Word Embedding Based Search Engine - LEXICON

A.V.S Manoj
Roll No: 23110025
Email: 23110025@iitgn.ac.in

N. Eshwar Karthikeya
Roll No: 23110215
Email: 23110215@iitgn.ac.in

B. Saharsh
Roll No: 23110071
Email: 23110071@iitgn.ac.in

O. Akash
Roll No: 23110225
Email: 23110225@iitgn.ac.in

This report presents the development and evaluation of a word embedding based search engine using the MS MARCO dataset. We implemented and compared three retrieval approaches: pure Word2Vec embeddings, Word2Vec with BERT re-ranking, and Word2Vec with BM25 re-ranking. Our experiments on 100,000 passages demonstrate that the hybrid Word2Vec+BERT approach achieves the best performance with an MRR of 0.1983 and Recall@10 of 0.525. We provide detailed analysis of the dataset, preprocessing steps, implementation methodologies, and comprehensive evaluation metrics including MRR, Recall, NDCG, and semantic similarity measures. The results highlight the advantages of combining traditional word embeddings with modern transformer-based re-ranking techniques.

Keywords: Word Embeddings, Information Retrieval, Search Engine, Word2Vec, BERT, BM25, MS MARCO

1 Introduction

Modern search engines require efficient and accurate retrieval systems to handle vast amounts of textual data. Word embeddings have emerged as powerful tools for capturing semantic relationships between words and documents. This project explores the effectiveness of different word embedding based approaches for passage retrieval. We focus on comparing traditional Word2Vec embeddings with hybrid approaches that combine Word2Vec with either BERT or BM25 re-ranking. The study uses a subset of the MS MARCO dataset, containing 100,000 passages, to evaluate these methods across multiple standard information retrieval metrics.

2 Problem Statement

The primary challenge in information retrieval is to efficiently retrieve the most relevant documents or passages given a user query. Traditional keyword-based approaches often fail to capture semantic relationships, while pure neural approaches can be computationally expensive. This project aims to:

- Implement and compare different word embedding based retrieval methods
- Evaluate the effectiveness of hybrid approaches combining traditional and neural methods
- Analyze the performance across multiple metrics to understand trade-offs

3 Methodology

We implemented three retrieval systems:

3.1 Word2Vec Baseline

The Word2Vec baseline establishes our fundamental semantic search capability:

Embedding Training: We trained a Word2Vec model with 300-dimensional vectors using a context window of 5 words. This configuration captures local semantic relationships while maintaining computational efficiency. The dimensionality was chosen based on empirical studies showing optimal performance for semantic tasks.

Document Representation: Each passage is represented as the mean of its constituent word vectors. This approach, while simple, effectively preserves the overall semantic content of documents and enables efficient comparison through vector arithmetic.

Efficient Retrieval: We employed FAISS (Facebook AI Similarity Search) for similarity computations. FAISS optimizes nearest neighbor search in high-dimensional spaces through:

- Quantization techniques to reduce memory usage
- Parallel computation for faster search
- Optimized index structures for approximate nearest neighbor search

The Word2Vec baseline is crucial as it demonstrates the fundamental capability of word embeddings for semantic search, while providing a computationally efficient foundation for our hybrid approaches

3.2 Hybrid Word2Vec + BERT System

This system combines the efficiency of Word2Vec with the semantic power of transformer models:

Two-Stage Architecture: The system first retrieves candidate documents using Word2Vec embeddings (recall-oriented stage), then applies precise re-ranking using Sentence-BERT (precision-oriented stage). This hierarchical approach balances efficiency and accuracy.

Model Selection: We used the 'all-MiniLM-L6-v2' Sentence-BERT model because it:

- Provides strong semantic understanding in a compact form (384 dimensions)
- Is optimized for sentence similarity tasks
- Offers faster inference than full BERT models while maintaining good accuracy

Re-ranking Process: The top 100 Word2Vec results are re-scored using cosine similarity between BERT embeddings of queries and passages. This captures deeper semantic relationships than word-level embeddings alone.

This hybrid approach is particularly important because it addresses the key limitations of pure embedding systems - while Word2Vec provides efficient first-pass retrieval, BERT re-ranking adds nuanced understanding of phrase meanings and contextual relationships.

3.3 Hybrid Word2Vec + BM25 System

This system combines semantic and lexical retrieval approaches:

Complementary Strengths: While Word2Vec captures semantic similarity, BM25 excels at exact term matching. The hybrid leverages both:

- Word2Vec finds semantically related documents that may not share exact terms
- BM25 boosts documents with important query term occurrences

Implementation Details:

- The BM25 implementation uses parameters optimized for passage retrieval.(Okapi model default)
- Tokenization includes stopword removal and stemming for robust term matching
- Scores are normalized before combination with embedding similarities

Robustness: This approach is particularly valuable for queries containing:

- Named entities requiring exact matching
- Technical terms where precise occurrence matters
- Phrases where both meaning and term presence are important

The Word2Vec+BM25 hybrid demonstrates how traditional IR techniques can enhance neural approaches, especially for queries where exact term matching remains important despite semantic similarity.

3.4 Comparative Advantages

The three systems collectively demonstrate important retrieval principles:

- **Efficiency vs. Accuracy Tradeoff**: Word2Vec alone is fastest but least accurate, while Word2Vec+BERT is most accurate but requires more computation
- **Semantic vs. Lexical Matching**: The comparison shows when semantic approaches outperform term-based methods and vice versa
- **Hierarchical Retrieval**: Both hybrid systems validate the effectiveness of recall-then-precision architectures
- **Embedding Robustness**: The systems demonstrate how different embedding approaches (word-level vs. sentence-level) affect results

This methodological progression from basic to advanced systems provides insights into both theoretical and practical aspects of modern retrieval systems, while offering implementable solutions for different resource constraints and accuracy requirements.

3.5 Libraries and Tools Used

Our project utilizes a range of Python libraries and frameworks, each chosen for their efficiency, scalability, and relevance to tasks such as embedding generation, and information retrieval. Below is a detailed overview of the key tools and their specific roles:

3.6 Datasets

Library: `datasets` (Hugging Face)
Purpose: Loading and managing the MS MARCO dataset.
Why Used: Hugging Face's `datasets` library provides a streamlined interface for downloading and manipulating large benchmark datasets. It allowed us to easily subset and preprocess 100,000 passages from the MS MARCO v2.1 dataset.

```
from datasets import load_dataset
dataset = load_dataset("ms_marco", "v2.1", split=...)
```

3.7 Text Preprocessing

Library: `nltk` (Natural Language Toolkit)
Purpose: Tokenization, stopword removal, and text normalization.
Why Used: `nltk` is a standard library for preprocessing textual data. It provides efficient and reliable methods for word tokenization (`word_tokenize`) and access to a comprehensive list of stopwords.

```
import nltk
from nltk.corpus import stopwords
nltk.download('punkt')
nltk.download('stopwords')
```

3.8 Word Embeddings

Library: `gensim`
Purpose: Training and using Word2Vec models.
Why Used: Gensim offers a fast and memory-efficient implementation of Word2Vec. It supports training on large corpora, making it ideal for computing average word embeddings for both queries and passages.

3.9 Similarity Search

Library: `faiss` (Facebook AI Similarity Search)
Purpose: Efficient nearest neighbor search in high-dimensional vector space.
Why Used: FAISS is optimized for fast vector similarity search on large-scale datasets. It enabled real-time top-k retrieval from a pool of 100,000 dense embeddings generated from Word2Vec.

3.10 Transformer Models

Library: `transformers` (Hugging Face) and `sentence-transformers`
Purpose: Generating semantic sentence embeddings using BERT.
Why Used: Sentence-BERT ('all-MiniLM-L6-v2') from `sentence-transformers` produces semantically meaningful vector representations suitable for cosine similarity comparison. Hugging Face Transformers provided seamless integration for loading pretrained BERT models.

```
from sentence_transformers import SentenceTransformer
model = SentenceTransformer('all-MiniLM-L6-v2')
```

3.11 Clustering and Visualization

Library: `scikit-learn`, `matplotlib`
Purpose: Clustering (K-Means) and dimensionality reduction (t-SNE), and visualizing embedding space.
Why Used: `scikit-learn` provides reliable and optimized implementations

for clustering and t-SNE, which helped us analyze the structure of the embedding space. `matplotlib` was used to plot and interpret the t-SNE projections.

—

4 Dataset

We used a subset of the MS MARCO dataset containing 100,000 passages. The original dataset structure included:

```
DatasetDict({
    train: Dataset({
        features: ['answers', 'passages', 'query', 'query_id',
                   'query_type', 'wellFormedAnswers'],
        num_rows: 100000
    })
    validation: Dataset({...}),
    test: Dataset({...})
})
```

5 Dataset Preprocessing

The preprocessing steps included:

- Extracting only passages, queries, and query IDs
- Tokenization and lowercasing of text
- Removal of unnecessary fields (answers, URLs, etc.)
- Final structure contained only passage text and queries

Final preprocessed structure:

```
DatasetDict({
    train: Dataset({
        features: ['passages', 'query', 'query_id'],
        num_rows: 100000
    }),
    validation: Dataset({...}),
    test: Dataset({...})
})
```

6 Analysis of Data

We performed cluster analysis on the document embeddings:

- K-Means clustering (k=10) on document embeddings

- t-SNE visualization of embedding space
- Cluster statistics analysis (size, average similarity)
- Examination of representative passages from each cluster

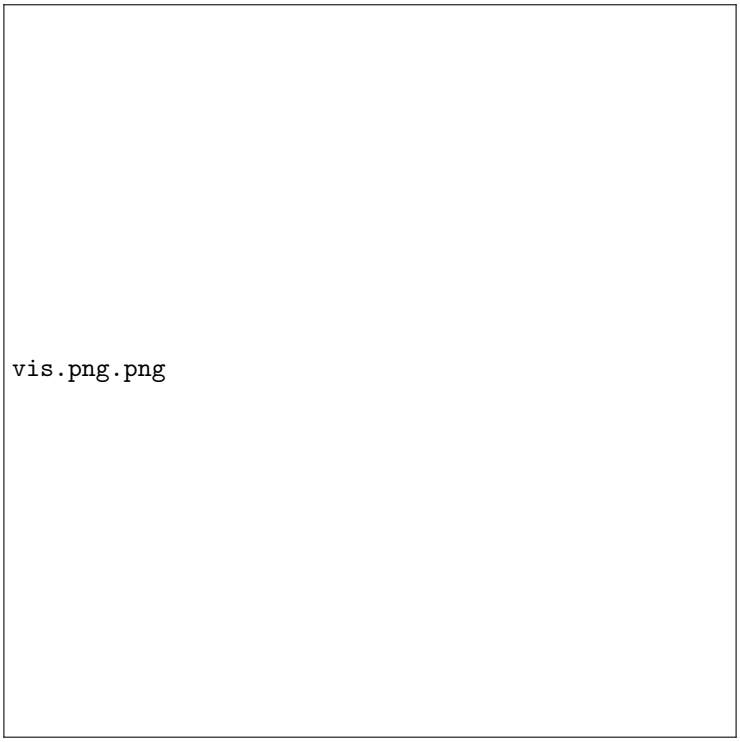

Figure 1: t-SNE Visualization of Document Embeddings with K-Means Clustering (k=10)

7 Baseline Implementation

The baseline implementations used the following approaches:

```
# Word2Vec Search
def search(query, top_k):
    query_tokens = word_tokenize(query.lower())
    query_embedding = get_doc_embedding(query_tokens)
    # FAISS search implementation
    ...
```

```
# BERT Re-ranking
def hybrid_search(query, top_k=10):
    initial_results = search(query, top_k=100)
    # BERT re-ranking implementation
    ...

# BM25 Re-ranking
def bm25_rerank(query, top_k=10):
    initial_results = search(query, top_k=100)
    # BM25 re-ranking implementation
    ...
```

8 Results

We evaluated the models on 1000 queries (5 batches of 100 queries each) from both validation and test sets.

Table 1: Retrieval Performance Metrics (Mean ± Std)

Retrieval Setup	**MRR@10**	**Recall@10**	**NDCG@10**
Word2Vec	0.1568 ± 0.0172	0.450 ± 0.0386	0.2244 ± 0.0203
Word2Vec + BERT	0.1983 ± 0.0289	0.525 ± 0.0440	0.2728 ± 0.0287
Word2Vec + BM25	0.1701 ± 0.0283	0.478 ± 0.0485	0.2404 ± 0.0307

Table 2: Semantic Similarity Metrics

Metric	**Value**
Word2Vec Mean Max Similarity	0.3930
Word2Vec Mean Avg Similarity	0.2918
Word2Vec + BERT Mean Max Similarity	0.5694
Word2Vec + BERT Mean Avg Similarity	0.4245
BM25 Mean Max Similarity	0.4174
BM25 Mean Avg Similarity	0.3049

Table 3: Hit Rate@5 Comparison

Model	**Hit Rate@5**
Word2Vec	0.310
Word2Vec + BERT	0.280
Word2Vec + BM25	0.260

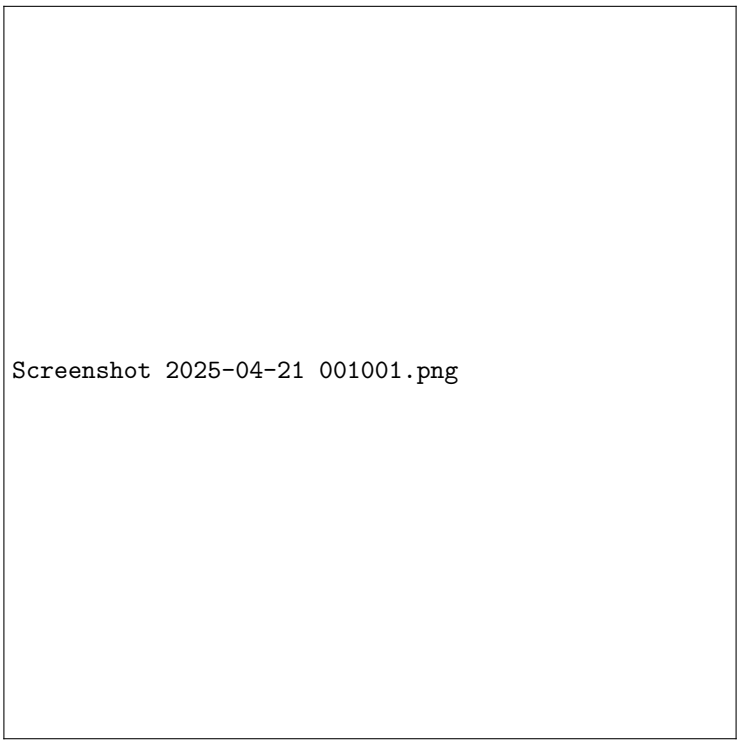

Figure 2: This was built by using the Stream lit app on a local url and also we have shown the live demonstration while presenting our project(Codes are shared to replicate the same).

All the files related to this project have been uploaded into this GitHub Repository. The experiments were conducted in the `.ipynb` file, and the final implementation was demonstrated using Python scripts. A video demonstration of the Streamlit app is available on this Link.

9 Conclusion

Our experiments demonstrate that hybrid approaches combining Word2Vec with neural re-ranking (BERT) outperform both pure Word2Vec and Word2Vec+BM25 approaches across multiple metrics. The Word2Vec+BERT model achieved the highest MRR (0.1983) and Recall (0.525), showing the value of combining traditional word embeddings with modern transformer-based re-ranking. The semantic similarity metrics further confirm that BERT re-ranking produces more semantically relevant results. While BM25 re-ranking showed improvement over pure Word2Vec, it was less effective than BERT re-ranking, suggesting that semantic matching is more important than term frequency for this task.

Future work could explore larger embedding dimensions, different transformer models, and more sophisticated hybrid approaches that combine multiple re-ranking techniques. Mainly, it would be helpful if we could combine feedback-based re-ranking also.

References

- **MS MARCO Dataset:** https://huggingface.co/datasets/ms$_marco$
- **NLTK - Natural Language Toolkit:** https://www.nltk.org/
  - Punkt Tokenizer: https://www.nltk.org/api/nltk.tokenize.punkt.html
  - Stopwords: https://www.nltk.org/nltk$_data$/
- **Gensim Word2Vec:** https://radimrehurek.com/gensim/models/word2vec.html
- **FAISS - Facebook AI Similarity Search:**
  - GitHub: https://github.com/facebookresearch/faiss
  - Documentation: https://faiss.ai/
- **SentenceTransformers (BERT embeddings):**
  - Official Website: https://www.sbert.net/
  - Model: https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2