**Real-Time Whiteboard Transcription System**

**Detailed Design**

**Introduction**

The system aims to capture and transcribe whiteboard content in real time. This includes handwritten text, mathematical symbols, The transcribed data can be exported to digital formats such as .docx, .txt, or .pdf.

This document outlines the detailed design of the system, including its architecture, components, data design, and user interface. The system leverages OCR, AI/ML models, and advanced image processing for accuracy and efficiency

**System overview**

The system comprises three main components: user interface, backend processing, and data export.

* **User Interface:** Allows users to upload images or video files containing whiteboard data for transcription.
* **Backend Processing:** Processes visual inputs using OCR and handwriting recognition modules to convert them into text.
* **Data Export:** Integrates APIs to export processed content into various file formats.

**Design Considerations**

* **Assumptions:**
  + Access to high-resolution images or video feeds of whiteboard content.
  + Minimal latency for real-time performance.
* **Constraints:**
  + Initial support for limited languages (English, mathematical symbols)
  + Dependency on third-party LLM/AI mode, e.g. LLAMA and Gemini
* **Standards:**
  + Data is transcribed into text file formats (.docx, .pdf, .txt).

**System Architecture**

The system employs a Model-View-Controller (MVC) architecture to ensure scalability and maintainability:

* **Model:**
  + Handles OCR and handwriting recognition models.
  + Processes and filters whiteboard content to remove irrelevant or erased data.
* **View:**
  + Provides an intuitive interface for uploading videos, viewing transcriptions, and configuring export settings.
  + Supports device-agnostic access, ensuring ergonomic interaction on various monitor resolutions and tab sizes.
* **Controller:**
  + Bridges the Model and View.
  + Handles user inputs, orchestrates OCR processing, and updates the view.

**Graphic Description**

A screenshot of a computer

Description automatically generated

**Component Design**

**User Interface Module:**

* **Purpose:** Provide a seamless interface for data uploads and exporting processed text files.
* **Features:**
  + Options to upload captured videos or images.
  + Real-time display of transcription results.

**OCR Processing Module (Backend):**

* **Purpose:** Perform OCR on visual inputs for text, handwriting, and symbols.
* **Features:**
  + Preprocessing using Pillow or FFMPEG.
  + Model inference for handwriting and mathematical symbols using LLAMA
  + Using Gemini to create an easy-to-read format of the text

**Data Export Module:**

* **Purpose:** Convert transcribed content into user-friendly formats.
* **Features:**
  + Export to .docx, .pdf, or .txt.
  + Ensures consistent formatting across all exported files.

**Data Design**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Function Design**

**1. Real-Time Processing Module:**

* Captures and processes frames from live feeds or uploaded videos.
* Utilizes FFMPEG for frame extraction and preprocessing.

**2. OCR Module:**

* LLAMA 4 for accurate text recognition.
* Using a custom prompt the transcription will be processed and outputted chronologically and with timestamps for easy tracking

**3. Export Module:**

* Converts transcriptions to user-specified formats.
* Ensures formatting consistency across all exports.

**4. User Interaction Module:**

* Dynamic front-end built with React.js for configuring settings, reviewing transcriptions, and exporting files.

**5. APIs and Integration**

External Libraries and Tools:

* Pillow: For frame preprocessing.
* FFMPEG: For video processing and resizing frames.
* LLAMA 4 scout: For handling handwriting and mathematical content.
* Gemini flash 2.0: For creating a chronological flow and easy to read transcribed content.

**Internal APIs:**

* Processing API: Interfaces with OCR and machine learning modules.
* Export API: Facilitates document generation in .docx, .pdf, and .txt formats.
* User Preference API: Stores and retrieves user settings.

**6. Technology Stack:**

* Frontend: React.js for responsive design.
* Backend: Python or Node.js for server-side functionality.

**User Interface Design**

* Allows users to upload videos or images for transcription.
* Displays a preview of the processed content.
* Provides options to configure export file format settings.
* Enables downloading of exported files in chosen formats.

**Algorithm Description**

**1. Frame Extraction & Preprocessing:**

* **Tool Used: FFMPEG**
* **Function:** Extracts frames from uploaded video files or live streams for further analysis
* **Enhancements:** Uses Pillow for resizing and basic preprocessing to improve OCR visibility.
* **Time Complexity:** O(f) where f is the number of frames processed.

**2. OCR & Handwriting Recognition:**

* **Model: NVIDIA NIM using LLaMA 4 Scout**
* **Function:** Recognizes both handwritten text and mathematical formulas in each frame.
* Capabilities: Robust to different handwriting styles, Detects inline math and block equations
* **Time Complexity:** O(n) per frame, where n is the number of character/symbol regions.

**3. Text Structuring & AI Refinement:**

* **Model: Gemini 2.0 Flash via OpenRouter API**
* **Function:** Refines and organizes the raw OCR output, ensuring readability, coherence, and chronological ordering with optional timestamps.
* Features: Groups related text logically Removes noise and repeated content
* **Time Complexity:** Proportional to input size — typically O(t) where t is the text length.

**4. Frame Filtering:**

* **Function: Filters out redundant or static frames by comparing visual differences.**
* **Technique:** Heuristic-based change detection.
* **Benefit:** Increases performance by skipping unchanging content.
* **Time Complexity:** O(f) where f is the number of frames analyzed.

**5. Export Formatting:**

* **Libraries: python-docx, jsPDF, file-saver**
* **Function:** Converts structured transcription into downloadable formats: .docx, .pdf, or .txt.
* Formatting Includes: Consistent styling and spacing, Timestamp tagging.
* **Time Complexity:** O(t) where t is the number of text elements.