**AI Quiz Tutor - Application Documentation**

**1. Introduction / Overview**

**1.1. Purpose**

The AI Quiz Tutor is a Streamlit web application designed to help users learn and assess their understanding of content from uploaded documents. It intelligently processes documents, identifies key themes and objectives, and automatically generates interactive multiple-choice quizzes based on the document's content.

**1.2. Key Features**

* **Document Upload:** Supports various document formats (DOCX, PDF, PPTX, TXT).
* **AI-Powered Document Analysis:**
  + Intelligent chunking of document content using advanced document understanding libraries.
  + Automatic determination of the document's core subject and primary learning objective using Generative AI.
* **AI-Powered Quiz Generation:** Creates relevant multiple-choice questions with plausible distractors based on the document's content.
* **Interactive Quiz Interface:** User-friendly interface for taking quizzes and receiving immediate feedback.
* **Quiz Summary & Performance Review:** Provides a summary of quiz performance, including score and a review of incorrectly answered questions with full answer text.
* **Document Coverage Heatmap:** Visualizes which sections of the document have been covered by quiz questions and the user's performance on them. Allows users to explore specific document chunks.
  + Statuses include: Not Quizzed, Correct, Incorrect (1x), Incorrect (2+x), and Reviewed.
* **"Quiz me on this chunk" Feature:** Allows users to request a focused quiz question directly from a selected chunk in the heatmap, enabling targeted learning.

**1.3. Target Audience**

This application is designed for a wide range of users, including:

* **Students:** To test their comprehension of study materials.
* **Professionals:** To quickly assess understanding of reports, technical documents, or training materials.
* **Educators:** As a tool to help generate quiz ideas or assess material coverage.
* Anyone looking to actively engage with and learn from textual documents.

**1.4. Technologies Used**

* **Frontend & Web Application:** Streamlit
* **Backend Logic:** Python
* **Generative AI (Embeddings & Content Generation):** Google Gemini API (specifically gemini-1.5-flash for generation, models/text-embedding-004 for embeddings) via the google-generativeai library.
* **Document Processing & Chunking:** docling library and its dependencies (including docling-core, transformers, sentence-transformers).
* **Semantic Search Index:** FAISS (Facebook AI Similarity Search).
* **File Parsing:** python-docx, PyPDF2, python-pptx.

**2. User Guide**

**2.1. Accessing the Application**

The application is deployed and accessible at the following URL:

* <https://ai-quiz-tutor-upload.streamlit.app/>

No special system requirements are needed beyond a modern web browser (like Chrome, Firefox, Safari, or Edge).

**2.2. Step-by-Step Usage**

**A. Uploading a Document:**

1. **Initial Screen:** When you first open the app, you will see the title "AI Quiz Tutor."
2. Below the title, you will find a "Data Privacy" notice (hover over it for details on data handling).
3. Use the file uploader widget labeled "Upload your document (DOCX, PDF, PPTX, or TXT)" to select a file from your computer. You can either drag and drop a file onto the designated area or click the "Browse files" button.
   * Supported file types are: .docx, .pdf, .pptx, and .txt.

**B. Document Processing:**

1. Once you select and upload a file, the application will begin processing it.
2. During processing, the app performs several steps:
   * **Chunking:** The document is divided into meaningful semantic chunks.
   * **Theme Analysis:** The AI determines the core subject and primary learning objective of the document.
   * **Vectorization:** Text chunks are converted into numerical representations (embeddings) for semantic search.
   * **Index Building:** A FAISS index is built from these embeddings.
3. You will see progress bars for embedding generation and chunk label generation.
4. When processing is complete, a message "Document Analyzed and ready to test your knowledge" will appear. Below this, the determined **Subject** and **Document objective** will be displayed.

**C. Starting the Main Quiz:**

1. Once the document is analyzed, a **"Start Quiz!"** button will become active.
2. Click this button to begin the quiz. The first question should appear after a brief loading period.

**D. Answering Questions:**

1. The document name will be displayed at the top as a caption.
2. Each question will be displayed with four multiple-choice options (A, B, C, D).
3. Select your answer using the radio buttons.
4. Click the **"Submit Answer"** button.
5. **Feedback:**
   * You'll receive immediate feedback: "Correct!" or "Incorrect. Correct was: [Correct Letter]."
   * An explanation based on the document context will also be shown.
6. **Navigating Questions:**
   * After submitting an answer, a **"Next Question"** button will appear. Click this to proceed.
   * The type of next question (new topic or related to the previous one if answered incorrectly) is determined by the application's logic.

**E. Stopping the Quiz:**

* At any point during the quiz (after submitting an answer or before), you can click the **"Stop Quiz"** button. This will end the current quiz session and take you to the Quiz Summary page.

**2.3. Quiz Summary Page**

When you stop the quiz or complete it, you will be taken to the "Quiz Summary" page. The page title will now be "Quiz Summary".

1. **Document Name:** The name of the processed document will be displayed as a caption under the title.
2. **Score and Statistics:**
   * Your overall score (percentage).
   * Total questions answered.
   * Number of correct and incorrect answers.
3. **Review Topics for Incorrect Answers (Collapsible):**
   * This section is collapsed by default (unless you expanded it in the current session). Click on its title to expand it.
   * It lists all questions you answered incorrectly, showing:
     + The question text.
     + Your answer (letter and full text).
     + The correct answer (letter and full text).
     + The explanation.
4. **Document Coverage & Performance Heatmap (Collapsible):**
   * This section is also collapsed by default (unless you expanded it in the current session). Click on its title to expand it.
   * **Legend:** Explains the meaning of the colored squares:
     + 🟦 **Not Quizzed:** Chunk not yet covered by a question you answered.
     + 🟩 **Correct:** Chunk was part of the context for a question you answered correctly.
     + 🟨 **Incorrect (1x):** Chunk was part of the context for a question you answered incorrectly once.
     + 🟥 **Incorrect (2+x):** Chunk was part of the context for questions you answered incorrectly multiple times.
     + 🟣 **Reviewed:** You viewed the content of this chunk via the heatmap but haven't been quizzed on it directly, or it was "Not Quizzed" and became "Reviewed" upon viewing or being part of a focused quiz context before being answered.
   * **Heatmap Grid:** Displays the document's headings and a grid of squares representing the chunks. Hovering over a square shows a tooltip with the beginning of the chunk's text.
   * **Clicking a Square:**
     + If the square was "Not Quizzed" (blue), it will turn "Reviewed" (purple).
     + A pop-up (expander) will appear **above** the "Document Coverage & Performance Heatmap" section.
     + **Pop-up Label:** Shows Path: [Full Heading Path] (Context for Paragraph Index [Index]).
     + **Pop-up Content:** Displays the text of the clicked chunk (bolded), and for context, the text of its immediate preceding and succeeding neighbors if they share the same heading structure, separated by horizontal lines.
     + **"Quiz me on this chunk" Button:**
       - Clicking this initiates a focused, single-question quiz.
       - The app will navigate to a quiz interface. A question will be generated based on the selected chunk and its semantically similar neighbors.
       - Answer the question and submit.
       - If **correct**, scores/heatmap are updated, and you'll see a "Back to Quiz Summary" button.
       - If **incorrect**, scores/heatmap are updated. You'll see feedback and two buttons: "Try Another Question on this Topic" (generates a new question on the same topic) and "End Focused Quiz & View Summary".
       - The main "Stop Quiz" button at the bottom of the quiz interface also changes to "End Focused Quiz & View Summary" in this mode.
     + **"Close Detail" Button:** Closes the pop-up.
5. **"Start New Quiz Once More" Button:**
   * Resets the quiz statistics (score, answered questions) and heatmap statuses for the *currently loaded document*.
   * Allows you to take a fresh quiz on the same document. The document itself is not re-processed.

**2.4. Data Privacy**

(This text is also available by hovering over "Data Privacy" on the initial upload screen)

To provide quiz features, this application processes your uploaded document. Snippets of your document are sent to Google's Generative AI services to generate relevant content. Google's API policies state that this data is not used to train their general models. No original documents are stored by this application after your session ends.

**3. For Developers / Self-Hosting**

*This section is for users who want to run or modify the application's source code.*

**3.1. Prerequisites**

* Python 3.11 installed.
* Git for cloning the repository.
* An internet connection for downloading packages and accessing the Gemini API.

**3.2. Setup**

1. **Clone the GitHub Repository:** bash git clone [https://github.com/impensjonathan/ai-quiz-tutor-upload.git](https://github.com/impensjonathan/ai-quiz-tutor-upload.git) cd ai-quiz-tutor-upload
2. **Create and Activate a Python Virtual Environment:** It's highly recommended to use a virtual environment (e.g., named py311\_env). bash python3.11 -m venv py311\_env source py311\_env/bin/activate (On Windows: py311\_env\Scripts\activate)
3. **Install Dependencies:** Ensure you have a requirements.txt file in the repository root with all necessary packages and their pinned versions. bash pip install -r requirements.txt Key packages include streamlit, google-generativeai, numpy, faiss-cpu, python-docx, PyPDF2, python-pptx, docling, docling-core, transformers, sentence-transformers.

**3.3. Configuration**

1. **Gemini API Key:**
   * You need a valid API key for Google's Gemini models. You can obtain one from Google AI Studio ([aistudio.google.com/app/apikey](https://aistudio.google.com/app/apikey)).
   * Create a folder named .streamlit in the root of your project directory (ai-quiz-tutor-upload).
   * Inside the .streamlit folder, create a file named secrets.toml.
   * Add your API key to secrets.toml in the following format: toml GEMINI\_API\_KEY = "YOUR\_ACTUAL\_API\_KEY\_VALUE"
2. **Application Constants (in app.py):** Several constants at the beginning of app.py can be tuned:
   * CORE\_SUBJECT: Default subject if AI theme determination fails.
   * EMBEDDING\_MODEL: Specifies the Google embedding model used (e.g., "models/text-embedding-004").
   * NUM\_CONTEXT\_CHUNKS\_TO\_USE: Number of chunks used as context for generating questions.
   * MIN\_WORDS\_FOR\_CONTENT\_CHUNK: Minimum words for a chunk to be considered substantive after Docling processing.
   * NUM\_CHUNKS\_TO\_FETCH\_SEMANTICALLY: How many chunks FAISS initially retrieves for semantic similarity searches.

**3.4. Running Locally**

1. Ensure your virtual environment (e.g., py311\_env) is active.
2. Navigate to the project's root directory in your terminal.
3. Run the Streamlit application: bash streamlit run app.py or, to be specific with the Python interpreter: bash python3.11 -m streamlit run app.py
4. Open the local URL provided in your terminal (usually http://localhost:8501) in a web browser.

**4. How Generative AI and AI-Supported Tools Are Used**

**4.1. Overview of AI Integration** The AI Quiz Tutor leverages several AI and machine learning technologies to understand your documents, generate relevant educational content, and enable semantic search capabilities. This involves interactions with external Generative AI models provided by Google and the use of local AI-supporting libraries for document parsing and search.

**4.1.1. Specific AI Models Used by This Application**  It's important to note that the AI models used by the "AI Quiz Tutor" application for its tasks are explicitly defined within its app.py code. This is distinct from any conversational AI model you might be interacting with (like the one assisting you with this code).

* **For Text Generation:** The application is configured to use **gemini-1.5-flash** for all text generation tasks. This includes generating quiz questions, determining document themes and objectives, and creating chunk labels. This model is initialized in the code with st.session\_state.gemini\_model = genai.GenerativeModel('gemini-1.5-flash').
* **For Text Embeddings:** To convert document chunks into numerical vectors (embeddings) for semantic search, the application uses Google's **models/text-embedding-004**. This is specified by the EMBEDDING\_MODEL constant in the code and used with genai.embed\_content().

These model choices were made considering a balance of capability and accessibility (including free tier usage). If different performance characteristics or features are desired (e.g., potentially higher quality generation from a model like gemini-1.5-pro), the model name strings in app.py can be changed, keeping in mind any differences in API availability or associated costs.

**4.2. Google Gemini API for Text Generation** (Previously 4.1, content adjusted slightly) The application uses the specified Google Gemini model (gemini-1.5-flash) via the google-generativeai Python library for several key generative tasks:

* **Quiz Question Generation:** (As described in 1.4.1)
* **Document Theme & Objective Determination:** (As described in 1.4.1)
* **Chunk Labels (for Heatmap Tooltips):** (As described in 1.4.1)

*In all these generative tasks, relevant text portions from the user's uploaded document (or summaries/samples thereof) are formatted into carefully constructed prompts and sent to the Google Gemini API for processing.*

**4.3. Google Gemini API for Embeddings** (Previously 4.2, content adjusted slightly)

* To enable semantic understanding of the document content, the application converts text chunks into numerical vector representations called "embeddings."
* This is achieved using the specified Google embedding model (models/text-embedding-004), accessed through the genai.embed\_content() function.
* Each substantive document chunk (identified by docling) is sent to this API, and the resulting embedding vector captures the semantic meaning of that chunk.

**4.4. Docling - AI-Powered Document Understanding** (Previously 4.3)

* The docling library (and docling-core) is fundamental for the initial, intelligent processing and parsing of the uploaded documents (DOCX, PDF, PPTX, TXT).
* **Document Conversion & Structuring:** docling.document\_converter.DocumentConverter is used to parse these various file formats. This component likely uses sophisticated methods, potentially including internal AI/ML models, to understand document layout, identify structural elements (headings, paragraphs, lists, tables), and extract clean text content.
* **Semantic Chunking:** The docling.chunking.HybridChunker is then used to segment the document into meaningful pieces. This is not a simple fixed-size split; it aims to create semantically coherent chunks by leveraging a tokenizer from the Transformers library (specifically configured with sentence-transformers/all-MiniLM-L6-v2 model's tokenizer). This helps ensure that chunks are contextually relevant for subsequent AI processing. The chunker also associates extracted heading information with each chunk.

**4.5. FAISS - Supporting AI-Driven Search** (Previously 4.4)

* FAISS (Facebook AI Similarity Search) is a library for efficient similarity search among dense vectors.
* While FAISS itself is an algorithmic library rather than an AI model, it is a critical enabler for AI-powered semantic search.
* The application builds an in-memory FAISS index from the embeddings generated by the Gemini API.
* This FAISS index is then queried by the generate\_quiz\_question function to quickly find chunks that are semantically most similar to a given query.

**4.6. Transformers Library** (Previously 4.5)

* The Transformers library from Hugging Face is a widely used toolkit for working with state-of-the-art machine learning models, especially for Natural Language Processing.
* In this application, transformers.AutoTokenizer is used to load a tokenizer compatible with the sentence-transformers/all-MiniLM-L6-v2 model. This tokenizer is then passed to docling's HuggingFaceTokenizer for use within the HybridChunker.

**5. Application Architecture Overview** (Renumbered)

**5.1. Key Python Script:**

* app.py: Contains all the Streamlit UI logic, session state management, function definitions for document processing, AI interactions, and quiz flow.

**5.2. Project File Structure (Typical):**

AI\_Quiz\_Tutor\_Upload/

│

├── app.py # Main Streamlit application script

├── requirements.txt # Python dependencies

├── .streamlit/ # Streamlit configuration folder

│ └── secrets.toml # API keys and other secrets

└── (py311\_env/) # Python virtual environment (typically not in Git)

**5.3. Component Interaction Flow (Text-based Description):**

* **User Interaction (Browser via Streamlit Frontend Server on share.streamlit.io)**
  + Uploads documents, clicks buttons, answers questions. ↓ ↑
* **Streamlit Python Backend (app.py running on Streamlit Cloud Server)**
  + **File Ingestion & Initial Parsing:**
    - st.file\_uploader receives the file.
  + **Document Processing Pipeline:**
    - **process\_document\_with\_docling() [AI-SUPPORTED LOCAL LIBRARY OPS]:**
      * Uses docling.DocumentConverter to parse raw file formats.
      * Uses docling.HybridChunker with a Transformers (sentence-transformers/all-MiniLM-L6-v2) tokenizer for semantic chunking based on document structure and headings.
      * *Output:* A list of processed text chunks with their associated heading metadata.
  + **Content Analysis & Embedding Generation:**
    - determine\_document\_theme():
      * *Input:* Sampled text chunks.
      * **Google Gemini API Call (gemini-1.5-flash) [AI-SUPPORTED EXTERNAL API]:** Generates "Core Subject" and "Primary Objective" for the document.
    - generate\_chunk\_labels():
      * *Input:* All substantive chunks.
      * **Google Gemini API Call (gemini-1.5-flash) [AI-SUPPORTED EXTERNAL API]:** Generates concise topic labels for heatmap tooltips.
    - setup\_vector\_store():
      * *Input:* All substantive text chunks.
      * **Google Gemini API Call (models/text-embedding-004) [AI-SUPPORTED EXTERNAL API]:** Called via genai.embed\_content() to convert each chunk into a vector embedding.
      * **FAISS Index Building [AI-SUPPORTING LOCAL LIBRARY OPERATION]:** An in-memory FAISS index is created from these embeddings.
  + **Quiz Generation & Interaction:**
    - generate\_quiz\_question():
      * *Context Selection:* Uses various strategies including:
        + Shuffled available chunks (for initial questions).
        + FAISS semantic search (**FAISS Index [AI-SUPPORTING LOCAL LIBRARY OPERATION]**) based on previous question text or a focused\_chunk\_idx (for "simpler," "harder," or "Quiz me on this" scenarios). The FAISS search queries embeddings generated by Gemini.
      * **Google Gemini API Call (gemini-1.5-flash) [AI-SUPPORTED EXTERNAL API]:** A detailed prompt containing the selected context chunks and instructions is sent to generate a multiple-choice question, options, the correct answer, and an explanation.
    - User answers are processed, scores updated, and feedback provided.
    - Heatmap statuses (chunk\_review\_status) are updated based on quiz performance or direct review.
  + **State Management:**
    - st.session\_state holds all application state (uploaded data, processed content, quiz progress, UI flags) across user interactions and script reruns.
  + **UI Rendering:**
    - Standard Streamlit commands (st.title, st.button, st.radio, st.expander, st.markdown, st.success, st.info, st.warning, st.error) display the application interface and dynamic content.
    - display\_heatmap\_grid() renders the heatmap visualization. The associated chunk detail expander is rendered conditionally on the summary page.

**5.4. Key Functions in app.py** \* setup\_vector\_store(): Generates embeddings via Gemini and builds the local FAISS index. \* determine\_document\_theme(): Uses Gemini to find the document's core subject and learning purpose. \* process\_document\_with\_docling(): Uses docling to convert and chunk the uploaded document. \* generate\_chunk\_labels(): Employs Gemini to create short topic labels for each chunk. \* display\_heatmap\_grid(): Renders the heatmap legend and grid. (The detail expander logic is now separate but triggered by this). \* generate\_quiz\_question(): Core logic for creating questions using Gemini, with context from FAISS/other strategies. \* Main conditional blocks in app.py: Manage UI flow for different app states (initial, processing, ready, quiz, focused quiz, summary).

**5.5. Session State Management** st.session\_state is crucial for maintaining state across reruns. Key variables track:

* Uploaded file, processing status (uploaded\_file\_key, vector\_store\_setup\_done).
* Document content (doc\_chunk\_details, substantive\_chunks\_for\_quiz).
* AI-generated data (dynamic\_doc\_subject, dynamic\_doc\_objective, chunk\_hover\_labels).
* FAISS data (faiss\_index).
* Quiz states (quiz\_started, in\_heatmap\_quiz\_mode, current\_question\_data, question\_number, incorrectly\_answered\_questions, total\_questions\_answered, etc.).
* UI flags (show\_summary, show\_heatmap\_chunk\_detail, etc.).

**5.6. Main External Libraries Used**

* streamlit: Web application framework.
* google-generativeai: Interface to Google Gemini AI models.
* docling, docling-core: Document parsing and semantic chunking.
* transformers: Tokenization (used by Docling).
* faiss-cpu: Efficient similarity search for embeddings.
* numpy: Numerical operations.
* python-docx, PyPDF2, python-pptx: Specific file format parsing.

**6. Deployment Notes**

* The application is deployed on Streamlit Community Cloud (share.streamlit.io).
* Deployment requires a GitHub repository with app.py, requirements.txt (pinned versions recommended), and .streamlit/secrets.toml (for private repos) or secrets configured in the Streamlit Cloud UI.
* Resource limits on the free tier of Streamlit Community Cloud should be considered for performance with large documents or high traffic.

**7. Troubleshooting (Brief Common Issues)**

* **API Key Errors:** Check GEMINI\_API\_KEY in Streamlit Cloud secrets settings.
* **Dependency Issues:** Verify requirements.txt and check Streamlit Cloud deployment logs.
* **Performance:** Large documents can be slow to process.
* **LLM Errors:** May relate to API issues, rate limits, or prompt content.