# Transcript Q&A Search System: Process Documentation

This document outlines the steps, design decisions, and usage instructions for the Transcript Q&A Search System, implemented both as a CLI and a Streamlit web app.

## 1. Project Overview

- **Goal: Build a semantic search system for a timestamped transcript. Users can ask questions and receive the most relevant transcript passages along with their timestamps.**
- **Modes Implemented:**
  - **TF-IDF + Cosine Similarity (tfidf)**
  - **Hugging Face Sentence Transformer (llm2)**
  - **Google Gemini Embeddings + Generative Answer (llm1)**
- **Interfaces:**
  - **Command-Line (CLI) in transcript.py**
  - **Web App using Streamlit in transcript_qna_web.py**

# Components of the Semantic Search System

## Web App Interface

A user-friendly web application for access

## TF-IDF + Cosine Similarity

A traditional method for text similarity

## Command-Line Interface

A text-based interface for direct interaction

## Hugging Face Sentence Transformer

A modern approach using pre-trained models

## Google Gemini Embeddings

Advanced embeddings for generative answers

## 2. Transcript Parsing & Chunking

1. **Loading the Transcript**
    - Read lines from a plain-text file (**transcript.txt**), stripping empty lines.
2. **Timestamp Extraction**
    - Use regex **^\[(\d{2}:\d{2})[?::\d{2}]?\s*-\s*(\d{2}:\d{2})[?::\d{2}]?\]** to capture start and end times.
3. **Chunking Strategy**
    - Default chunk size: **5 lines** per chunk (configurable).
    - For each chunk group:
        - Determine **start_ts** from the first line and **end_ts** from the last line (with fallback to nearest valid timestamps).
        - Strip timestamps and combine lines into one text string.
    - Result: A list of chunks, each with **id**, **timestamp**, and **text** fields.

**Rationale:** 5-line chunks balance context and precision. Smaller chunks may be too granular; larger ones may dilute relevance.

## 3. Search & Retrieval Methods

### 3.1 TF-IDF Search (tfidf)

- **Library: scikit-learn**
- **Process:**

1. Vectorize all chunk texts (unigrams + optional stop-word removal).
2. Vectorize the user query.
3. Compute cosine similarity between query vector and chunk matrix.
4. Return top-**k** chunks with non-zero similarity.

### 3.2 Semantic Search via Sentence-Transformers (llm2)

- **Library: sentence-transformers**
- **Model: all-MiniLM-L6-v2**
- **Process:**
    1. Encode all chunk texts into embeddings (cached).
    2. Encode the user query.
    3. Perform **util.semantic_search** to get top-**k** highest-scoring chunks above a threshold.

### 3.3 Google Gemini Embeddings & Generative QA (llm1)

- **Library: google.generativeai**
- **Embedding Model: models/embedding-001**
- **Generative Model: gemini-1.5-flash** (or **gemini-pro** as needed)
- **Process:**
    1. **Precompute** embeddings for all chunks once at startup.
    2. Compute embedding for the query.
    3. Cosine similarity ranking to select top-**k** chunks.
    4. Construct a prompt with chunk excerpts and ask Gemini to generate a concise (2-3 sentence) answer.

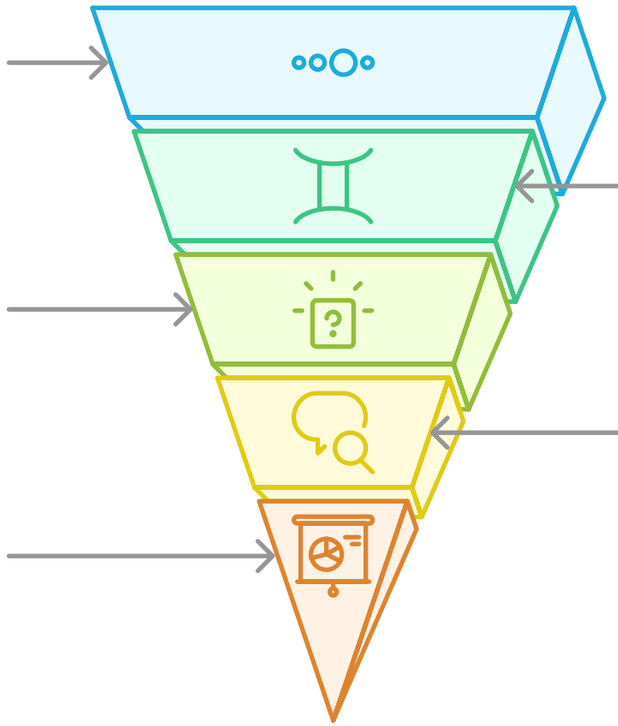
## 4. CLI Implementation (transcript.py)

- **Entry Point: main() parses sys.argv:**

```
python transcript.py transcript.txt tfidf
python transcript.py transcript.txt llm1
python transcript.py transcript.txt llm2
```

- **Workflow:**
    1. **Load & chunk transcript.**
    2. **(For llm1): precompute Gemini embeddings.**
    3. **Enter interactive prompt:**
        - **Prompt: Transcript loaded. You can now ask questions.**
        - **Exit on entering 8.**
        - **Allow switching modes via switch to <mode>.**
    4. **On each query:**
        - **Call the corresponding search function.**
        - **Display the top chunks and, for Gemini, also the generated answer and source timestamps.**

# Transcript Q&A Workflow



**Load Transcript**

Load and prepare the transcript data

**Precompute Embeddings**

Generate embeddings for Gemini mode

**Enter Interactive Prompt**

Engage with the user for questions

**Process Query**

Execute the search based on the query

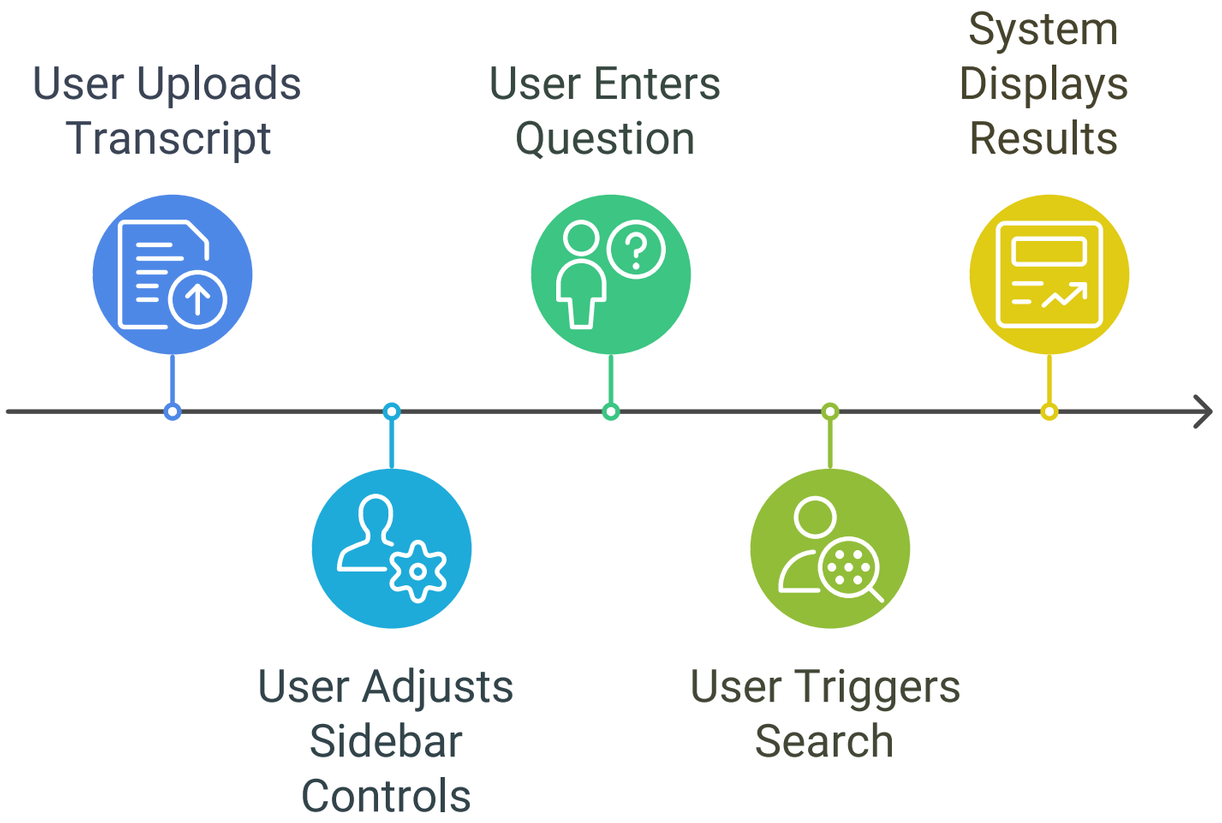
**Display Results**

Show the top chunks and answers

## 5. Web App Implementation (transcript_qna_web.py)

- **Framework: Streamlit**
- **Key Features:**
    - **File uploader for transcript.txt.**
    - **Sidebar controls:**
        - **Lines per chunk (1–15)**
        - **Top-k results (1–10)**
        - **Mode selector (TF-IDF, Sentence Transformer, Gemini AI)**
    - **Main area:**
        - **Display success message with chunk count.**
        - **Text input for the user question.**
        - **Search button triggers selected mode.**
        - **Expanders show each chunk, similarity score, and—if Gemini—AI-generated answer.**

# Streamlit Web App Workflow



## 6. Usage Instructions

1. **Clone repository**:

```
git clone https://github.com/yourusername/transcript-search.git
cd transcript-search
```

2. **Install dependencies**:

```
pip install -r requirements.txt
```

3. **Set Gemini API Key (optional)**:

```
export GEMINI_API_KEY="YOUR_KEY_HERE"
```

4. **Run CLI**:

```
python transcript.py transcript.txt tfidf
python transcript.py transcript.txt llm2
python transcript.py transcript.txt llm1
```

5. **Run Web App**:

```
streamlit run transcript_qna_web.py
```

6. **Exit CLI**: Type **8** at the prompt.

## 7. Testing & Sample Outputs

In **screenshots/** you will find console screenshots for four sample questions in each mode. In **output.txt**, the pasted outputs include:

1. **Question:** "example of machine learning"
    - TF-IDF chunk(s) with **[00:10 - 00:15]**
    - Hugging Face chunk(s)
    - Gemini answer: "A machine learning example is a system predicting a person's weight based on their height. This system improves over time with more data, creating lines to predict future data"
2. **Question:** "artificial intelligence"
3. **Question:** "machine learning"
4. **Question:** "Next steps for deployment?"

*[Full outputs in **output.txt**.]*

## 8. Dependencies (requirements.txt)

- Python ≥ 3.8
- scikit-learn
- numpy
- sentence-transformers
- google-generativeai
- streamlit (for web)
- torch

- transformers

## 9. Bonus Notes

- The Streamlit UI closely mirrors CLI functionality and adds
    - Progress indicators when loading models/embeddings
    - Expanders for readability
- You can deploy the app via Streamlit Cloud or any other hosting by setting **GEMINI_API_KEY** in environment variables.

**End of Document**