**Process Followed**

1. **Transcript Preparation and Loading:**  
   The project begins by preparing the transcript file, a plain text document containing time-stamped dialogue or content. A dedicated function reads the transcript line by line, extracting timestamps and corresponding text chunks using regular expressions, and stores them as manageable units for efficient searching.
2. **Text Chunking:**  
   Each transcript line acts as a chunk, simplifying the matching process. This chunking ensures that the semantic search returns coherent, time-referenced passages rather than isolated words or sentences.
3. **Implementation of Search Methods:**  
   Three semantic search approaches were implemented independently for flexibility and comparison:
   * **TF-IDF Vectorization with Cosine Similarity:** Converts transcript chunks and queries into numerical vectors to compute similarity scores based on keyword frequency.
   * **OpenAI Embeddings:** Utilizes the OpenAI API to convert text and queries into semantic embeddings that capture deeper language context, enabling more accurate matching.
   * **Hugging Face Transformer Embeddings:** Leverages pre-trained transformer models from Hugging Face to generate semantic vectors, offering a powerful open-source alternative for embedding generation.
4. **Command-Line Interface (CLI):**  
   A user-friendly CLI was created to allow users to specify the transcript file and choose the search method via command-line arguments. The interface repeatedly prompts users for questions and returns the best matching transcript chunk along with its timestamp until the user exits.
5. **Optional Web Interface:**  
   For ease of access, a simple Flask-based web application was developed, replicating CLI functionality with an input form and dropdown for method selection. It provides real-time answers on a webpage without requiring command-line interaction.
6. **Testing and Validation:**  
   Unit tests were written to verify the correctness of the TF-IDF search functionality. Sample queries were tested to ensure that the system accurately returns relevant transcript passages.
7. **Documentation and Packaging:**  
   The project includes a README file explaining the setup, usage, dependencies, and design choices. A requirements.txt file lists all Python libraries needed. A Dockerfile was added to enable easy deployment if desired.
8. **Version Control and Submission:**  
   The entire project was managed using Git for version control, with the final codebase pushed to a GitHub repository to facilitate easy sharing, collaboration, and evaluation.