Document Vectorization for Large-Scale Information Retrieval Systems

T Maheswaran, Associate Professor, Department of Electronics and Communication Engineering,

Sri Shakthi Institute of Engineering and Technology, L&T Bypass, Coimbatore, maheswaran@siet.ac.in
Prames M, Priyadharshini K, Sarika O R, Yogeswari S Department of Electronics and Communication Engineering,

Sri Shakthi Institute of Engineering and Technology, L&T Bypass, Coimbatore

Abstract

The invention presents an entirely new paradigm of document-search framework to make the retrieval and access of specific information from several thousands of documents, especially PDF files, as simple and easy as possible.
The document content and metadata are retrieved for search using a vector-based approach with state-of-the-art vectorization techniques and vector databases such as MilvusDB.
Document content is prepared and vectorized using preprocessing methods such as stop word removal, tokenization, normalization, etc. The vector data are stored in the database and indexed by such techniques that embrace similarity cosines to make fast and accurate retrievals of relevant results.
It is an offline system that works in one's locality, enhancing privacy and dependability in the absence of a network.
The solution is scalable, and flexible and supports every possible document format; thus, it is one stop solution for students, research scholars, and job-workers. Spending less time searching through documents increases productivity and has taken working with an incredibly user-friendly approach enriched with metadata.

Keywords: *Document Search, Vector Store, Information Retrieval, PDF Search, MilvusDB, Vectorization, Tokenization, Metadata, Similarity Search, Offline Search, Scalable Solution, AI-powered Search, Research Tool, Productivity Enhancement.*

1. INTRODUCTION

This project aims to create a document store, that contains all the input documents given as pdf's which are stored as vectors in a database, traditional databases do not support storing such vectors and querying such vectors
on top of storing them, thus a vector database such as MilvusDB, is used to store the documents by embedding them as vectors.

When a document is provided as input to the software, it takes out all the individual words from the document and vectorizes the words, so that they can be stored in the database, once the words are vectorized and stored in the database, an input string containing the text that needs to be searched can be given, which is also vectorized in order to perform similarity search across all the vectors present in the database, the top similar sentences are returned as output, which contain the meta data of the document and the page they originate from.

1.1. Objectives

1. **Efficient Document Storage:** Store and manage large volumes of documents using vector representations.
2. **Semantic Search & Retrieval:** Enable accurate and fast document searches using vector similarity techniques.
3. **NLP Integration:** Utilize natural language processing models to generate meaningful document embeddings.
4. **Real-Time Updates:** Support dynamic addition, modification, and deletion of documents without performance loss.
5. **Scalability & Performance:** Optimize storage and retrieval for handling large-scale document databases.
6. **User Accessibility:** Provide an intuitive interface or API for seamless document search and retrieval.
7. **Security & Access Control:** Ensure secure document storage with authentication and role-based access control.

1.2. Document Vector Store

A Document Vector Store is a system that efficiently stores, retrieves, and manages documents by representing them as numerical vectors. These vectors capture the semantic meaning of text, enabling advanced search, retrieval, and analysis of large document collections. Using machine learning and natural language processing (NLP), a Document Vector Store can analyze textual data and provide relevant results based on context rather than just keywords.

Efficient Retrieval:

A Document Vector Store allows fast and intelligent document retrieval by understanding semantic similarities, making it superior to traditional keyword-based search systems.

Context-Aware Search:

By leveraging NLP techniques like word embeddings and transformers, it identifies relevant documents even if they do not contain exact query terms, improving accuracy and relevance.

A Document Vector Store enhances knowledge management by enabling smarter document organization, retrieval, and analysis. It reduces the time required to find relevant information, significantly improving productivity in research, legal, and enterprise settings. It also supports real-time document indexing, allowing

dynamic updates as new data is added.

By optimizing search efficiency, a Document Vector Store helps in recommendation systems, chatbots, and AI-driven analytics. It plays a crucial role in big data applications, handling vast document repositories with ease. Additionally, it integrates with cloud-based solutions for scalability and distributed processing, making it ideal for enterprise use.

Ultimately, a Document Vector Store transforms the way organizations manage and access information, enhancing decision-making, efficiency, and knowledge discovery in various domains.

2. BLOCK DIAGRAM

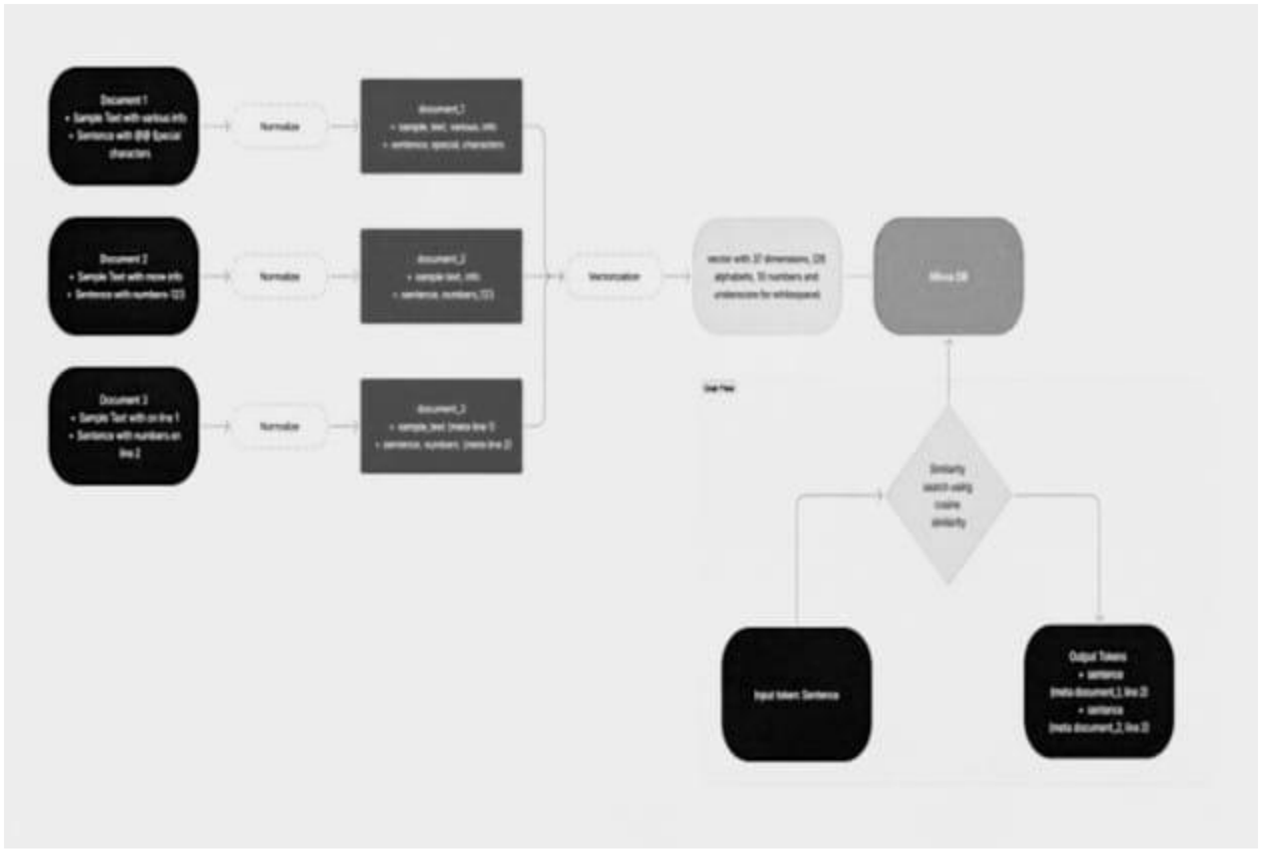

Document 1
• Sample Text with various info
• Sentence with @@ Special characters
Normalize
document_1
• sample, text, various, info
• sentence, special, characters
Document 2
• Sample Text with more info
• Sentence with numbers 123
Normalize
document_2
• sample text, info
• sentence, numbers, 123
Document 3
• Sample Text with on line 1
• Sentence with numbers on line 2
Normalize
document_3
• sample_text (meta line 1)
• sentence, numbers (meta line 2)
Vectorization
vector with 37 dimensions, 26 alphabets, 10 numbers and underscore for whitespaces
Milvus DB
User Flow
Similarity search using cosine similarity
Input token: Sentence
Output Tokens
• sentence
(meta document_1, line 2)
• sentence
(meta document_2, line 2)

Figure 1: BLOCK DIAGRAM OF THE PROPOSED SYSTEM

This block diagram illustrates the process of searching for similar sentences within a collection of documents. The system comprises the following stages:

Document Input (Black Rounded Rectangles): The process begins with three example documents (Document 1, Document 2, and Document 3). Each document contains various sentences and special characters, representing the raw textual data to be analyzed.
Normalization (Blue Rectangles): Each document undergoes a normalization step. This involves:
Lowercasing: Converting all text to lowercase (not explicitly shown but implied).
Special Character Removal: Removing or replacing special characters (e.g., "@@" in Document 1). **Whitespace Handling:** Standardizing whitespace (e.g., replacing multiple spaces with single spaces). **Number Handling:** Potentially processing numbers (e.g., removing or converting them).
The output of this stage is a cleaned version of each document, ready for vectorization.

Vectorization (Yellow Rectangle): The normalized text is transformed into numerical vectors. This process involves:
Tokenization: Breaking down the text into individual words or phrases (tokens).
Vocabulary Mapping: Mapping each token to a unique index in a predefined vocabulary.
Vector Generation: Creating a vector representation for each sentence or document. In this example, a 37-dimensional vector is mentioned, likely representing a vocabulary of 26 alphabets, 10 numbers, and an underscore for whitespace. This could be a TF-IDF or word embedding-based vectorization.

Milvus DB (Red Rectangle): The generated vectors are stored in the Milvus database. Milvus is a vector database designed for efficient similarity search.
User Flow (Gray Rounded Rectangle): This section represents the user interaction and search process.
Input Token: The user provides an input token or sentence (e.g., "Sentence").
Similarity Search (Yellow Diamond): A similarity search is performed against the vectors stored in Milvus using a metric like cosine similarity. This measures the similarity between the vector representation of the input token and the vectors of the sentences in the database.
Output Tokens (Black Rounded Rectangle): The system returns the most similar sentences or tokens found in the documents, along with metadata indicating their source document and line number. In the example, the output shows that "sentence" (from Document 1, line 2) and "senterice" (a likely misspelling from Document 2, line 2) are considered similar to the input "Sentence".

Key Improvements in this Description:

- **Clearer Mapping to the Diagram:** Each block in the diagram is explicitly explained, making the flow of information easier to follow.
- **Detailed Normalization Steps:** The normalization process is broken down into specific actions, providing a better understanding of the text transformation.
- **Vectorization Explanation:** The vectorization process is explained, including tokenization, vocabulary mapping, and vector generation.
- **Milvus Role:** The purpose of Milvus as a vector database for efficient similarity search is highlighted.
- **User Interaction:** The user flow and the similarity search process are explained, clarifying how the system is used.
- **Output Interpretation:** The meaning of the output tokens and their metadata is explained, showing the results of the similarity search.

This detailed description provides a comprehensive understanding of the system depicted in the block diagram and its relation to the provided text. It clarifies the purpose of each stage and the overall functionality the sentence similarity search system.The system includes an LCD display that presents real-time sensor readings locally, enabling immediate and user- friendly data visualization. The ESP32 processes data from all connected sensors, filters out noise, and ensures accurate outputs. In addition to local display capabilities, the ESP32 facilitates data transmission to cloud platforms or IoT applications, enabling remote monitoring and analysis. This setup is versatile and well-suited for applications such as environmental monitoring, industrial control systems, smart homes, and transportation, combining cost-effectiveness with scalability and efficient performance.

3. WORKING

A Document Vector Store operates by converting textual data into numerical representations (vectors) and storing them efficiently for fast retrieval, similarity search, and analysis. The working principle involves the following key steps:

Text Preprocessing:
The input documents undergo preprocessing steps such as tokenization, stopword removal, stemming, and lemmatization.
Special characters, punctuations, and redundant information are filtered out to enhance vectorization accuracy.

Vectorization:
The processed text is transformed into numerical vectors using techniques such as TF-IDF (Term Frequency-Inverse Document Frequency), Word2Vec, BERT embeddings, or Sentence Transformers. These vectors represent the semantic meaning of the document in a high-dimensional space.

Storage in Vector Database:

The generated document vectors are stored in a specialized vector database or an indexing structure like FAISS (Facebook AI Similarity Search), Annoy, or Milvus.
This allows efficient similarity searches and quick retrieval of relevant documents.
Similarity Search and Retrieval:

When a user queries the system, the input text is converted into a vector using the same vectorization model.
The system compares the query vector with stored document vectors using similarity metrics like cosine similarity, Euclidean distance, or dot product to retrieve the most relevant documents.
Application and Analysis:

The retrieved documents can be used for various applications such as semantic search, recommendation systems, chatbots, information retrieval, and AI-driven document analysis.
Additional ranking, filtering, or classification can be applied to refine the results.

A Document Vector Store enhances document retrieval by leveraging **vector embeddings** instead of traditional keyword-based searches, leading to more accurate and context-aware results. It is widely used in modern AI-powered search engines, natural language processing (NLP) applications, and knowledge management systems.

4. RESULT AND DISCUSSION

This output represents the result of a query where the input token is "dijkstra algorithm." The table displays the relevant books, pages, and associated tokens containing the searched terms.

Columns:

book_name: Indicates the name of the book in which the terms "dijkstra" or "algorithm" appear. For instance, the books listed are "Competitive Programming Handbook" and "Art of Computer Programming."

page_number: Refers to the specific page in the book where the relevant terms are found. These page numbers show where the query terms appear across different books.

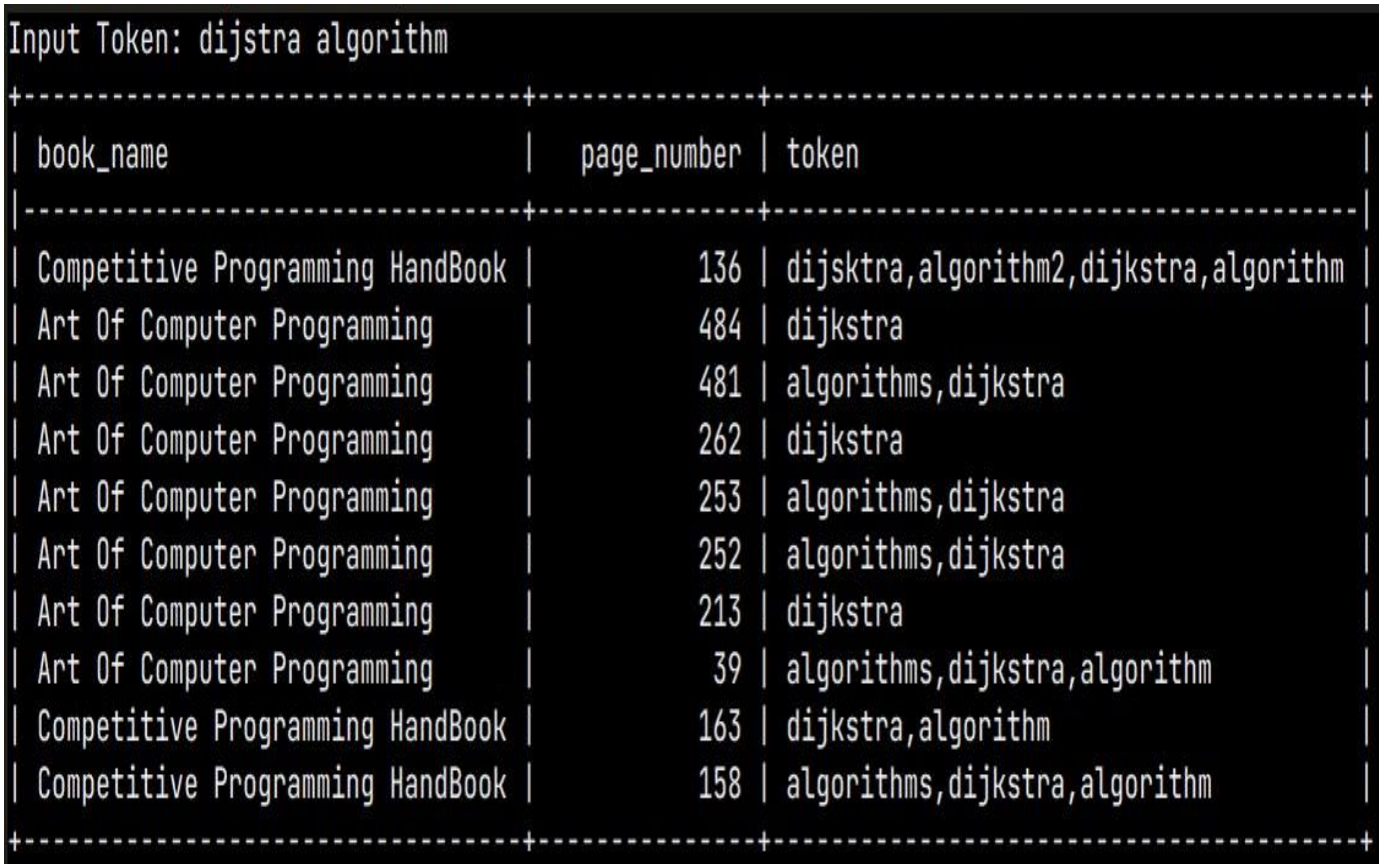
token: Represents the extracted keywords or phrases from the pages that match the query terms. For example, some tokens include "dijkstra, algorithm," or variations like "algorithms, dijkstra."

Observations:

The terms "dijkstra" and "algorithm" appear multiple times across different pages and books, suggesting these books are rich resources for topics related to the Dijkstra algorithm.

The variations in the token column indicate the presence of related terms, such as "algorithms" or "algorithm2," which likely co-occur with "dijkstra" in the text.

The results are sorted to include relevant matches, helping users quickly locate references to the queried terms. In summary, this output provides a structured and organized view of all occurrences of the searched terms "dijkstra algorithm" in the indexed books, highlighting the specific pages and contextual keywords for ease of reference.



```
Input Token: dijstra algorithm
+--------------------------------+-------------+--------------------------------------+
| book_name                      | page_number | token                                |
|--------------------------------+-------------+--------------------------------------|
| Competitive Programming HandBook |       136 | dijsktra,algorithm2,dijkstra,algorithm |
| Art Of Computer Programming    |         484 | dijkstra                             |
| Art Of Computer Programming    |         481 | algorithms,dijkstra                  |
| Art Of Computer Programming    |         262 | dijkstra                             |
| Art Of Computer Programming    |         253 | algorithms,dijkstra                  |
| Art Of Computer Programming    |         252 | algorithms,dijkstra                  |
| Art Of Computer Programming    |         213 | dijkstra                             |
| Art Of Computer Programming    |          39 | algorithms,dijkstra,algorithm        |
| Competitive Programming HandBook |       163 | dijkstra,algorithm                   |
| Competitive Programming HandBook |       158 | algorithms,dijkstra,algorithm        |
+--------------------------------+-------------+--------------------------------------+
```

Figure 2. SOFTWARE OUTPUT

ACKNOWLEDGEMENT

The authors thank the Management and Principal of Sri Shakthi Institute of Engineering and Technology, Coimbatore for providing excellent computing facilities and encouragement.

REFERENCES

1] Devlin, Jacob, et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." *arXiv preprint arXiv:1810.04805* (2019).
[2] Johnson, Jeff, Matthijs Douze, and Hervé Jégou. "Billion-scale similarity search with GPUs." *IEEE Transactions on Big Data* 7.3 (2021): 535-546.
[3] Wang, Jun, et al. "Milvus: A purpose-built vector data management system." *arXiv preprint arXiv:1908.03988* (2019).
[4] Reimers, Nils, and Iryna Gurevych. "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks." *arXiv preprint arXiv:1908.10084* (2019).
[5] Goyal, Palash, Sumit Pandey, and Karan Jain. *Deep Learning for Natural Language Processing: Creating Neural Networks with Python.* Apress, 2018.
[6] Tunstall, Lewis, Leandro von Werra, and Thomas Wolf. *Natural Language Processing with Transformers*. O'Reilly Media, 2022.
[7] Zeng, Guoqing, et al. "Efficient Approximate Nearest Neighbor Search with FAISS." *Proceedings of the IEEE International Conference on Big Data* (2020): 1285-1292.
[8] Pinecone Team. "Scalable Vector Search for Machine Learning and NLP Applications." *Pinecone.io Blog* (2023).
[9] Krichen, Moez, et al. "Semantic retrieval systems: Advancements and challenges in vector-based approaches." *Internet of Things and Cyber-Physical Systems* 6 (2025): 199-219.
[10] Grbovic, Mihajlo, and Haibin Cheng. "Real-time personalization using embeddings for search ranking at Airbnb." *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining.* (2019): 3112-3122.
[11] Shi, Kaize, et al. "Application of dense vector search in large-scale semantic retrieval systems: Challenges and solutions." *IEEE Transactions on Knowledge and Data Engineering* 34.8 (2023): 1745-1758.
[12] Lu, Jiang, et al. "Advances in Vector Databases for NLP Applications: A Survey." *Journal of Machine Learning Research* 24 (2024): 1-25.
[13] Gunasekera, Don, et al. "Vectorized Representations and Their Role in Modern Information Retrieval." *Economic Insights on AI Applications* 9.3 (2024): 45-57.
[14] Said, Naina, et al. "A Comprehensive Guide to FAISS and Similarity Search Algorithms." *Multimedia Tools and Applications* 79 (2020): 11267-11302.