COIMBATORE INSTITUTE OF TECHNOLOGY

(Government Aided Autonomous Affiliated to Anna University, Chennai)

COIMBATORE – 641014, TAMILNADU, INDIA

M.Sc. ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

19MAMEL07 – GENERATIVE AI LAB

PROJECT

GENERATIVE SEARCH ENGINE FOR LOCAL FILE

NAME: SUSMITHA M

REG NO: 71762134051

ABSTRACT:

In an age where digital documents accumulate rapidly across diverse formats PDFs, Word files, presentations, and plain text accessing specific information when it's needed most can be both time-consuming and inefficient. Traditional keyword-based search methods often fall short in capturing the actual intent behind a query, especially when the answer lies deep within unstructured or context-heavy content.

This project introduces a **Generative Search Engine for Local Files**, a streamlined and privacy-first solution that combines the strengths of **semantic search** and **generative AI** to enable intelligent, question-driven exploration of user-uploaded documents. Users can interact with the system using natural language, receiving precise, contextually accurate answers grounded directly in their own files.

At the core of this solution is the **Sentence-Transformers** `msmarco-bert-base-dot-v5` model, which converts document content into dense semantic embeddings. These are indexed using **FAISS**, a highly efficient similarity search library, enabling rapid retrieval of document chunks most relevant to a user's query. The retrieved content is then passed to a local **Large Language Model (LLM)** such as **Gemma**, **Mistral**, or **LLaMA 2** which synthesizes a concise and context-aware response.

Unlike cloud-based solutions, this application is built to run entirely offline, ensuring user data remains secure and private. With a user-friendly interface powered by **Streamlit**, it supports a wide range of document types, allowing users to seamlessly upload files, index their content, and initiate queries all in a few clicks.

This tool is particularly beneficial for researchers, analysts, educators, and professionals who deal with large volumes of reference material. It effectively transforms static files into an **interactive knowledge base**, making information retrieval more intuitive, intelligent, and aligned with how we naturally seek answers.

By bridging traditional information retrieval with the capabilities of modern LLMs, this system paves the way for a smarter and more accessible approach to document understanding right from your desktop.

KEYWORDS: Embedding-Based Document Search, Vector Similarity Search Engine, FAISS-Powered Semantic Indexing, Context-Aware Answer Generation, Local Document Intelligence, AI-Powered Document Query System

PRE-TRAINED MODELS AND ARCHITECTURE:

1. Sentence Embedding Model

Model: `sentence-transformers/msmarco-bert-base-dot-v5`
Library: SentenceTransformers
Architecture: BERT (Bidirectional Encoder Representations from Transformers)
Fine-tuned on: MS MARCO (Microsoft MAchine Reading COmprehension) passage ranking dataset
Embedding Dimension: 768

Purpose:

- Converts text (from documents or user queries) into high-dimensional vector embeddings.
- Supports **dense retrieval** by mapping semantically similar content to nearby points in vector space.

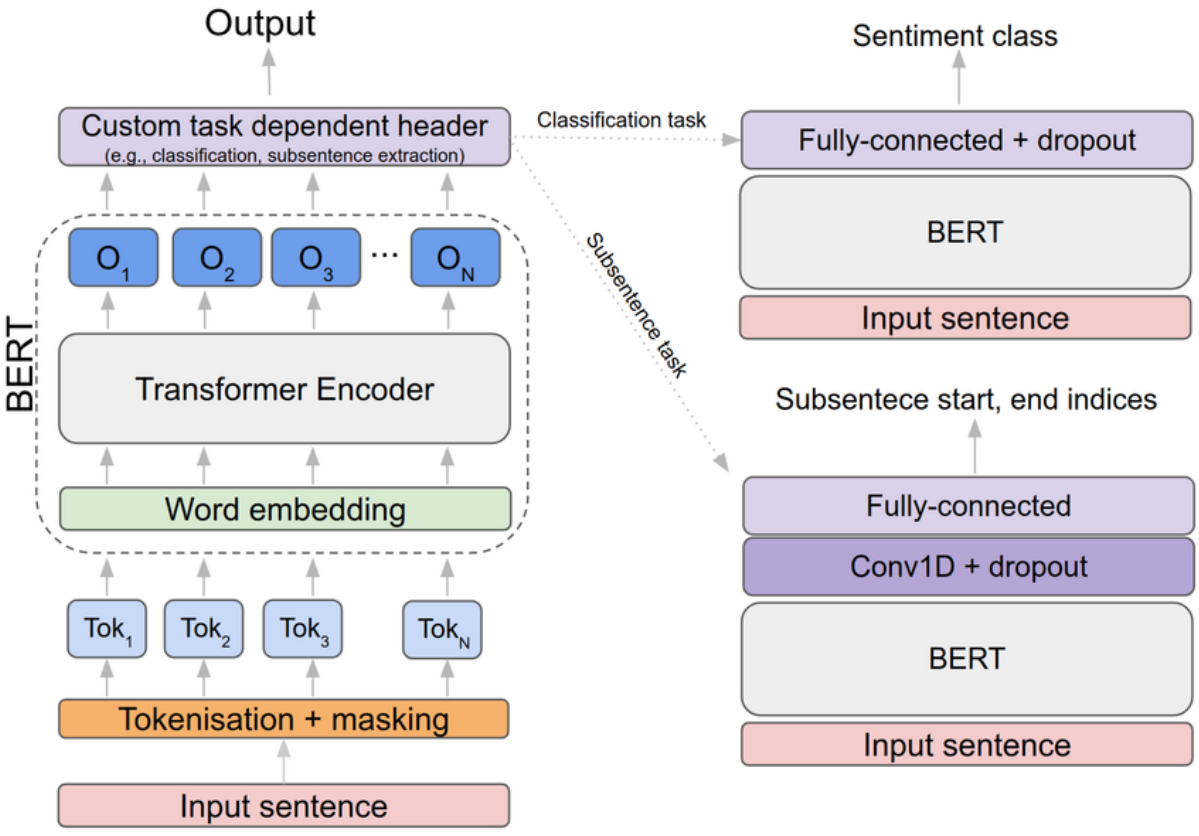

Fig 1: An overview of Bidirectional Encoder Representations from Transformers (BERT) (left) and task-driven fine-tuning models (right). Input sentence is split into multiple tokens (TokN) and fed to a BERT model, which outputs embedded output feature vectors, ON, for each token. By attaching different head layers on top, it transforms BERT into a task-oriented model.

2. Language Models for Answer Generation

Powered by Ollama – a local LLM runner.

Supported Models:

Model	Parameters	Source	Key Traits
`gemma:2b`	~2B	Google DeepMind	Fast, efficient, ideal for lightweight inference
`mistral`	7B	Mistral AI	High performance in reasoning and comprehension
`llama2`	7B	Meta AI	General-purpose, strong language understanding

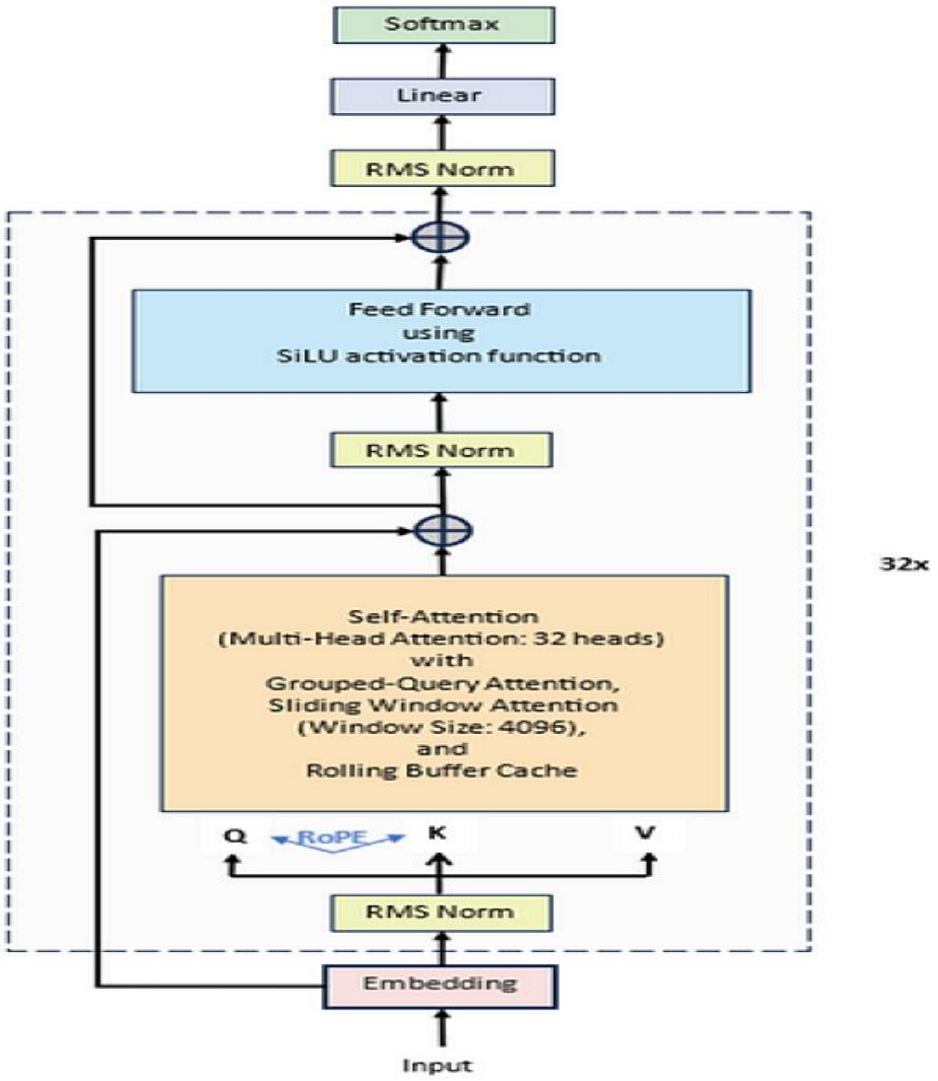

Fig 2: This is the architecture of Mistral 7B

SAMPLE CODE:

```
def generate_answer(query: str, context: List[str], model_name: str = 'gemma:2b') -> str:
    """Generate an answer using the provided context. Returns either:
    - LLM-generated answer
    - Summary of relevant passages if generation fails"""
    if not context:
        return "No relevant context found to answer the question."

    joined_context = "\n\n".join([f"Source {i+1}:\n{ctx}" for i, ctx in enumerate(context)])

    prompt = f"""You are a helpful AI assistant. Answer the question based ONLY on the following context.
If the answer isn't in the context, say you don't know. Be concise and accurate.
Context:
{joined_context}
Question: {query}
Answer:"""

    try:
        response = ollama.generate(
            model=model_name,
            prompt=prompt,
            options={'temperature': 0.1}
        )
        return response['response'].strip()
    except Exception as e:
        # Create summary from references when generation fails
        summary = "\n".join([f"• {ctx[:300]}..." for ctx in context])
        return summary
```

```
def semantic_search(query: str, index: faiss.Index, metadata: List[Dict], k: int = 5) ->
List[Dict]:
    query_embedding = model.encode([query])[0].astype('float32')
    distances, indices = index.search(np.array([query_embedding]), k)
    results = []
    for score, idx in zip(distances[0], indices[0]):
        if idx < len(metadata):
            results.append({
                'score': float(score),
                'file': metadata[idx]['file'],
                'chunk': metadata[idx]['chunk'],
                'content': metadata[idx]['content']
            })
    return results


def index_documents(files: List) -> Tuple[Optional[faiss.Index], Optional[List[Dict]]]:
    index = faiss.IndexFlatIP(dimension)
    metadata = []
    documents = []
    reader = DocumentReader()
    for file in files:
        name = file.name
        content = ''
        if name.lower().endswith('.pdf'):
            content = reader.read_pdf(file)
        elif name.lower().endswith('.docx'):
            content = reader.read_docx(file)
        elif name.lower().endswith('.pptx'):
            content = reader.read_pptx(file)
        elif name.lower().endswith('.txt'):
            content = reader.read_txt(file)
```

OUTPUT:

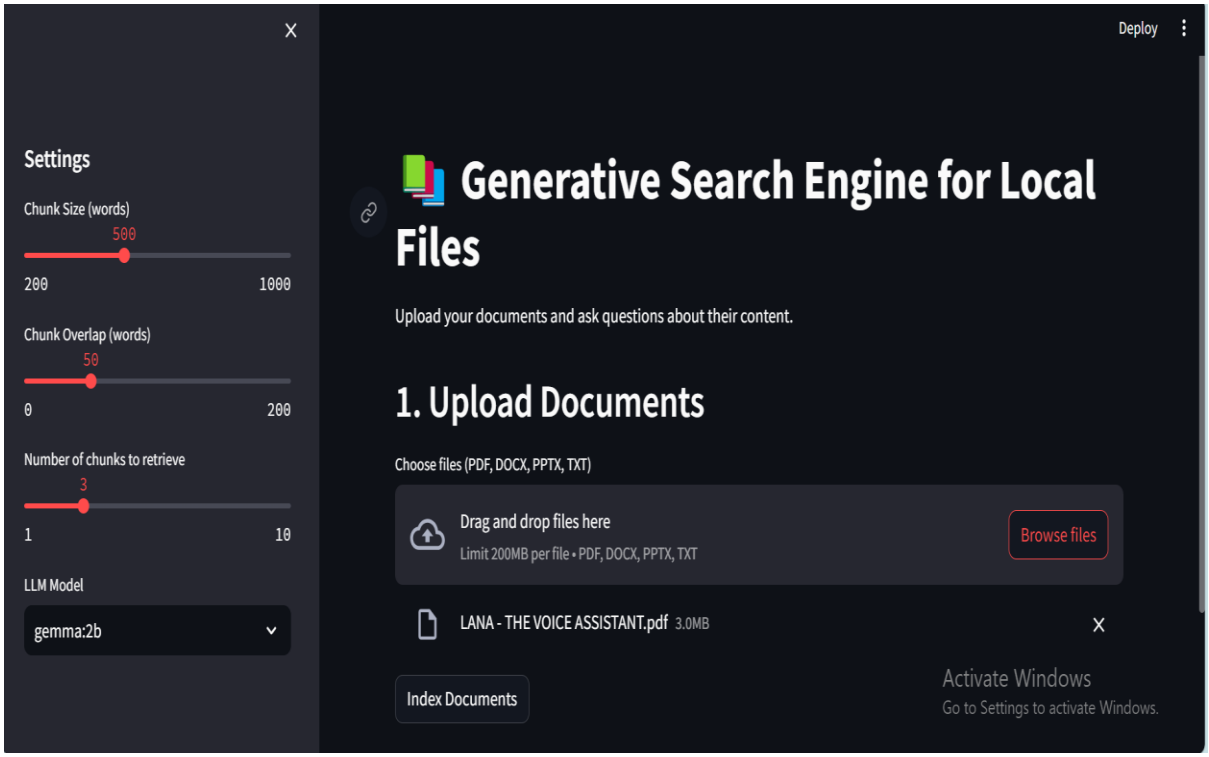

Fig 3: Settings Panel: Interface showing document processing parameters (chunk size, overlap, retrieval count) and LLM model selection.

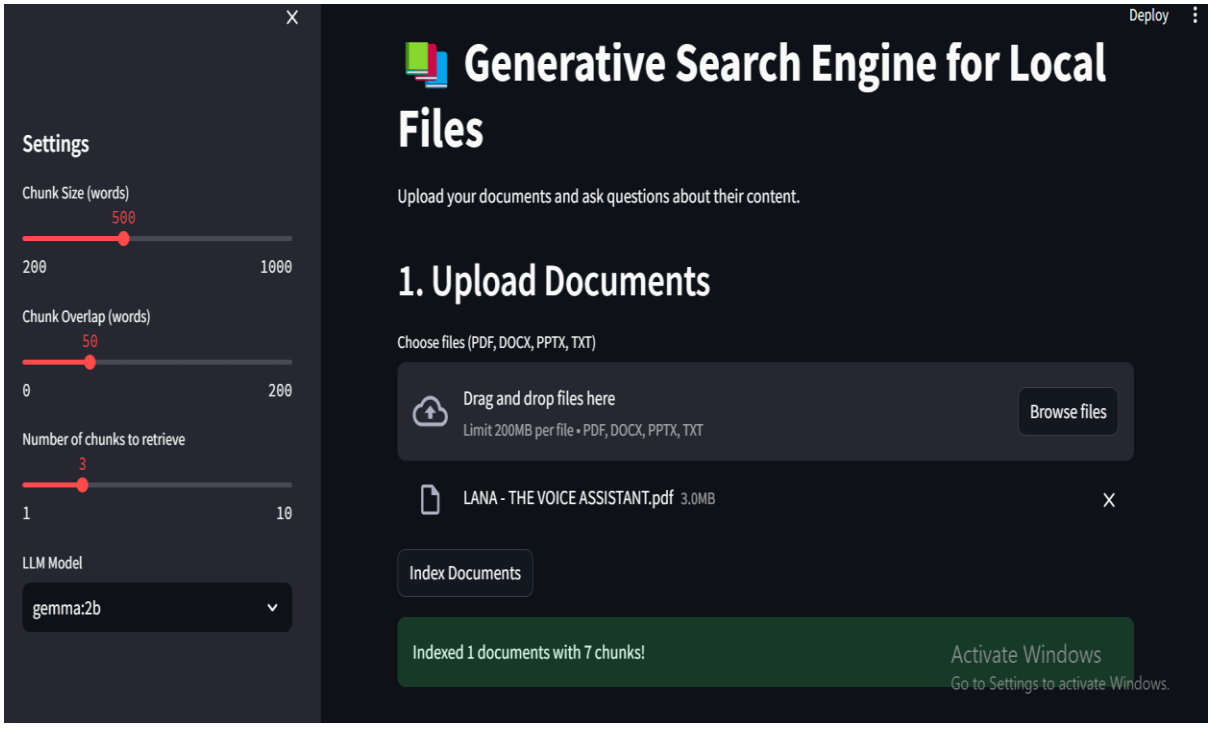

Fig 4: Document Upload Screen: File upload interface with drag-and-drop functionality and indexed document confirmation.

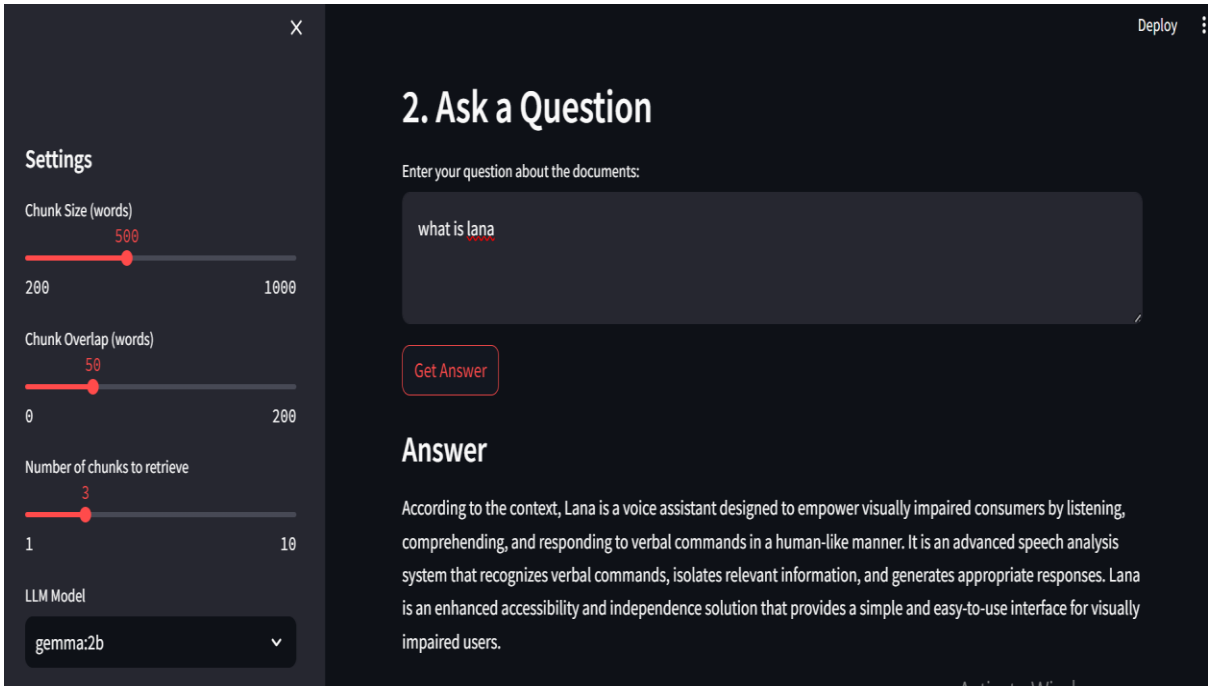

Fig 5: Question Input & Answer: Query box with generated response about "Lana" voice assistant from uploaded PDF.

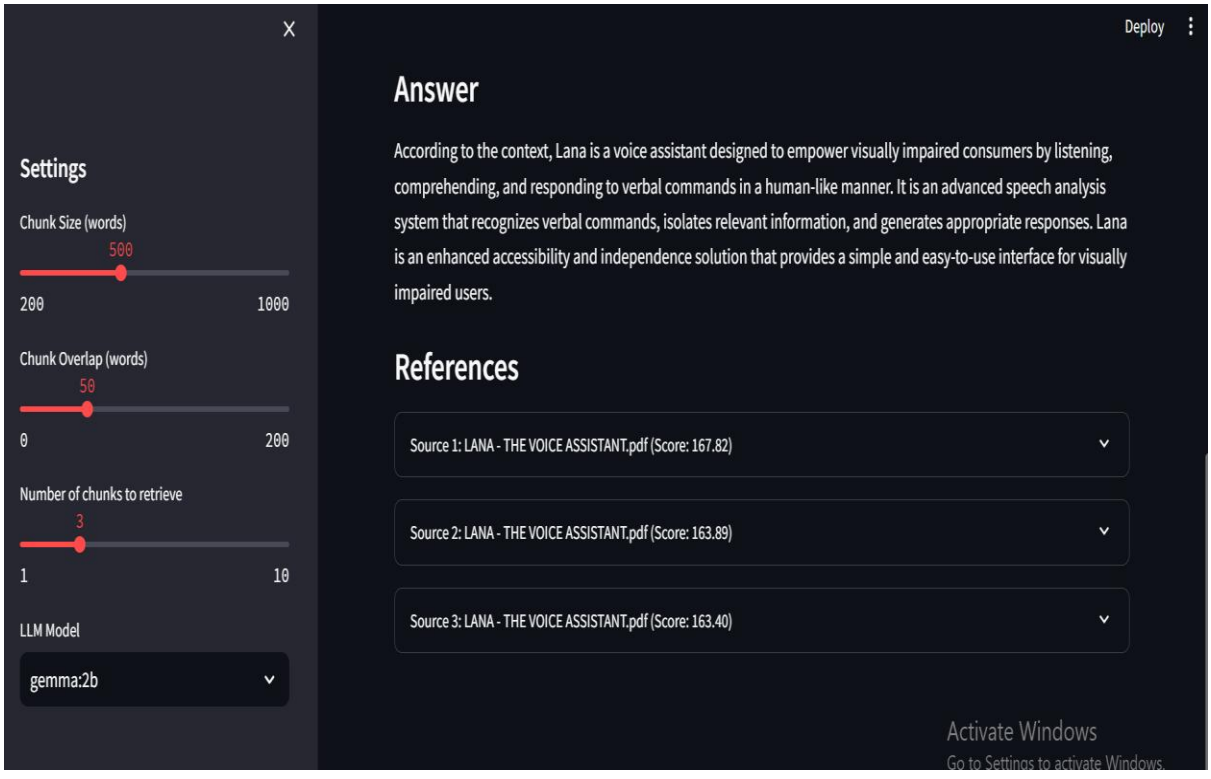

Fig 6: Answer with References: Display of LLM-generated answer and top-matched document chunks with similarity scores.