**DocuMind – A Beginner’s Guide to Chatbots**

Table of Contents

[2 What is this project about? 4](#_Toc197242091)

[2.1 Why are chatbots important / cool? 4](#_Toc197242092)

[2.2 Who is this guide for? 4](#_Toc197242093)

[2.3 What you’ll learn by the end? 4](#_Toc197242094)

[3 Project Workflow (High-Level Overview) 4](#_Toc197242095)

[3.1.1 PDF Input 4](#_Toc197242096)

[3.1.2 Text Extraction and Chunking 4](#_Toc197242097)

[3.1.3 Embedding Generation 4](#_Toc197242098)

[3.1.4 Storage and Caching 5](#_Toc197242099)

[3.1.5 Question Answering Setup: 5](#_Toc197242100)

[3.1.6 User Interaction: 5](#_Toc197242101)

[3.1.7 Feedback Collection: 5](#_Toc197242102)

[4 Technologies Used (Summary) 5](#_Toc197242103)

[4.1.1 PyMuPDF 5](#_Toc197242104)

[4.1.2 NLTK (Natural Language Toolkit) 5](#_Toc197242105)

[4.1.3 Sentence Transformers 5](#_Toc197242106)

[4.1.4 FAISS (Facebook AI Similarity Search) 5](#_Toc197242107)

[4.1.5 SQLite 5](#_Toc197242108)

[4.1.6 Transformers (Hugging Face) 5](#_Toc197242109)

[4.1.7 Torch (PyTorch) 5](#_Toc197242110)

[4.1.8 Regex (re module) 5](#_Toc197242111)

[4.1.9 Logging 6](#_Toc197242112)

[4.1.10 NumPy: 6](#_Toc197242113)

[4.1.11 Datetime: 6](#_Toc197242114)

[5 Deep Dive into Each Technology 6](#_Toc197242115)

[5.1 Tkinter 6](#_Toc197242116)

[5.1.1 What is Tkinter? 6](#_Toc197242117)

[5.1.2 How Tkinter works behind the scenes? 6](#_Toc197242118)

[5.1.3 How is Tkinter used in the project? 6](#_Toc197242119)

[5.2 PyMuPDF 6](#_Toc197242120)

[5.2.1 What is PyMuPDF? 6](#_Toc197242121)

[5.2.2 How does PyMuPDF work behind the scenes? 6](#_Toc197242122)

[5.2.3 How is PyMuPDF used in the project? 7](#_Toc197242123)

[5.3 NLTK 7](#_Toc197242124)

[5.3.1 What is NLTK? 7](#_Toc197242125)

[5.3.2 How does NLTK work behind the scenes? 7](#_Toc197242126)

[5.3.3 How is NLTK used in the project? 7](#_Toc197242127)

[5.4 Sentence Transformers 7](#_Toc197242128)

[What are Sentence Transformers? 7](#_Toc197242129)

[5.4.1 How do Sentence Transformers work behind the scenes? 7](#_Toc197242130)

[How are Sentence Transformers used in the project? 8](#_Toc197242131)

[5.5 FAISS 8](#_Toc197242132)

[5.5.1 What is FAISS? 8](#_Toc197242133)

[5.5.2 How does FAISS work behind the scenes? 8](#_Toc197242134)

[5.5.3 How is FAISS used in the project? 8](#_Toc197242135)

[5.6 SQLite 8](#_Toc197242136)

[5.6.1 What is SQLite? 8](#_Toc197242137)

[5.6.2 How does SQLite work behind the scenes? 9](#_Toc197242138)

[5.6.3 How is SQLite used in the project? 9](#_Toc197242139)

[5.7 Transformers (Hugging Face): 9](#_Toc197242140)

[5.7.1 What are Transformers (Hugging Face)? 9](#_Toc197242141)

[5.7.2 How do Transformers work behind the scenes? 9](#_Toc197242142)

[5.7.3 How are Transformers used in the project? 9](#_Toc197242143)

[5.8 Torch (Pytorch) 10](#_Toc197242144)

[5.8.1 What is Torch (PyTorch)? 10](#_Toc197242145)

[5.8.2 How does Torch (PyTorch) work behind the scenes? 10](#_Toc197242146)

[5.8.3 How is Torch (PyTorch) used in the project? 10](#_Toc197242147)

[5.9 Regex (re module) 10](#_Toc197242148)

[5.9.1 What is Regex (re module)? 10](#_Toc197242149)

[5.9.2 How does Regex work behind the scenes? 10](#_Toc197242150)

[5.9.3 How is Regex (re module) used in the project? 10](#_Toc197242151)

[5.10 Logging 11](#_Toc197242152)

[5.10.1 What is Logging? 11](#_Toc197242153)

[5.10.2 How does Logging work behind the scenes? 11](#_Toc197242154)

[5.10.3 How is Logging used in the project? 11](#_Toc197242155)

[5.11 NumPy 11](#_Toc197242156)

[5.11.1 What is NumPy? 11](#_Toc197242157)

[5.11.2 How NumPy works behind the scenes? 11](#_Toc197242158)

[5.11.3 How is NumPy used in the project? 11](#_Toc197242159)

[5.12 Datetime 12](#_Toc197242160)

[5.12.1 What is Datetime? 12](#_Toc197242161)

[5.12.2 How Datetime works behind the scenes? 12](#_Toc197242162)

[5.12.3 How is Datetime used in the project? 12](#_Toc197242163)

[6 Model Selection and Usage 12](#_Toc197242164)

[6.1 Sentence Transformer: all-MiniLM-L6-v2 12](#_Toc197242165)

[6.1.1 What is all-MiniLM-L6-v2? 12](#_Toc197242166)

[6.1.2 Architecture and Explanation 12](#_Toc197242167)

[6.1.3 How does all-MiniLM-L6-v2 work? 13](#_Toc197242168)

[6.1.4 Why is it used in this project? 13](#_Toc197242169)

[6.2 Text to Text Model: Google Flan T5 13](#_Toc197242170)

[6.2.1 What is FLAN-T5? 13](#_Toc197242171)

[6.2.2 Architecture and Explanation 13](#_Toc197242172)

[6.2.3 How does FLAN-T5 work? 14](#_Toc197242173)

[6.2.4 Why is FLAN-T5 used in this project? 14](#_Toc197242174)

[7 Project Workflow (Detailed) 15](#_Toc197242175)

[7.1 Stage 1: Reading the PDF 15](#_Toc197242176)

[7.2 Chunking and Storing 15](#_Toc197242177)

[7.3 Creating Embeddings 16](#_Toc197242178)

[7.4 Searching with FAISS 16](#_Toc197242179)

[7.5 Answering Questions 16](#_Toc197242180)

[7.6 Feedback Saving (Optional) 17](#_Toc197242181)

[8 Project structure and Code documentation 17](#_Toc197242182)

[8.1 Project Structure 17](#_Toc197242183)

[8.2 Code Documentation 18](#_Toc197242184)

[Code Documentation is done in the form of docstrings for modules as well as functions. Refer them for better understanding of code. 18](#_Toc197242185)

[9 How to Run the Project 18](#_Toc197242186)

[9.1 Clone the Repository 18](#_Toc197242187)

[9.2 Install Dependencies 18](#_Toc197242188)

[9.3 Run the Script 19](#_Toc197242189)

[10 Learning Resources (Extra Links) 19](#_Toc197242190)

[10.1 Learning Resources: Transformers 19](#_Toc197242191)

[10.2 Learning Resources: Sentence Transformers 19](#_Toc197242192)

[10.3 Learning Resources: FAISS 20](#_Toc197242193)

[10.4 Learning Resources: NumPy 21](#_Toc197242194)

[10.5 Learning Resources: Pytorch 22](#_Toc197242195)

[10.6 Some more Links 23](#_Toc197242196)

[10.6.1 Regular expressions 23](#_Toc197242197)

[10.6.2 Logging 23](#_Toc197242198)

[10.6.3 Datetime 23](#_Toc197242199)

[10.6.4 PyMuPDF 23](#_Toc197242200)

[10.6.5 SQLite 23](#_Toc197242201)

[11 Future Improvements / Fun Ideas 23](#_Toc197242202)

# What is this project about?

DocuMind🤖💬 is an open-source chatbot that can read, understand, and answer questions about your documents! 📄📚 It’s designed to help teams get instant support ⏱️ and clarify doubts from their existing knowledge base, saving time and boosting employee efficiency 🚀 — all without worrying about costly API bills 💸 or subscriptions 🔒.

## Why are chatbots important / cool?

Chatbots are important because they provide 24/7 support 🕒 for teams without the need for manual assistance. This enables users to get instant answers 💡 and understand complex information with ease 📘➡️✨.

They help break down difficult documents 🧠📑, speed up productivity ⚡, and offer personalized support 🎯, making the user experience much more enjoyable and efficient. Plus, they’re always improving! 🔄📈

## Who is this guide for?

This guide is for anyone new to Artificial Intelligence 🤔🧠 but who has a basic understanding of computer science 💻. You don’t need to be familiar with all the technologies used — just a willingness to learn and explore! 🌱

## What you’ll learn by the end?

By the end of this guide, you’ll gain a clear understanding 🕵️‍♀️ of how the Chatty Chat Bot works under the hood 🔧🤖.

You’ll explore not just AI, but also the fundamental theories 📘 and practical code 💡👨‍💻 that power the whole project. It’s your first step into the world of chatbots 🚪➡️🤖, and a solid foundation for building your own! 🏗️✨

# Project Workflow (High-Level Overview)

### PDF Input

The user uploads a PDF file with a file uploader GUI.

### Text Extraction and Chunking

PDF content is extracted and split into smaller pieces for easier processing

### Embedding Generation

Each piece of content is converted into a numeric representation called an embedding. This is done using a pretrained Sentence Transformer model and is used for similarity search.

### Storage and Caching

The embeddings and text chunks are saved for fast retrieval without reprocessing the same PDF again.

### Question Answering Setup:

A language model is loaded to generate answers based on a given context.

### User Interaction:

User input is converted into embedding and the chatbot finds the most relevant chunk embedding from the PDF and generates an answer using relevant chunks.

### Feedback Collection:

Users can optionally provide feedback on the answers, which is stored in databases and log files for review.

# Technologies Used (Summary)

### PyMuPDF

Library for data extraction, analysis, and manipulation of PDF documents.PDF given by user is processed using PyMuPDF.

### NLTK (Natural Language Toolkit)

Natural Language Library. NLTK tokenizes(splits) sentences into smaller chunks.

### Sentence Transformers

Sentence Transformers is a library used for creating embeddings (Numerical representation) of text chunks and queries.

### FAISS (Facebook AI Similarity Search)

Library created by meta for clustering (grouping of similar chunks) and searching (searching most relevant chunk based on user query) embeddings.

### SQLite

SQLite is a free and open-source relational database. Here SQLite stores PDF embeddings and chunks along with user feedback.

### Transformers (Hugging Face)

Library which contains various Open-source AI models. AI model used in these projects are imported from hugging face.

### Torch (PyTorch)

PyTorch is an open-source deep learning framework developed by Meta. In this project Hugging Face models are implemented using Pytorch.

### Regex (re module)

Library for regular expression matching operations. Regex is used for finding certain patterns which needs to be removed from input text before it is processed and tokenized.

### Logging

Library for event logging. Logging helps in recording errors and feedback in log files for monitoring and debugging purposes.

### NumPy:

Library for efficient computation on large arrays and matrices. Here it is used for operations related to storage and retrieval of embedding vectors.

### Datetime:

The datetime library supports manipulating dates and times. It is used here for timestamping data such as logs.

# Deep Dive into Each Technology

## Tkinter

### What is Tkinter?

Tkinter is the standard GUI (Graphical User Interface) library of Python. It gives a simple way to create windows, dialogs and other common GUI elements. Tkinter acts as a python wrapper around the Tcl(Tool Command Language)/Tk GUI toolkit(A high level language made for building GUI applications) making it easier for Python developers to build GUI elements and applications without external dependencies.

### How Tkinter works behind the scenes?

Tkinter internally bridges Python and the Tcl/Tk interpreter. When a Python script uses Tkinter to create a window or a widget, Tkinter generates the corresponding Tcl commands under the hood. These commands are executed by the Tcl interpreter to render the GUI elements on the screen. Tkinter maintains its own event loop (called mainloop), listening for user actions like clicks or keystrokes and dispatching the appropriate callback functions when events occur. This event-driven model allows dynamic interaction between the user and the application.

### How is Tkinter used in the project?

In the project, Tkinter is used for:

* Creating GUI for user to upload pdf file
* Restricting user to upload only pdf files to avoid errors in further processing

## PyMuPDF

### What is PyMuPDF?

PyMuPDF is a lightweight, high-performance Python library for working with PDF documents and even other file formats. It provides simple functions for extracting text, images, and metadata as well as for modifying documents.

### How does PyMuPDF work behind the scenes?

PyMuPDF is a python binding for MuPDF C library, which is designed for fast and memory-efficient PDF rendering. A python binding are functions that allows calling of C/C++ functions with python code. When a PDF is loaded, PyMuPDF parses document structure and identifies pages, text blocks and other data. For text extraction, it reads the internal structure of each page (page dictionary) and reconstructs the visible text by analysing layout, fonts, positions, and characters without relying on OCR (Optical Character Recognition). This makes extraction very fast and accurate for digital PDFs.

### How is PyMuPDF used in the project?

In this project, PyMuPDF is used to:

* Open the PDF uploaded by the user.
* Extract raw text from each page of the PDF.
* Provides extracted text for further processing like cleaning, tokenization, and embedding generation.

## NLTK

### What is NLTK?

NLTK (Natural Language Toolkit) is a Python library for working with human language data (Natural Language). It provides easy-to-use interfaces for Natural Language Processing(NLP) tasks like tokenization (splitting text into smaller words or sentences). NLTK is commonly used in research and education due to its ability to make text processing easier.

### How does NLTK work behind the scenes?

NLTK provides pre-built datasets, models, and algorithms that can work with textual data. For tokenization, NLTK uses a mix of rule-based methods and trained models to split text. Some rules like punctuation, spaces, and language rules are used to recognize where sentences end or words break. NLTK stores data like stop words, corpora (collections of text), and grammar structures that can be used directly without training models from scratch.

### How is NLTK used in the project?

In this project, NLTK is used to:

* Break (tokenize) the extracted text from the PDF into smaller chunks.
* Tokenized chunks are then used to generate numerical representations(embeddings).
* Processes text into manageable segments for better analysis and performance.

## Sentence Transformers

(For more explanation on transformers refer 4.6 Transformers (Hugging Face))

What are Sentence Transformers?Sentence Transformers is a Python library that creates vector representations (embeddings) of sentences or text chunks. Embeddings are dense numeric arrays that represent and capture the semantic meaning of input text. Sentence Transformers are built on top of pre-trained models like BERT for tasks like sentence similarity, clustering, and paraphrasing mining.

### How do Sentence Transformers work behind the scenes?

Sentence Transformers use attention-based transformer models like BERT. Normally BERT produces token-level (word or letter level) and not sentence level embeddings. Sentence transformers modify architecture to help create a single embedding for a full sentence. The numeric representation of sentence (sentence vector) is done by pooling strategies like mean pooling (average of all token embeddings).

How are Sentence Transformers used in the project?In this project, Sentence Transformers are used to:

* Conversion of text chunks into dense vectors(embeddings).
* Embeddings capture the true semantic meaning of text for searching.
* These embeddings are later stored in FAISS for fast similarity search.

## FAISS

### What is FAISS?

FAISS (Facebook AI Similarity Search) is an open-source library developed by Meta for efficient similarity search and clustering of dense vectors. A dense vector is an array with mostly non zero real numbers. It is mainly used for finding nearest neighbours or points that are most similar to a given query point in large datasets.

### How does FAISS work behind the scenes?

FAISS works by representing data points as numerical vectors called embeddings. To find similar vectors, FAISS compares the distance between vectors using metrics like Euclidean distance or cosine similarity. Cosine similarity is a measure of similarity between two vectors that determines the cosine of the angle between them. It quantifies how similar two vectors are in terms of their direction, regardless of their magnitude.For large datasets, FAISS does Indexing by createing special data structures (indexes) like flat indexes, inverted files, or graph-based structures to organize vectors and speed up the similarity search process. Using indexes allows for more efficient storage and retrieval of text data compared to storing the raw text strings. FAISS uses Quantization, a technique to reduce the computational and memory costs by representing vectors with low-precision data types like 8-bit integer instead of the usual 32-bit float. FAISS does searching using methods like Approximate Nearest Neighbour (ANN). Instead of scanning every vector one by one, FAISS searches only a small portion of the dataset which provides a balance between speed and accuracy. This makes FAISS fast for real-time search tasks even with millions of vectors.

### How is FAISS used in the project?

In this project, FAISS is used to:

* Create an index from the embeddings generated from PDF text chunks.
* Search the most relevant chunk quickly when user enters query.
* Optimize the chunk matching process for large pdfs.

## SQLite

### What is SQLite?

SQLite (Structed query Language Lite) is a lightweight and serverless relational database engine. Unlike traditional databases that require a server process (like MySQL or PostgreSQL), SQLite stores the database as a single file on disk. This allows for fast and simple set up making it perfect for local storage and prototyping.

### How does SQLite work behind the scenes?

When a program interacts with SQLite, it reads from and writes directly to the database file using optimized file I/O operations. SQLite directly links into the application and does not require a separate server process. It is a relational database where data is stored in tables as rows and columns. Relationships can be established between different tables and data operations are done by queries. Internally, SQLite uses a B-tree (optimized data structure for data access) to organize tables and indexes for fast lookup. All operations like inserts and updates are wrapped inside transactions to ensure atomicity (operations fully complete or fully fail).

### How is SQLite used in the project?

In this project, SQLite is used to:

* Store the extracted chunks of PDF text and their corresponding embeddings.
* Save user feedback for further analysis and fine tuning.

To Allow immediate retrieval of data when needed without requiring an external database server

## Transformers (Hugging Face):

### What are Transformers (Hugging Face)?

Transformers are a type of deep learning model architecture that are effective for tasks with sequential data like text and audio. Entire sequences are processed at once compared to older models like RNNs using a mechanism called attention. Attention allows capturing long-range dependencies better.

Hugging Face provides an open-source platform and library which contains pre-trained models that can be easily fine-tuned or used directly for many tasks like text classification, translation, summarization, and more.

### How do Transformers work behind the scenes?

Transformers mainly rely on the *self-attention* mechanism. Self-attention allows transformers to look at all parts of the input sequence simultaneously and decide which parts are important for making predictions. Instead of processing inputs one-by-one (like RNNs), transformers process the full input in parallel, using layers made up of attention blocks and feed-forward neural networks (Neural networks in which information flows in one direction without loops or feedback). The attention mechanism assigns attention scores that determine the importance given different parts of the input sequence. Models like BERT, RoBERTa, and GPT are built based on transformer architecture. These models trained on huge amounts of text data (corpora) so that the models can understand grammar, context, meaning, and even relationships between words.

### How are Transformers used in the project?

In this project, transformers module of HuggingFace is used for:

Loading corresponding tokenizer for pretrained model using Auto

Loading pretrained model using AutoModelForSeq2SeqLM

Creating a text-to-text generation pipeline (input of the model is query and output is answer generated based on relevant text chunk) with the model, tokenizer and device (CPU/GPU)

## Torch (Pytorch)

### What is Torch (PyTorch)?

Torch (PyTorch) is an open-source deep learning framework developed by Facebook’s AI Research lab. It is used for building and training neural networks. PyTorch gives easy and flexible methods to define, compute, and optimize operations on tensors (algebraic objects) which are the fundamental blocks of deep learning models.

### How does Torch (PyTorch) work behind the scenes?

PyTorch works by creating a dynamic computation graph (also called define-by-run). Computation graphs are used to represent mathematical expressions. It provides functional description of the required computation. Instead of predefining the entire computation graph before execution, PyTorch builds the graph dynamically along with operations performed. This makes it easier to debug and modify models on the fly. PyTorch uses Tensors (multi-dimensional arrays similar to NumPy arrays but with GPU acceleration support). When tensor operations are performed, PyTorch records these operations to later compute gradients (which measure how much the model's predictions should change to reduce errors) efficiently.

### How is Torch (PyTorch) used in the project?

In this project, Torch (PyTorch) is used for:

* Computing and handling tensor operations in transformer models.
* Provides a backend for computations in embedding generation and text-to-text prediction in models imported from HuggingFace library.

## Regex (re module)

### What is Regex (re module)?

Regex (Regular Expression) is a sequence of characters that specifies a match pattern in text. Python’s re module gives methods to search, match and manipulate strings based on these patterns. Regular expressions can be used for extracting text that follows a specific format(email), validation (checking input format) and splitting strings.

### How does Regex work behind the scenes?

Regex works by parsing a regular expression written in a syntax where certain characters have specific meanings. For example, “\d” matches any digit, “\w” matches any alphanumeric character, and “.” matches any character. Under the hood when the pattern is compiled with re.compile(), the regex engine converts the pattern into a state machine (a computation model) that checks the input text character-by-character according to the pattern rules. Optimized internal algorithms allow faster execution of complex matches

### How is Regex (re module) used in the project?

In this project, the re module is used for:

* Sanitize text given by user in the form of questions and feedback by removing unprintable characters.
* The sanitized text can be further processed for creating embeddings of user question or storing user feedback in SQLite.

## Logging

### What is Logging?

Logging is the process of recording events, messages and data generated by an application at execution. It helps developers track flow of application, monitoring and debugging, In python, the built-in logging module provides methods to create and store logs of different types (DEBUG, INFO, WARNING, ERROR, CRITICAL).

### How does Logging work behind the scenes?

When a log message is generated using Python’s logging module, it is passed through a Logger object, which decides how to handle the message based on its severity level. Logger object routes log messages to different log handlers like console output or log files. Behind the scenes, the module uses an internal tree-like hierarchy of loggers, filters, formatters, and handlers to control what is logged, how it is formatted, and where it is stored. This ensures that messages are recorded consistently across the application without handling file writes or console prints.

### How is Logging used in the project?

In this project, logging is used for:

* Recording errors and exceptions during different stages like model loading, text generation, and database operations.
* Logging user feedback data for later analysis and model fine-tuning.

## NumPy

### What is NumPy?

NumPy (Numerical Python) is a Python library used for numerical and scientific computing. It provides tools for working with large multi-dimensional arrays and matrices, and also allows a wide range of optimized mathematical functions to operate on these arrays efficiently. It is used in data science, machine learning, and scientific research.

### How NumPy works behind the scenes?

NumPy arrays are stored in contiguous blocks of memory unlike python lists, which are collections of pointers to list object. List objects are slow since list elements do not have a fixed data type and need to be allocated dynamically. Since Numpy arrays contain the same type of elements they can be stored in a contiguous manner in memory. This type of memory storage can be used for hardware level optimization in Numpy operations (using C and Fortran internally). This results in much faster computations compared to standard Python operations. NumPy also uses a technique called broadcasting which allows operations on arrays of different shapes without writing for loops.

### How is NumPy used in the project?

In the project, NumPy is used for:

* Internally, it is used for handling large sets of numerical data efficiently. It helps in performing mathematical operations like summations, averages, and matrix manipulations easily.
* Explicitly, the Numpy module is used in the project to Load and store embedding vectors effectively.

## Datetime

### What is Datetime?

Datetime is a Python built-in module that provides classes for manipulating dates (year, month, day) and times (hour, minute, second, microsecond). It supports date arithmetic operations, comparison of dates, supports different date formats and parse dates from strings. It is used when dealing with timestamps, logging events and working with time-based data.

### How Datetime works behind the scenes?

In the datetime module, dates and times are internally represented as numbers (with methods like number of days since a reference date, or the number of microseconds). This numerical representation makes it easier to optimize operations on dates. The module combines date, time, datetime, and timedelta classes to give a full toolkit for working with time-related data. It also handles calculations like leap years behind the scenes to abstract complexity from the user.

### How is Datetime used in the project?

In the project, datetime is used for:

* Generating timestamps for error logs and feedback logs so that user can observe when the error or feedback submission occurred
* Formatting timestamps into human-readable formats when storing feedback data in SQLite database.
* Keeping track of feedback timestamps to enforce a minimum gap between submissions.

# Model Selection and Usage

## Sentence Transformer: all-MiniLM-L6-v2

### What is all-MiniLM-L6-v2?

all-MiniLM-L6-v2 is a pre-trained Sentence Transformer model designed to convert sentences or short paragraphs into 384-dimensional dense vector embeddings. These embeddings capture the semantic meaning of the text, enabling tasks like semantic search, clustering, and similarity comparison.​

### Architecture and Explanation

The model is based on MiniLM, a distilled version of BERT (Bidirectional Encoder Representations from Transformers) developed by Google. MiniLM uses only 6 Transformer layers (L6) and fewer attention heads, making it significantly more lightweight. It is created through knowledge distillation, a process where a smaller student model learns to replicate the behaviour of a larger teacher model by mimicking its predictions and internal representations. Despite its reduced size, a distilled model like MiniLM often maintains comparable performance while requiring far fewer computational resources, making it ideal for deployment on low-power devices and scalable applications. MiniLM doesn’t just copy the output predictions of the teacher—it also learns to replicate the self-attention distributions, including attention maps (which show how tokens relate to each other) and value-layer outputs from the teacher’s attention blocks. This allows the student model to internalize the teacher’s reasoning and structural understanding of language, resulting in strong generalization even with limited capacity. Additionally, Sentence Transformers adapt the architecture for sentence-level embeddings by introducing pooling strategies. In the case of all-MiniLM-L6-v2, mean pooling is used, where token embeddings are averaged to produce a fixed-size vector representing the entire sentence. This enables the model to efficiently handle tasks like semantic similarity, clustering, and search.

### How does all-MiniLM-L6-v2 work?

The input sentence is tokenized into smaller units called tokens.​These tokens pass through 6 transformer layers, each performing self-attention (considers the entire input sequence to determine the importance of each word in relation to others) and feed-forward (Neural networks with no feedback or loops) operations to build contextual embeddings for each token.​After the transformer layers, a pooling layer (mean pooling) aggregates token embeddings into a single vector that represents the entire sentence.​The final result is a 384-dimensional dense vector that captures the semantic meaning of the sentence.​. ​

### Why is it used in this project?

In the project, all-MiniLM-L6-v2 is used because:

* **Semantic Understanding**: It captures the real meaning behind user queries, not just the words used.​
* **Speed & Efficiency**: It runs fast, even on computers with limited processing power.​
* **High-quality Sentence Embeddings**: The 384-dimensional vectors it produces allow for accurate comparison between queries and document chunks.​
* **Smooth Integration with FAISS**: These embeddings is stored in a FAISS index, which instantly retrieves the most relevant text chunks based on user questions.​

This makes all-MiniLM-L6-v2 ideal for tasks like semantic search and document Q&A in this application.​

## Text to Text Model: Google Flan T5

### What is FLAN-T5?

**FLAN-T5** (Fine-tuned LAnguage Net T5) is an advanced version of Google's T5 (Text-to-Text Transfer Transformer) model. The original T5 model treats every NLP task as a text-to-text problem, FLAN-T5 enhances this approach by incorporating instruction fine-tuning. This means the model is trained not just on tasks but also on understanding and following specific instructions, making it more adept at handling a wide range of tasks with better generalization.

### Architecture and Explanation

FLAN-T5 (Fine-tuned Language Net based on T5) builds upon the architecture of the original T5 (Text-to-Text Transfer Transformer) model, which uses a sequence-to-sequence (encoder-decoder) structure. In this architecture, the encoder is responsible for reading and understanding the input text, transforming it into a contextualized internal representation. This representation is then passed to the decoder, which generates the appropriate output text like translation, summary, answer to a question, or any other form of natural language output. This structure allows T5 and its derivatives to treat all NLP tasks uniformly as a text-to-text problem.

What sets FLAN-T5 apart is its instruction fine-tuning process. Unlike traditional fine-tuning, which typically adapts a model to perform well on a narrow set of tasks or datasets, instruction fine-tuning exposes the model to a diverse collection of NLP tasks, each presented in the form of explicit natural language instructions (e.g., "Translate this sentence into French," "Summarize the following paragraph"). These instructions act as prompts that guide the model in understanding the goal of the task, improving its ability to generalize to new or unseen tasks.

To implement this, existing datasets are reformatted using prompting templates that convert tasks into instructional examples. For instance, instead of training the model on raw data for sentiment classification, the data might be framed as: "Is the following review positive or negative? [review text]." By training on thousands of such examples spanning multiple NLP domains—including translation, summarization, question answering, and more—FLAN-T5 learns to follow instructions and adapt to new tasks without needing task-specific fine-tuning.

This instruction-tuned methodology results in a model that is not just capable of solving tasks it was explicitly trained on, but also performs strongly on zero-shot and few-shot evaluations—where the model is asked to perform a task with little to no prior exposure. As a result, FLAN-T5 demonstrates robust generalization, significantly improving performance across a wide range of benchmarks compared to models that were not trained with instructional data.

The FLAN-T5 family includes models of varying sizes to cater to different computational needs:

* **FLAN-T5 Small**: ~80 million parameters
* **FLAN-T5 Base**: ~250 million parameters
* **FLAN-T5 Large**: ~770 million parameters
* **FLAN-T5 XL**: ~3 billion parameters
* **FLAN-T5 XXL**: ~11 billion parameters

### How does FLAN-T5 work?

The model receives an input in the form of an instruction and associated text, such as "Translate English to French: 'Hello'". The encoder transforms this input into a sequence of hidden representations, capturing the contextual meaning of the instruction and text. The decoder generates the output text based on the encoded representations, producing responses like "Bonjour" for the above example. FLAN-T5 is trained on a mixture of tasks with varied instructions, enabling it to understand and follow a wide array of prompts effectively.

### Why is FLAN-T5 used in this project?

FLAN-T5 is chosen for this project due to its:

* **Versatility**: Its instruction fine-tuning allows it to perform well across diverse NLP tasks without task-specific fine-tuning.
* **Efficiency**: Smaller models like FLAN-T5 Base or Large offer a good balance between performance and computational resource requirements.
* **Accessibility**: Being open-source and available on platforms like Hugging Face, it's easy to integrate and deploy in various applications
* **Performance**: FLAN-T5 models have demonstrated strong performance on benchmarks, often rivalling larger models in effectiveness.

These attributes make FLAN-T5 a suitable choice for building applications that require understanding and generating human-like text based on instructions.

# Project Workflow (Detailed)

## Stage 1: Reading the PDF

**🎯 Goal**

Open a PDF file and read its entire text content.

**🧠 What's happening?**

First, a small window pops up asking the user to choose a .pdf file.

The code then uses a Python library called PyMuPDF to go through every page in that PDF and collect all the visible text.

This full text is joined into a single long string.

We then use a special function (hashlib) to generate a unique fingerprint (called a hash) of the entire text. This is like giving your PDF a digital ID, useful for caching (so the app doesn’t redo the same work next time).

**✅ What we achieved**

We loaded all the text from the PDF into memory and gave it a unique ID for later use.

## Chunking and Storing

**🎯 Goal**

Break that long text into small parts (“chunks”) for better processing.

**🧠 Why this step?**

Large language models (LLMs), like ChatGPT, can't process very large text inputs all at once. They work best with small, meaningful portions of text. So we divide the long text into small overlapping segments.

**💡 How it's done**

The text is split into sentences using the nltk library.

We combine sentences into groups (chunks), with about 500 words per chunk.

To maintain continuity between chunks, we let each one slightly overlap (100 words by default) with the next.

These chunks are stored in a small database (SQLite) on your computer. This makes it easy to look up later when we need relevant pieces.

**✅ What we achieved**

Turned one long PDF into organized, smaller pieces stored locally.

## Creating Embeddings

**🎯 Goal**

Convert each chunk of text into a numerical format that a computer can understand and compare.

**🧠 Why embeddings?**

Computers can't understand raw human language. Instead, we translate each chunk of text into a vector (a list of numbers). These vectors represent the meaning of the text.

**💡 How it's done**

We use a tool called SentenceTransformer (a type of AI model) to turn each chunk into a vector.

Each vector captures the semantic meaning of the text — meaning similar chunks get similar vectors.

All these vectors are saved in a file (.npy) to avoid recalculating them later.

**✅ What we achieved**

We now have a computer-readable map of all PDF content, ready for quick comparisons.

## Searching with FAISS

**🎯 Goal**

Find the most relevant chunks of text based on a user’s question.

**🧠 What is FAISS?**

FAISS is a powerful library developed by Facebook that lets you search through vectors very fast. Think of it like a smart search engine that finds “similar meanings” instead of exact words.

**💡 How it works**

We load all our saved vectors into a FAISS index.

When a user types a question, we also turn it into a vector.

FAISS compares this question vector with all the chunk vectors and gives back the top matching chunks.

We look up these matching chunks in our SQLite database.

**✅ What we achieved**

We can now find relevant parts of the PDF that relate to the user's question.

## Answering Questions

**🎯 Goal**

Generate a natural, human-readable answer using the relevant chunks.

**🧠 How does the model generate an answer?**

The top-matching chunks (from Stage 4) are combined into a single context paragraph.This paragraph is passed along with the user's question to a QA model (a fine-tuned AI model that specializes in answering questions).

A prompt is created that looks like this:

makefile

Copy

Edit

Context: [relevant chunks]

Question: [user question]

**Answer:**

The QA model then generates a sentence or paragraph as the answer.

**✅ What we achieved**

The system uses the original PDF content + AI to give you a smart answer.

## Feedback Saving (Optional)

🎯 **Goal**

Let users’ rate or comment on the answers they get, to improve future performance.

🧠 **Why feedback?**

Collecting user feedback helps developers understand if the answers are useful. It can also be used later to fine-tune the AI model or improve data quality.

**💡 How it works**

After every answer, the user can leave a comment or rate it.

The feedback is saved in two places:

A simple log file (feedback.log) for manual reading.

A database (feedback.db) for structured analysis.

There's also a rate limit (e.g. 30 seconds) to avoid spam feedback on the same question.

✅ **What we achieved**

A feedback loop that lets the system learn and improve over time.

# Project structure and Code documentation

## Project Structure

Chatty/

├── cache/                     # Stores FAISS index, embeddings, and chunk DBs

│   └── <pdf\_hash>/            # Subfolder for each uploaded PDF

│   ├── chunks.db #(created)Database for storing text chunks

│   ├──embeddings.npy #(created)Store embeddings arrays

│ ├──faiss.index #faiss index file for fast retrieval of data

├── constants.py               # Configuration constants

├── chunk\_extraction.py        # Extracts and chunks PDF text

├── create\_db.py               # Creates SQLite DB to store chunks

├── download\_models.py         # (Optional) Script to download models locally

├── feedback\_logger.py         # Logs user feedback to text file and SQLite

├── file\_uploader.py           # Handles PDF file selection using GUI

├── get\_chunk.py               # Retrieves chunks from SQLite DB by ID

├── load\_qa\_model.py           # Loads local QA (T5) model pipeline

├── loading\_and\_caching.py # Caches embeddings, FAISS index, and DB

├── main.py                    # Main CLI app for PDF Q&A

├── sanitizer.py               # Cleans and sanitizes input text

├── local\_models/              # Directory to store downloaded local models

│   ├── sentence\_model/        # Saved sentence transformer model

│   └── qa\_model/              # Saved QA (text2text) model

├── errors.log                 # Error logs for debugging

├── feedback.db                # SQLite DB for feedback entries

├── feedback.log               # Text log for feedback entries

├── requirements.txt           # Requirements needed for running chatbot

├── documentation.docx         # Detailed word document about project

└── README.md                  # Get a quick gist about project.

## Code Documentation

📝 Code documentation is done in the form of docstrings 📄 for modules 📦 as well as functions ⚙️. Refer to them for a better understanding of the code 🧠💡.

# How to Run the Project

Follow these simple steps to get the project up and running on your machine:

## Clone the Repository

**Download the project files to your local machine:**

git clone https://github.com/KDSCRIPT/DocuMind.git

cd DocuMind

## Install Dependencies

**Make sure you’re using Python 3.7+ and install all required libraries:**

pip install -r requirements.txt

💡 *Tip*: Consider using a virtual environment to avoid conflicts:

python -m venv venv

**Linux:** source venv/bin/activate

**Windows:** venv\Scripts\activate

pip install -r requirements.txt

## Run the Script

**Execute the main Python script to test or run the model:**

python script.py

✅ If everything is set up correctly, you should see file upload GUI for PDF and chatbot output will be visible in terminal after uploading pdf.

# Learning Resources (Extra Links)

## Learning Resources: Transformers

**🔰 Easy Start**

* **📺 The Illustrated Transformer by Jay Alammar**A visual and intuitive guide that demystifies the transformer model's inner workings.[**Explore the guide**](http://jalammar.github.io/illustrated-transformer/)

**🛤️ Intermediate**

* **📘 Visual Guide to Transformer Neural Networks (**[**Hedu AI by Batool Haider**](https://www.youtube.com/@HeduAI)**)**

This is a YouTube series explains transformer architecture in a visual way

[**Visual Guide to Transformer Neural Networks**](https://www.youtube.com/watch?v=dichIcUZfOw)

**🚀 Deep Dive**

* **📚 Attention is all you need**This is the first research paper by Google that introduced transformer architecture. [**Access the paper**](https://arxiv.org/abs/2311.17633)
* **💻 Transformers from Scratch - DL**

Kaggle notebook that implements encoder of a transformer from scratch[**tranformers-from-scratch**](https://www.kaggle.com/code/auxeno/transformers-from-scratch-dl?scriptVersionId=137372275)

## Learning Resources: Sentence Transformers

**🔰 Easy Start**

* **📖 GeeksforGeeks article on Sentence Transformers**A post from geeks for geeks to get started with sentence transformers [**GeeksforGeeks- Sentence Transformers**](geeksforgeeks.org/sentence-transformer/)
* **♠️All all-MiniLM-L6-v2 hugging face model card.** Model card in hugging face for all-MiniLM-L6-v2 that gives a gist about the model and how to use it from hugging face model hub.

[**HuggingFace all-MiniLM-L6-v2**](https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2)

**🛤️ Intermediate**

* **📘 Official Sentence-Transformers Documentation**Direct from the source. Includes setup, training, and use cases like semantic search and clustering.[**Sentence Transfomers documentation**](https://www.sbert.net/)
* **🔍 Hugging Face space for semantic search using Sentence Transformers**A Hugging Face Space showing how to use sentence transformers for similarity search.[**Using Sentence Transformers for semantic search**](https://huggingface.co/spaces/sentence-transformers/embeddings-semantic-search)
* **💡 Blog: Using Sentence Transformers for Semantic Search** [**Read Here**](https://sakilansari4.medium.com/unleashing-the-power-of-sentence-transformers-revolutionising-semantic-search-and-sentence-29405c13f2b0)

**🚀 Deep Dive**

* **📄 Sentence-BERT: Making BERT Efficient for Semantic Similarity**The original research paper that introduced Sentence Transformers and their training strategy.[**Read the paper**](https://arxiv.org/abs/1908.10084)
* **💻 GitHub: Sentence Transformers Codebase**Explore the source code, models, pooling strategies, and training methods.[**Check the repo**](https://github.com/UKPLab/sentence-transformers)

## Learning Resources: FAISS

**🔰 Easy Start**

* **📔Faiss: The Missing Manual**

Manual with blogs and videos explaing FAISS from a highlevel overview to an in-depth level.

[**Faiss: The Missing Manual**](https://www.pinecone.io/learn/series/faiss/)

* **🔍 Simple Python Example Using FAISS**A quick notebook-style tutorial showing how to set up FAISS, index embeddings, and retrieve similar vectors.[**See this GitHub example**](https://github.com/facebookresearch/faiss/wiki/Getting-started)

**🛤️ Intermediate**

* **🧪 Official FAISS Wiki (Facebook Research)**Explains indexing strategies, GPU/CPU usage, quantization, and more. Ideal if you want to optimize your searches or use FAISS at scale.[**Explore the Wiki**](https://github.com/facebookresearch/faiss/wiki)

**🚀 Deep Dive**

* **📄 FAISS Research Paper: “FAISS: A Library for Efficient Similarity Search”**The original paper from Facebook AI Research outlining the design, performance, and scalability of FAISS.[**Read the paper**](https://arxiv.org/abs/1702.08734)
* **📘 FAISS Index Types and Trade-offs (Official Guide)**Detailed descriptions of flat vs. quantized indexes, IVF, HNSW, PQ, and more — great for architecture-level understanding.[**Study here**](https://github.com/facebookresearch/faiss/wiki/Faiss-indexes)
* **📊 Benchmarks: FAISS vs Other Vector Databases**In-depth comparisons of FAISS vs alternatives like Annoy, HNSWLib, and ScaNN.[**View benchmark results**](https://github.com/erikbern/ann-benchmarks)

## Learning Resources: NumPy

**🔰 Easy Start**

* **📺 YouTube:"Python NumPy Crash Course" by freeCodeCamp**A beginner-friendly video that covers basics of Numpy.[**Watch here**](https://www.youtube.com/watch?v=QUT1VHiLmmI)
* **📘 Official NumPy Beginner Tutorial**Official documentation of numpy for Absolute Beginners[**NumPy: the absolute basics for beginners**](https://numpy.org/doc/stable/user/absolute_beginners.html)
* **📒 NumPy Illustrated (by Jay Alammar)**Visual explanation of key NumPy concepts [**A Visual Intro to NumPy and Data Representation**](https://jalammar.github.io/visual-numpy/)

**🛤️ Intermediate**

* **📗 NumPy Documentation**

Get deeper dive inside the numpy library straight from the official documentation

[**Numpy Documentation**](https://numpy.org/doc/stable/user/index.html#user)

* **📖Numpy chapter of Python Data Science Handbook by Jake VanderPlas**

Explore basics of Machine learning with Numpy, pandas, Matplotlib and Scikit Learn.

[**Introduction to NumPy**](https://jakevdp.github.io/PythonDataScienceHandbook/02.00-introduction-to-numpy.html)

* **🧰 Cheat Sheet: NumPy for Data Science**A downloadable quick-reference for array operations, math functions, reshaping, and performance tips.[**View and download**](https://github.com/numpy/numpy/blob/main/doc/source/user/quickstart.rst)

**🚀 Deep Dive**

* **📖 Broadcasting: Working with arrays of different shapes.**A deeper look at how broadcasting actually works[**Broadcasting**](https://numpy.org/doc/stable/user/basics.broadcasting.html)
* **🔍 Performance Optimization with NumPy**Learn how to write fast, vectorized code with tips on avoiding loops, using memory efficiently, and benchmarking.[**RealPython Numpy Array Programming**](https://realpython.com/numpy-array-programming/)
* **📄 NumPy Under the Hood**A detailed look at how NumPy arrays work internally, with insights into memory layout, C-extensions, and dtype mechanics.[**https://scipy-lectures.org/advanced/advanced\_numpy/**](https://scipy-lectures.org/advanced/advanced_numpy/)

## Learning Resources: Pytorch

**🔰 Easy Start**

* **📺 PyTorch for Deep Learning (Full Course) by freeCodeCamp**  
  A beginner-friendly 6-hour course that walks through the basics of tensors, models, training loops, and building your first neural networks.  
  [Watch here](https://www.youtube.com/watch?v=GIsg-ZUy0MY)
* **📘 PyTorch 60-Minute Blitz (Official)**  
  Hands-on introduction by PyTorch itself. Teaches tensors, autograd, and training a simple neural network.  
  [Start here](https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html)
* **👩‍🏫 Learn PyTorch from Scratch (GeeksforGeeks)**  
  Simple and progressive explanation of PyTorch fundamentals [Read here](https://www.geeksforgeeks.org/introduction-to-pytorch/)

**🛤️ Intermediate**

* **📝 Learn Pytorch in a day (Daniel Bourke)**  
  YouTube video that helps learn pytorch in 24 hours.  
  [Learn Pytorch in a day](https://www.youtube.com/watch?v=Z_ikDlimN6A)
* **🔧 PyTorch Tutorials: Pytorch official documentation**  
  Official documentation for pytorch.  
  [Explore tutorials](https://pytorch.org/tutorials/)
* **💻 Hands-On with PyTorch Notebooks (by Yann LeCun's team)**  
  A practical collection of interactive notebooks for training neural networks, CNNs, and GANs using PyTorch.  
  [View the GitHub](https://github.com/yunjey/pytorch-tutorial)

**🚀 Deep Dive**

* **📖 PyTorch Internals (by Edward Raff)**  
  A deep dive into how PyTorch works under the hood — including autograd mechanics, tensor operations, and backpropagation internals.  
  [Read here](https://github.com/EdwardRaff/DeepLearningBook/blob/master/notes/pytorch.md)
* **⚙️ PyTorch Profiler Tutorial**  
  Benchmark and optimize your training loops, memory usage, and GPU utilization.  
  [Profiler docs](https://pytorch.org/tutorials/recipes/recipes/profiler_recipe.html)
* **🧠 From Scratch: Building PyTorch Autograd**  
  Recreate a mini version of PyTorch’s automatic differentiation engine to really understand how gradients work.  
  [Walkthrough](https://pytorch.org/tutorials/beginner/transformer_tutorial.html)
* **🧪 "The Annotated Transformer" (Harvard NLP)**  
  Builds a full Transformer model using PyTorch. It is explained with code and intuition.  
  [Read here](http://nlp.seas.harvard.edu/2018/04/03/attention.html)

## Some more Links

### Regular expressions

🔬Learn Regex in Depth **-** [**https://www.rexegg.com/**](https://www.rexegg.com/)

**Website for trying regular expressions** - <Regex101.com>

### Logging

🚩Geeks for Geeks post for logging in python **-** [**https://www.geeksforgeeks.org/logging-in-python/**](https://www.geeksforgeeks.org/logging-in-python/)

📄Official Python documentation for Logging **-** [**https://docs.python.org/3/howto/logging.html**](https://docs.python.org/3/howto/logging.html)

### Datetime

📝Get started with datetime **-** [**https://www.w3schools.com/python/python\_datetime.asp**](https://www.w3schools.com/python/python_datetime.asp)

📄Official python documentation for datetime **-** [**https://docs.python.org/3/library/datetime.html**](https://docs.python.org/3/library/datetime.html)

### PyMuPDF

📄Official documentation for PyMuPDF **-** [**https://pymupdf.readthedocs.io/en/latest/index.html**](https://pymupdf.readthedocs.io/en/latest/index.html)

### SQLite

📽️Freecode camp video to learn SQL **-** [**https://www.youtube.com/watch?v=HXV3zeQKqGY**](https://www.youtube.com/watch?v=HXV3zeQKqGY)

📄Documentation for using SQLite in python **-** [**https://docs.python.org/3/library/sqlite3.html**](https://docs.python.org/3/library/sqlite3.html)

# Future Improvements / Fun Ideas

## Try different Models

## File Validation

## Evalaution of model pending

## Support for multiple Document types

## Scale model for multiple PDFs

## Fine tune based on Feedback