Comparison of Deep Learning-Powered Search Engine with Traditional Search Engines

Krishanu Banerjee
AIML 'B'
44

Ambuj Shukla
AIML 'B'
52

Christie Mathews
AIML 'B'
53

***Abstract*—This document outlines a structured workflow for designing and implementing a deep learning-powered search engine. The workflow integrates web crawling, data preprocessing, embedding generation, vector indexing, query processing, and ranking to deliver relevant search results efficiently.**

I. Introduction

A search engine powered by deep learning (DL) offers superior retrieval accuracy by leveraging contextual embeddings and advanced ranking algorithms. This document provides a step-by-step workflow for building such a system.

II. Methodology

The workflow consists of the following steps:

A. Web Crawling (Data Collection)

A web crawler collects data from the web by extracting HTML content, including:

- Page title.
- Main content.
- Metadata (e.g., keywords, descriptions).
- URLs.

Potential Tools: Scrapy, BeautifulSoup, or custom scripts.

B. Preprocessing and Cleaning

Crawled data undergoes preprocessing:

- Removing unnecessary elements (scripts, ads, stylesheets).
- Tokenizing text into words or sentences.
- Lowercasing and removing stopwords.
- Stemming or lemmatization.

Potential Tools: Python libraries such as NLTK, spaCy, or transformers.

C. Embedding Generation (Vectorization)

Textual data is converted into dense numeric embeddings using pre-trained language models like BERT or Sentence-BERT. Each document is represented as a high-dimensional vector.

- Example: A sentence is transformed into a 768-dimensional vector using BERT.
- **Potential Tools:** Hugging Face Transformers.

D. Database and Indexing

Embeddings and metadata are stored in a vector database for fast retrieval. Popular options include:

- Pinecone.
- Weaviate.
- Milvus.
- FAISS.

E. Query Processing

User queries are processed as follows:

- Preprocessed similarly to document content.
- Converted to embeddings using the same DL model.

The vector database retrieves the most similar embeddings.

F. Ranking and Scoring

Retrieved documents are ranked using similarity measures like cosine similarity or dot product. Advanced re-ranking models or heuristics can improve relevance.

- **Example:** Cosine similarity ranks documents by measuring closeness between query and document embeddings.

G. Result Display

Top-ranked results, including metadata (e.g., title, URL, snippet), are displayed in a user-friendly format through a web interface.

H. Feedback Loop (Optional)

User interactions can be logged to fine-tune the ranking algorithm and improve search quality over time. Reinforcement learning can be employed for continuous optimization.