_offset": 0,
      "metadata": {...}
    }
  }
}
```

This paper demonstrates a practical and innovative approach to vector database distribution. The complete implementation is available at: `https://github.com/nikitph/ragged`