Building a Billion-Scale Vector Embeddings Dataset for Real-World ANN Benchmarking

Mentor: Jayjeet Chakraborty
Author: Prathamesh Devadiga

Abstract

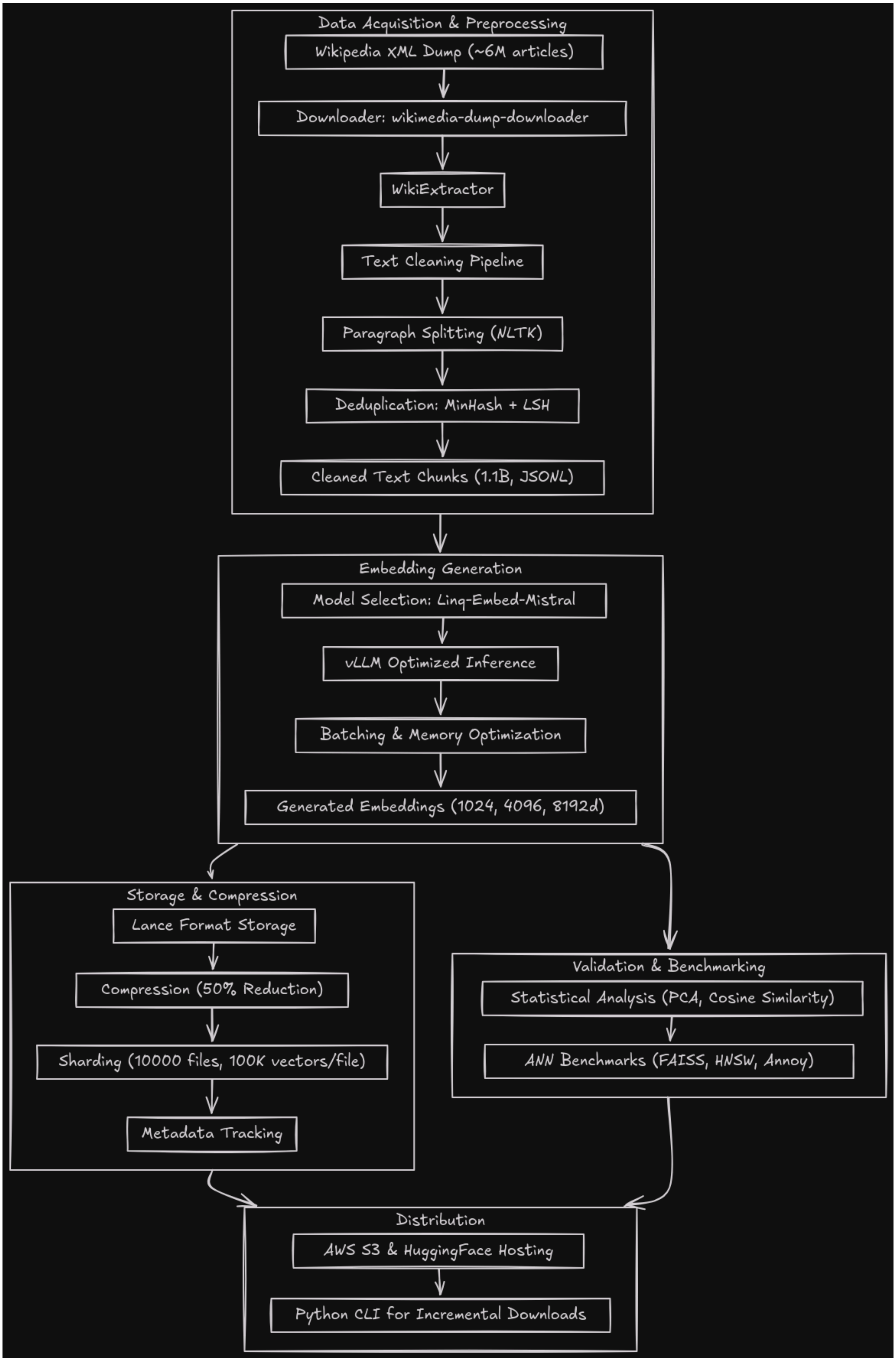
Current vector search benchmarks rely on synthetic or low-dimensional datasets (e.g., GloVe-25, SIFT-128), which fail to represent real-world workloads like those from modern LLMs (e.g., OpenAI's 3072d `text-embedding-3-large`). This project aims to create the **first open-source 1B-scale vector embedding dataset** using Wikipedia text processed through state-of-the-art open-source models, with three variants (1024, 4096, and 8192 dimensions). The dataset will enable realistic benchmarking of Approximate Nearest Neighbor (ANN) algorithms and empower research in retrieval-augmented generation (RAG) systems.

Problem Statement

- **Gap**: Existing ANN benchmarks (ANN-Benchmarks, BigANN) use small (≤1M samples) or synthetic data, lacking:
    - High dimensionality (>1000d)
    - Real-world text distributions
    - Scale (>100M vectors)
- **Impact**: Researchers and companies currently rely on proprietary datasets, hindering reproducibility and fair algorithm comparisons.

Project Goals

1. Generate **1 billion text embeddings** from English Wikipedia using open-source models.
2. Provide **multiple embedding dimensions** (1024, 4096, 8192, etc) to study dimensionality's impact on ANN performance.
3. Ensure **deduplication**, **compression**, and **metadata tracking** for usability.
4. Validate embeddings via statistical analysis and ANN benchmarks (FAISS/HNSW).
5. Distribute the dataset efficiently via sharded cloud storage and documentation.


Data Acquisition & Preprocessing
Wikipedia XML Dump (~6M articles)
Downloader: wikimedia-dump-downloader
WikiExtractor
Text Cleaning Pipeline
Paragraph Splitting (NLTK)
Deduplication: MinHash + LSH
Cleaned Text Chunks (1.1B, JSONL)
Embedding Generation
Model Selection: Linq-Embed-Mistral
vLLM Optimized Inference
Batching & Memory Optimization
Generated Embeddings (1024, 4096, 8192d)
Storage & Compression
Lance Format Storage
Compression (50% Reduction)
Sharding (10000 files, 100K vectors/file)
Metadata Tracking
Validation & Benchmarking
Statistical Analysis (PCA, Cosine Similarity)
ANN Benchmarks (FAISS, HNSW, Annoy)
Distribution
AWS S3 & HuggingFace Hosting
Python CLI for Incremental Downloads

Methodology

1. Data Acquisition & Preprocessing

- **Source**: Latest English Wikipedia dump (XML) → ~6M articles.
- **Tools**:
    - `wikimedia-dump-downloader` for XML retrieval.
    - `wikiextractor` to strip markup and extract plaintext.
- **Cleaning Pipeline**:
    - Regex-based removal of non-ASCII chars, tables, citations.
    - Paragraph splitting via `nltk.tokenize` (1 embedding/paragraph).
    - Deduplication using **MinHash + LSH** (LSHForest) to remove near-identical chunks.
- **Output**: ~1.1B cleaned text chunks stored in JSON Lines format.

Why MinHash and LSH?

- **MinHash**: A probabilistic algorithm to estimate the similarity between two sets (example could be b/w paragraphs). It works by:
    - Hashing elements of each set (words in a paragraph).
    - Selecting the minimum hash value for each set.
    - Comparing the fraction of matching minimum hashes to estimate similarity (Jaccard Index).
- **LSH (Locality-Sensitive Hashing)**: Groups similar items into buckets using hash functions. It works by:
    - Applying multiple hash functions to each MinHash signature.
    - Placing items with matching hashes into the same bucket.
    - Ensuring that similar items are likely to collide in the same bucket.
- **Why Use Them?**:
    - **Efficiency**: MinHash reduces the complexity of comparing billions of paragraphs.
    - **Scalability**: LSH groups similar paragraphs for deduplication without pairwise comparisons.
    - **Accuracy**: Ensures near-duplicate paragraphs (e.g., boilerplate text) are removed, improving dataset quality.

---

2. Embedding Generation

Model Selection

After careful evaluation of various embedding models, I've selected **Linq-AI-Research/Linq-Embed-Mistral** (subject to change as per experiments) as the optimal model for this project based on:

- **Performance**: Consistently ranks in the top positions on the MTEB leaderboard for retrieval and semantic similarity tasks
- **Efficiency**: Offers an excellent balance between embedding quality and computational requirements
- **Open Source**: Fully open-source model available on Hugging Face, ensuring reproducibility
- **Community Adoption**: Widely used in production RAG systems and retrieval applications

With the single-model approach, we can refine our infrastructure strategy:
- Deploy optimized vLLM configurations specifically tuned forLinq-AI-Research/Linq-Embed-Mistral.
- Implement model-specific batching strategies that maximize throughput
- Optimize memory usage patterns based on the model's specific characteristics
- Apply tailored quantization techniques appropriate for this embedding model

---

3. Storage & Compression

- **Format**: Lance (50% size reduction).
- **Sharding**: Split into 10,000 files (100K vectors/file) for partial downloads.
- **Metadata**: Track model versions, text source URLs, and processing timestamps.

---

4. Validation

- **Statistical Tests**:
  - PCA variance analysis (≥80% variance in ≤20% dimensions).
  - Cosine similarity distribution checks.
- **ANN Benchmarks**:
  - Recall@10 tests on FAISS-IVF, HNSW, and Annoy.
  - Query latency profiling on GPU/CPU platforms.

---

5. Distribution

- **Hosting**: AWS S3 (public bucket)+ HuggingFace
- **Tooling**: Python CLI for incremental downloads. (extra idea)
- **Documentation**: Tutorials for loading shards, reproducing results, and extending to new models.

Timeline

Phase 1: Data Acquisition & Preprocessing (Weeks 1-3)

- Week 1:
  - Set up development environment and version control
  - Implement Wikipedia dump downloader with monitoring
  - Create initial text extraction pipeline using wikiextractor
  - Set up CI/CD for continuous testing

- Week 2:
  - Implement text cleaning pipeline with regex and NLTK
  - Build paragraph splitting and normalization logic
  - Develop and test initial MinHash implementation
  - Create metrics for data quality assessment

- Week 3:
  - Implement full LSH-based deduplication system
  - Optimize MinHash + LSH for large-scale processing
  - Set up distributed processing for cleaning pipeline
  - Validate quality metrics on sample data

Phase 2: Embedding Generation (Weeks 4-7)

- Week 4:
  - Set up vLLM infrastructure on AWS for GPU optimization
  - Implement embedding generation pipeline for Linq-AI-Research/Linq-Embed-Mistral
  - Create benchmarking suite for throughput optimization
  - Develop sharding strategy for distributed computation

- Week 5-6:
  - Scale embedding generation to full Wikipedia corpus
  - Implement efficient batch processing strategies
  - Optimize memory usage for large-scale inference
  - Develop fallback mechanisms for handling failures

- Week 7:
  - Conduct quality assessment of generated embeddings
  - Implement dimension projection techniques (PCA, random projection)
  - Create derived datasets at different dimensionalities
  - Validate quality preservation across dimension transformations

Phase 3: Storage & Multi-Purpose Adaptation (Weeks 8-10)

- Week 8:
  - Implement Lance storage system with optimized compression
  - Design unified metadata schema across original and derived embeddings
  - Create efficient shard management system
  - Develop vector quality assessment tools

- Week 9-10:
  - Implement statistical validation suite for all embedding variants
  - Create ANN benchmarking framework for different dimensions
  - Develop specialized indices for different use cases
  - Document performance characteristics across dimensions

Phase 4: Evaluation & Distribution (Weeks 11-12)

- Week 11:
  - Conduct comprehensive benchmarking across all embedding variants
  - Evaluate performance in retrieval, classification, and clustering tasks
  - Create task-specific recommendation framework
  - Develop distribution tools and documentation

- Week 12:
  - Finalize dataset packaging for HuggingFace and AWS S3
  - Complete benchmark reports for all dimensionalities
  - Create interactive tutorials and examples
  - Prepare final documentation and submission

Expected Outcomes

1. Billion-Scale Vector Embeddings Dataset

- Complete Dataset: A comprehensive collection of 1 billion text embeddings derived from English Wikipedia, provided in three dimensionalities:
  - 1024-dimensional vectors
  - 4096-dimensional vectors
  - 8192-dimensional vectors

- Quality Assurance: Each embedding set undergoes rigorous deduplication, normalization, and statistical validation to ensure research-grade quality.
- Metadata Enrichment: Comprehensive metadata including source text, paragraph context, article titles, URL identifiers, and processing timestamps to enhance usability.

2. Reproducible Pipeline & Tooling

- End-to-End Processing Framework: A fully documented, modular pipeline for Wikipedia text extraction, cleaning, and embedding generation.
- Efficient Storage System: Implementation of Lance-based storage with optimized compression and sharding, reducing storage requirements by >50% compared to raw formats.
- CLI Tools: User-friendly command-line tools for dataset exploration, partial downloads, and custom embedding generation.

3. Comprehensive Benchmarking Suite

- ANN Algorithm Evaluation: Detailed performance analysis of leading vector search algorithms (FAISS-IVF, HNSW, ScaNN) across all three dimensionalities.
- Scaling Reports: Documentation of throughput, recall, and latency characteristics at varying index sizes (10M, 100M, 1B vectors).
- Hardware Profiling: Benchmarks across different hardware configurations (CPU, GPU, memory constraints) to guide real-world deployment decisions.
- RAG Performance Analysis: Evaluation of retrieval quality for question-answering tasks, demonstrating practical applications in retrieval-augmented generation systems.

4. Community Resources

- Interactive Documentation: Comprehensive guides, tutorials, and Jupyter notebooks demonstrating dataset usage.
- Academic Paper: Submission-ready research paper documenting methodology, statistical properties, and benchmark results.
- Extension Framework: Guidelines and tools for extending the dataset with new embedding models or data sources.

Open Source Impact

- **ANN Libraries**: FAISS, HNSWlib, and ScaNN can use this dataset to improve benchmarks.
- **Research**: Enables studies on high-dimensional ANN scalability and RAG optimization.
- **Sustainability**: Compressed/sharded design reduces access barriers for low-resource teams.

About Me

I am Prathamesh Devadiga, currently pursuing a Bachelor of Technology in Computer Science Engineering at PES University, Bangalore (2022-2026). My academic focus includes Data Structures, Algorithms, Machine Learning, Deep Learning, Operating Systems, Big Data, and Databases

Relevant Experience

- Lead Researcher & Founder, Adhāra AI Labs: Leading research in Machine Learning with a focus on Generative AI, Large Language Models, and Retrieval-Augmented Generation. Leading cross-functional teams to translate research into production applications. [https://aadhara-ai-labs.vercel.app/]

- AI Engineer Intern, IndhicAI: Contributing to customized Gen-AI workflows and pipelines, providing technical advisory for AI readiness evaluations, and researching ML/DL/NLP pipelines for end-to-end implementation.

- Research Intern, IIT Indore: Architected and implemented KASPER (Kernel Adaptive Spline-based PDF Attack Recognizer), a novel deep learning framework achieving 98.9% accuracy in PDF malware detection with robust defense against adversarial attacks.

- Summer Intern, The Innovation Lab (formerly, Microsoft Innovation Lab): Architected and fine-tuned a Mistral 7B model to create a multi-agent system comprising a code optimizer, reviewer, and test case writer, significantly enhancing automation and efficiency in code assessment. Achieved high evaluation scores (CodeBLEU: 80) for code review and generation, demonstrating effective AI-driven code analysis and optimization.

Relevant Projects

- Medical RAG: Developed a Retrieval-Augmented Generation (RAG) application for querying medical documents using vector search and semantic retrieval techniques with optimized embedding models**.**

- PyraFuseNet: Designed a dual-path network architecture for resource-constrained vision applications, achieving state-of-the-art accuracy with 55% fewer computations compared to ResNet-18 (accepted at ICIAI NTU Singapore).

- CoDSPy: Built an AI-powered code optimization system using DSPy and Gradio, implementing Chain-of-Thought and ReAct reasoning techniques for comprehensive code optimization.

- **E-Commerce Analytics:** Developed a real-time data processing system using Apache Flink to handle large-scale streaming data with PostgreSQL and Elasticsearch integration.

Technical Skills Relevant to This Project

- Languages & Libraries: Python, Go, SQL, Bash, PyTorch, Hugging Face, LangChain, LlamaIndex, Apache Spark, vLLM, Lance and DataSketch.

- Data Processing: Apache Spark, Kafka, and large-scale data pipelines

- Cloud & Infrastructure: AWS, Docker, and Kubernetes for scalable deployments

- Research Experience: Published paper at ICIAI NTU Singapore and under-review paper at IJCNN, showing expertise in computational efficiency and architecture optimization relevant to large-scale vector processing

Why I'm Ideal for This Project

My combined experience in deep learning research, vector embeddings work with RAG systems, and large-scale data processing makes me well-equipped to tackle the challenges of creating a billion-scale embedding dataset. I have hands-on experience with the exact embedding models proposed in this project, and my background in distributed systems will be crucial for the high-performance computing aspects of the work. As an active contributor to open-source AI projects and participant in the Oxford Machine Learning School 2024 and AWS AI-ML Scholar Program, I bring both technical expertise and a collaborative approach to open science that aligns perfectly with the goals of this GSoC project.