**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 OCR tools like Tesseract.
* **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:**

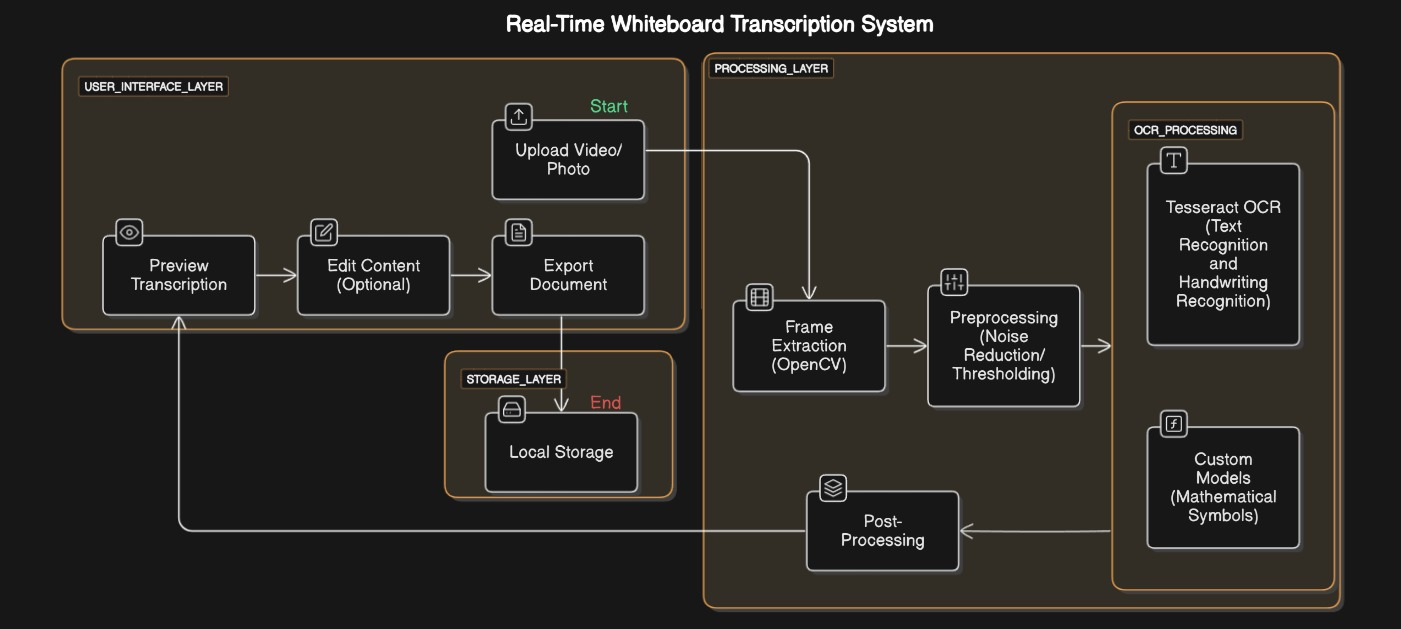
* **Purpose:** Provide a seamless interface for data uploads and exporting processed text files.
* **Features:**
  + Options to upload captured videos or images.
  + Real-time display of transcription results.

**OCR Processing Module (Backend):**

* **Purpose:** Perform OCR on visual inputs for text, handwriting, and symbols.
* **Features:**
  + Preprocessing using OpenCV.
  + Model inference for handwriting and mathematical symbols using TensorFlow or PyTorch.

**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**

**Function Design**

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

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

**2. OCR Module:**

* Tesseract for general text recognition.
* Custom-trained TensorFlow/PyTorch models for handwriting and mathematical symbols.

**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:

* OpenCV: For frame preprocessing and noise reduction.
* Tesseract OCR: For multilingual text recognition.
* TensorFlow/PyTorch: For custom models handling handwriting and mathematical content.
* FFMPEG: For video processing.

**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 live preview of the processed content.
* Provides options to configure export file format settings.
* Enables downloading of exported files in chosen formats.

**Algorithm Description**

**1. Optical Character Recognition (OCR) Pipeline:**

* **Algorithm:** The OCR pipeline processes frames extracted from video or image input. It uses OpenCV for preprocessing, including resizing, denoising, and adaptive thresholding, to enhance text visibility.
* **Time Complexity:** The preprocessing steps operate at O(n), where 'n' is the number of pixels per frame. The OCR recognition using Tesseract operates at O(t), where 't' is the number of text regions detected.

**2. Handwriting Recognition:**

* **Algorithm:** A convolutional neural network (CNN) trained on handwriting datasets processes individual characters or words. The system uses TensorFlow or PyTorch to predict characters based on features extracted from the image.
* **Time Complexity:** Training is O(n · m), where 'n' is the number of features and 'm' is the number of training samples. Inference operates at O(n).

**3. Mathematical Symbol Recognition:**

* **Algorithm:** Employs object detection models (e.g., YOLO or SSD) to identify symbols. Detected symbols are post-processed to assemble mathematical expressions.
* **Time Complexity:** Symbol detection runs at O(k · n), where 'k' is the number of symbols, and 'n' is the size of the image.

**4. Frame Filtering and Content Validation:**

* **Algorithm:** Filters frames with significant changes to ensure only relevant data is processed. Content validation removes noise and redundant data using heuristic rules.
* **Time Complexity:** Frame filtering operates at O(f), where 'f' is the number of frames.

**5. Export Formatting:**

* **Algorithm:** Converts structured text into .docx, .pdf, or .txt formats using libraries like ReportLab and Python-docx.
* **Time Complexity:** Formatting operates at O(n), where 'n' is the number of text elements.