**Acknowledgements**

I thank my project supervisor Ms. Nethmi Weerasingha for guidance, motivating me and always supporting me throughout the project. Her inspiring ideas and constructive suggestions helped define and design the AI Virtual Painter.

I also thank the who gave me support and facilities to help me make and test the application.

Big thanks to my family and my friends and loved ones for bearing with me encouraging me with motivation and constructive criticism during the toughest part of the project. Because of their support, I was able to complete this work successfully.

**Abstract**

The goal of the gesture enabled, real-time painting application DrawWave (AI Virtual Painter) is to completely transform how users engage with digital painting surfaces. To handle tasks like drawing, erasing, and color switching, utilize computer vision libraries like OpenCV and MediaPipe to detect hand motions from a shared camera feed. The software has been implemented in two forms: desktop application with local and offline capabilities and sketch preservation, and web application with real-time collaboration by means of WebSockets and cloud-based backends.

The project obliterates major disadvantages of traditional interactive tools like costly hardware, inadequate portability, and inaccessibility. It's low-cost and portable and is versatile and deployable across vast areas from the educational sector and artistic therapy to co-design and fun activities. Built using the Waterfall model, the process consisted of strict requirement analysis, system design in view of modules, and iterative system testing with fluctuating environmental conditions.

This report describes the life of the DrawWave (AI Virtual Painter) project in desktop and web forms. It emphasizes architectural design, system performance, and cross-platform comparative functionality. The final product is a low-cost and affordable product that puts to use vision-based interfaces to enhance gesture-based creativity. The system with only a standard webcam and free libraries brings digital sketching to the masses, and those from educational, therapeutic, and group environments benefit from its use. The DrawWave (AI Virtual Painter) proves that natural gestures are what have the power to redefine traditional forms of interaction as engaging digital experiences.

Contents

[Introduction 5](#_Toc197310017)

[Background 6](#_Toc197310018)

[Problem Definition 6](#_Toc197310019)

[Reason Behind the Project 6](#_Toc197310020)

[Challenges Faced During the Project 7](#_Toc197310021)

[Desktop Application Challenges 7](#_Toc197310022)

[Web Application Challenges 7](#_Toc197310023)

[Project Objectives 8](#_Toc197310024)

[Short-Term Objectives 8](#_Toc197310025)

[Long-Term Objectives 8](#_Toc197310026)

[Business Objectives 8](#_Toc197310027)

[Deliverables 9](#_Toc197310028)

[User Interfaces 10](#_Toc197310029)

[User Interfaces: Web Application 12](#_Toc197310030)

[Programming Languages and Technology Choices 14](#_Toc197310031)

[Literature Review 15](#_Toc197310032)

[Research Gaps Identified 17](#_Toc197310033)

[Conclusion of Review 17](#_Toc197310034)

# Introduction

DrawWave (AI Virtual Painter) is a new concept in human-computer interaction based on real-time hand gesture recognition. A cross-platform desktop and web client, it enables the user to draw, erase, and manipulate a virtual painting tool with no traditional input device. Made possible through the use of available technologies such as OpenCV and MediaPipe for recognizing gestures and new frontend and backend frameworks, the project aims to deliver an interactive, inclusive, and multi-user enabled painting platform.

Traditional interactive technology like smartboards and graphic tablets are normally barriers because of affordability and mobility. These are problems addressed by DrawWave (AI Virtual Painter) in the way of low-cost, light hardware that turns standard webcams into gesture interfaces. Not only will the system support individual creativity, but also real-time web collaboration, so it is appropriate in distributed design teams, classrooms, and even in art therapy and for activities and games.

This report captures the whole life cycle of DrawWave (AI Virtual Painter) development from analysis and design to system implementation and evaluation. Desktop and web-based systems are compared in this report with variations in architecture, user experience, and technological hurdles. The two divided approaches reveal how gesture recognition and real-time communications technologies are deployable across platforms with minimal variation in core function and user interaction.

During this project's development, usability, scalability, and availability were maintained as top considerations while delivering a robust and pleasant user experience. Development was conducted in accordance with the Waterfall Methodology, and system building followed in a straight line from element requirement collection, design, and iteration in development and testing.

The subsequent sections will summarize the background and research behind this project, describe the process involved in the development, and present a critical evaluation of findings and lessons.

# Background

## Problem Definition

Traditional gesture-based sketching tools and interactive displays such as smartboards and stylus-driven tablets are likely to be expensive, hardware-intensive, and inaccessible to many users like those in low-cost schools and physically disabled individuals. These and many existing solutions are inflexible and are likely to require some control environment in order to be effective. Most existing tools are not capable of providing real-time, multi-user, and collaborative interactions and are weaker in strong accessibility aspects. And this project targets children for interactive fun activities.

## Reason Behind the Project

DrawWave (AI Virtual Painter) was created to bring creative digital tools closer by taking advantage of readily available hardware -- that is, common webcams -- and free software libraries. The project aims to bridge the gap between functionality and availability by providing users with the ability to draw, erase, and interact with a virtual painting through natural hand gestures. It was implemented with two somewhat orthogonal versions: one desktop version to be used by one or several users and one web version to be used by many users in collaboration. The two products attend to different user needs with both offline usage and real-time online collaboration.

The idea came with the increasing need for interactive technology in areas such as education, design, and therapy, in entertainment uses. Most particularly, the project focuses on inclusiveness such that individuals who are low in mobility or who have special needs are able to participate in digital creativity through simple gestures.

## Challenges Faced During the Project

### Desktop Application Challenges

* **Gesture Accuracy**: Maintaining the same gesture recognition in all light and hand orientations was not easy. Mediapipe gave good baseline accuracy but required fine-tuning in different settings.
* **UI Integration**: the live camera feed, gesture tracking overlays, and the canvas into one PyQt5 interface in a smoothly integrated manner necessitated proper threading and performance control.
* **Undo/Redo and Persistence**: Stable save/load using SQLite and robust undo/redo necessitated advanced state management in order to keep a history of all gestures drawn.
* **Simultaneous Input Modes:** Because of the conflict between event systems if you implemented gesture and mouse input at the same time.
* **User Interface Optimization:** Balancing aesthetics with performance in PyQt5 while avoiding UI flicker and layout shifts.

### Web Application Challenges

* **WebSocket Synchronization**: Synchronization of the canvas in real-time between multiple users posed problems in event ordering, network latency, and session consistency.
* **Gesture-to-Command Mapping**: Efficient decoding & categorization and encoding of motions are necessary for mapping camera picture frames (30 FPS )to server-side drawing orders with minimal latency.
* **Scalability and Room Management**: A strong architecture was needed to manage several rooms and users at once with distinct canvas states without any data leaks and with proper sync.
* **Security and Deployment**: New deployment issues were brought up by the support for HTTPS/WSS encrypted connections, managing CORS rules, and configuring TLS certificates in production systems.

Both systems were successfully deployed and used in the target usage regions in spite of these problems. The system development process gained valuable insights in real-time cross-platform system design and demonstrated the potential and shortcomings of vision-based interactive systems.

# Project Objectives

### Short-Term Objectives

* Create and implement a Virtual-drawing board that is manually operated.
* Leverage OpenCV to integrate with Mediapipe to enable real-time detection of hand landmarks.
* Use PyQt5 to create a fully functional desktop application
* Create a cooperative web application with WebSockets and React.
* Developing the basic drawing-commands: color change, erase, and draw.

### Long-Term Objectives

* For accessibility, including voice command capabilities and user-defined gestures.
* Make it possible to save an artwork in the cloud.
* Use specialized machine learning models to broaden the categorization of gestures.
* Add support for various export-formats and include dynamic shape detection mechanisms.
* Simple access to all users, without complex and time consuming setups.

### Business Objectives

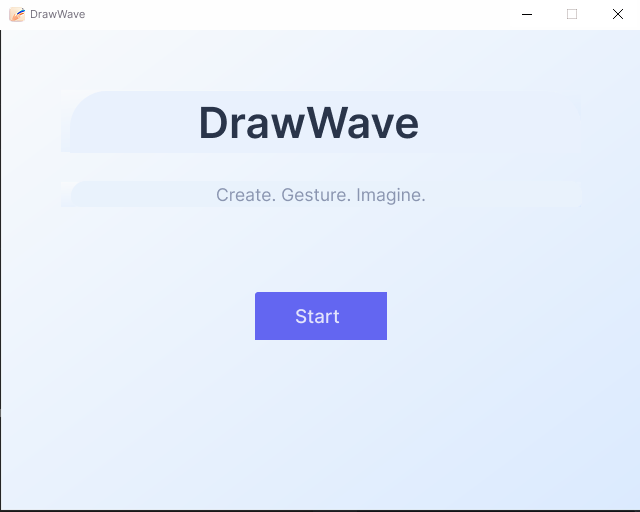
* Offer an cost effective alternative in comparison to interactive whiteboards and digital tablets.
* Enable the uptake of inclusive design by involving persons with mobility impairments in the early stages of idea generation.
* Suggest an open-source method that is easily adaptable across different academic settings.
* Promote digital technological advancements in areas where state-of-the-art hardware is not readily available.

# Deliverables

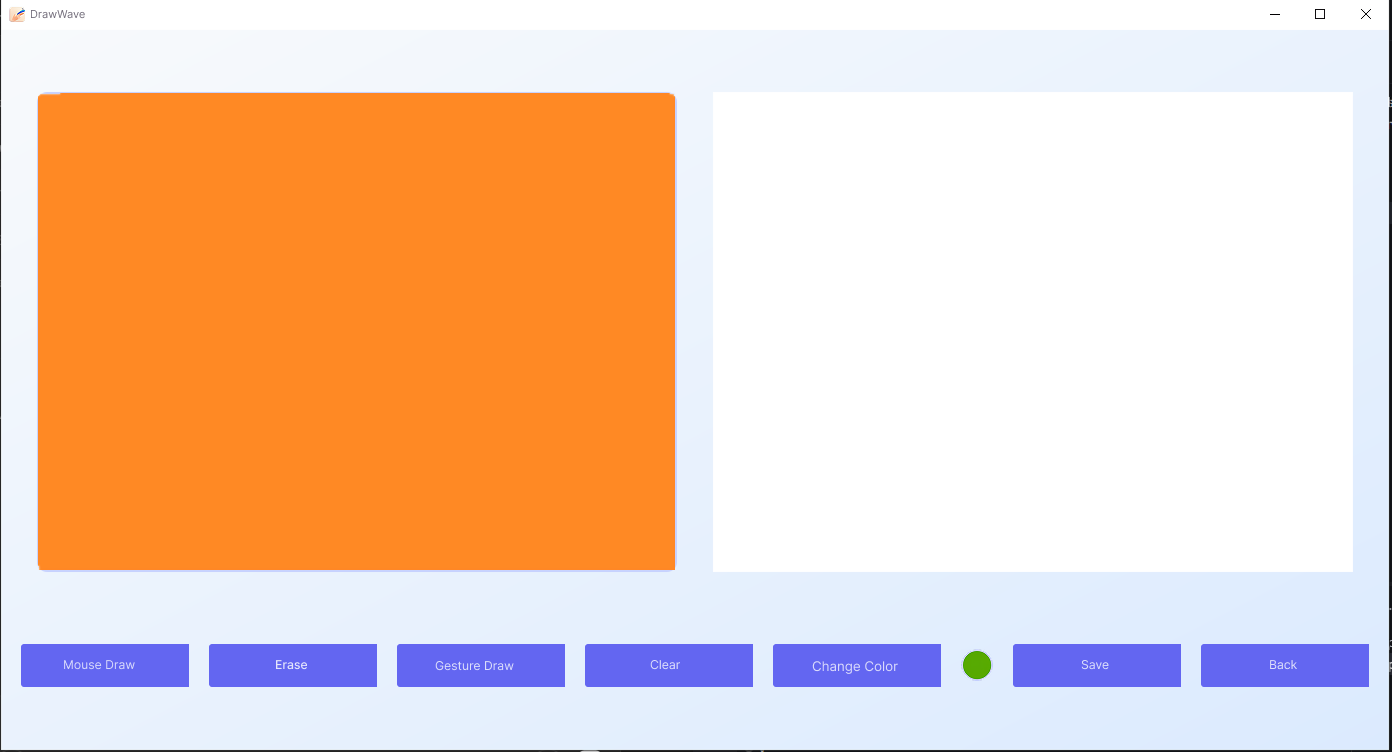
* **Desktop Application**
  + Real-time hand gesture detection
  + Drawing, erasing, undo/redo, save/load via SQLite
  + mouse drawing mode
  + User-friendly PyQt5-based user interface for drawing canvas and controls
* **Web Application**
  + Real-time multi-user collaborative canvas
  + WebSocket integration for gesture broadcast and canvas sync
  + Frame processing and gesture recognition via backend and python model
  + Gesture-to-command mapping
  + React-based front-end with intuitive gesture guide, drawing canvas, and tool controls

## User Interfaces

#### User Interfaces: Desktop Application



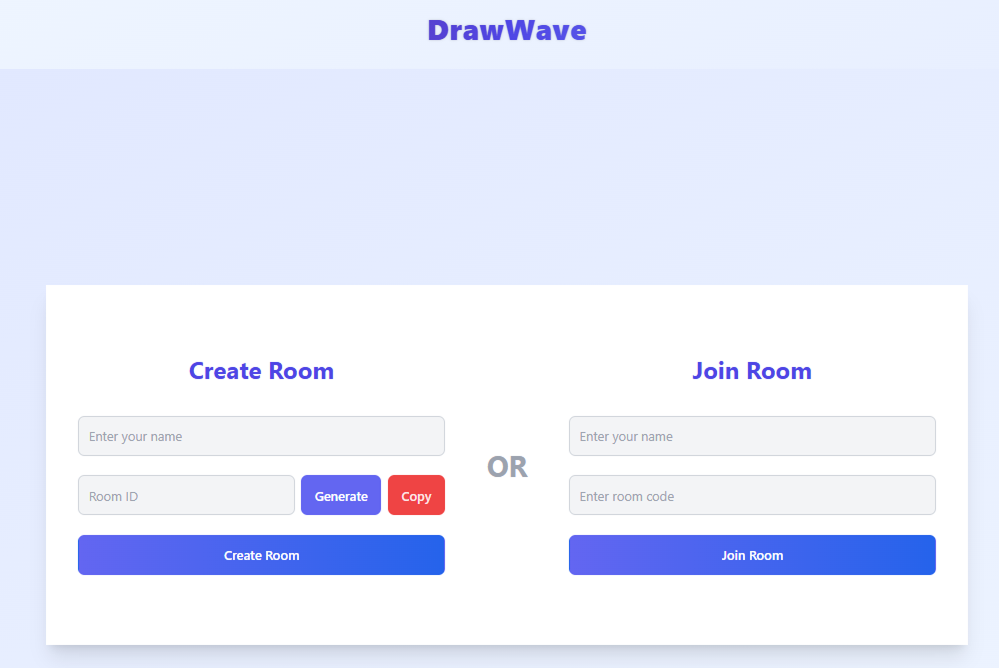
The start\_screen.py module is the default entry point of the DrawWave (AI Virtual Painter) application, that provides a very pleasant and user-friendly initial interface based on PyQt5. It provides the Start Screen class, a QWidget which initializes a fixed-size (640x480) window with a clean vertical layout and a custom-styled title, subtitle and "Start" button. The screen uses consistent spacing, font hierarchy and color themes for modern UI aesthetics. Functionally when the user clicks the "Start" button, the start\_button\_click(involves) method invokes the start screen. This closes the start screen and starts the main application window (since the QuickTime virtual plugin has been instantiated and displayed) through the implementation and display of VirtualPainterGUI.



The main interface interface is created as the class VirtualPainterGUI which handles video capture, gesture recognition, canvas rendering, and the handling of input mode. For this purpose it has a split layout: a live webcam feed processed (as with OpenCV and MediaPipe) for real-time landmark detection of the hands, and an CanvasWidget for drawing. Gesture recognition detects the state of the fingers using gesture information to determine whether they are drawn (index finger up), erased (index and middle fingers up), or idle and thus triggers the respective canvas update with corresponding normalized coordinates (for precision). The interface also provides a mouse input mode, where the camera feed is disabled and when it starts up the application direct drawing or erasing is done (via mouse events), which is done by the CanvasWidget.

A set of control buttons controls the actions of moving between modes, selecting colors, clearing the canvas, saving the drawing, undo/redo. Actions made during drawing are persisted to a local SQLite database (virtual\_painter. db) to support undo history and restore functionality. Undo is capable of being performed with Ctrl+Z as a global QShortcut. Module This project was developed using a modular architecture in which gesture logic is separated from UI rendering and database operations. This allows for responsive drawing in real time as well as extensibility of the program and efficient state management.

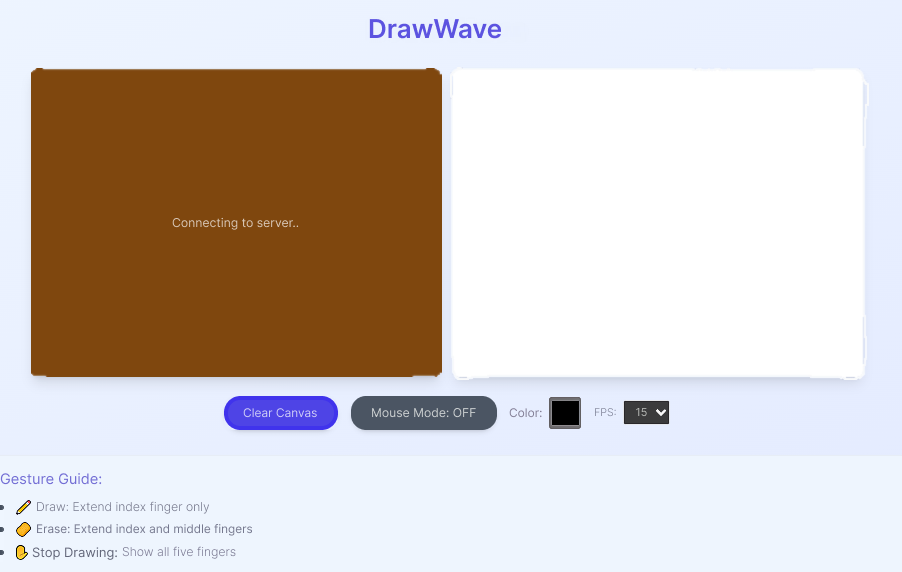
#### User Interfaces: Web Application



The room loading and initial room management of the Virtual Painter app is facilitated through client-server structured workflow that allows for easy onboarding of users. The App.tsx component, upon loading, conditionally renders what must be displayed, either the room entry interface or main drawing interface, from an associated state variable. RoomEntryWrapper.tsx renders the dual-panel interface through which the users can join an existing room or create room with responsiveness.

On the create side, the users input their name, generate an auto-matched 6-digit room code, and submit the form from CreateRoomForm.tsx. The submission of the form sends an issuance of the POST request to the backend API (/api/rooms) that talks with MongoDB for room uniqueness check as well as storage of room details.

JoinRoomForm.tsx offers equivalent functionality for joining with submission of the GET request in order to query for room presence before proceeding. When the app is successfully joined or created in the room, the onJoin callback appends the room code as well as the user's name directly to the app's state, flipping the interface over the VirtualPainter.tsx component. The state-based design offers dynamic room select-switching as well as collaborative drawing with the backend doing everything for data integrity as well as access control.



Virtual Painter implements a Bi-directional real-time drawing system that unifies frontend visualization with backend canvas state control through the drawing system. Users can access the drawing interface through a frontend component based on React-based VirtualPainter.tsx which utilizes HTML5 <canvas> to show user drawing interactions. Users can choose between drawing using a mouse or webcam video stream evaluation which happens in the backend system. WebSocket enables real-time transmission of drawing actions to synchronize all users who share the same room through connected websockets. Local rendering through a frontend component comes with the ability to customize features such as color picker tools and adjustable frame rates and drawing canvas clearing abilities.

The canvas.py backend module sustains an active canvas state by keeping records of brush strokes along with colors along with cursor positions and stored historical states for implementing undo and redo operations. Every drawn command processed by the system causes a server-side canvas update that sends real-time replication of the shared state to all connected clients. The drawing modes of drawing and erasing activate when MediaPipe detects specific finger positions while the user moves their hands in front of the webcam. The drawing application achieves scalable real-time collaboration through its combination of client-side rendering with backend state management which delivers responsive performance and minimal latency.

## Programming Languages and Technology Choices

* 1. **Web App Frontend — TypeScript**

TypeScript adds type safety to JavaScript to reduce bugs. By pairing this Javascript library with React, it allows updating a UI in real time. Moreover, it makes UI development easier by providing you with building block components that can be reused again and again.

* 1. **Web App Backend — Python + JavaScript (Node.js/Express)**

We used Python due to the strong AI and image-processing packages it has. Node.js and Express were used for handling room-based WebSocket connections due to the event-driven approach.

* 1. **Web App Model — Python + MediaPipe**

MediaPipe’s hand tracking implementation can provide robust results with only a few configuration parameters. Python is now frequently adopted for real-time gesture classification due to easy configuration with OpenCV.

* 1. **Database — SQLite (Desktop), MongoDB (Web)**

The desktop application will be using SQLite as it is simple software with zero configuration and light local storage. The principal database for the web application was MongoDB. MongoDB's NoSQL, documentation-oriented model helps developers exploit the great compatibility with the backends made by Node.js. It allows multiple customers to use the same data concurrently in a reliable and efficient manner.

* 1. **Desktop App Frontend — Python (PyQt5)**

You can design a fully-featured GUI with the help of PyQt5 which can interact with the image streams of OpenCV. The software is ideal for creating desktop applications with complex layouts and custom widgets.

* 1. **Desktop App Model — Python (OpenCV + MediaPipe)**

The detection, interpretation and mapping of hand gestures was done using OpenCV and MediaPipe in Python. The model is capable of running locally without a dependent server, thereby simplifying offline use.

# Literature Review

The DrawWave (AI Virtual Painter) project builds upon foundational research in the domains of hand gesture recognition, real-time image processing, and virtual canvas interaction. Several existing systems and academic efforts provide context for the development of gesture-based drawing tools. However, these systems often fall short in terms of environmental adaptability, user inclusiveness, scalability, and multi-user support—gaps this project specifically aims to address.

1. **Virtual Painting through Hand Gestures: A Machine Learning Approach**

**(R. Shanthi, S. A. Azeez, N. D. Kumar)**

This project demonstrated how motion in real-time was analyzed with machine learning algorithms and OpenCV in order to produce drawings. This brought with it improved recognition rates and usability of the technology. It was, though, not accessible to the disabled and not tested in new environments.

**Relevance:** mentions two major developments in virtual painter AI inclusive design problems and scalability, that show the accuracy of gesture recognition in recognizing them.

1. **Virtual Paint Application by Hand Gesture Recognition System**

**(P. Vidhate, R. Khadse, S. Rasal)**

Though the system employed OpenCV in realizing real-time gesture painting and was lauded for the minimal utilization of resources, it lacked advanced options such as color selection and erasure. Only static cases were considered.

**Relevance:** the principle of minimum viable product that DrawWave (AI Virtual Painter) is able to function on by real-time environmental adaptability and high-level instruction.

1. **AI Virtual Drawing Using OpenCV and MediaPipe**

**(S. Gokul, J. J. Franklin, S. Yogarajan)**

MediaPipe was used in combination with OpenCV to enhance gesture recognition and pick and draw color support. Multi-user input or collaboration was not supported by the technology.

**Relevance:** Validates the usage of OpenCV + Mediapipe and highlights the necessity of supporting multi-user configurations.

1. **Virtual Painting with OpenCV Using Python**

**(Y. Patil, M. Paun, D. Paun, K. Singh)**

provided a simple hand gesture-based hand-drawning interface. Although it produced demonstrations of what one could with OpenCV in virtual drawing, the system was not environment-independent and accessible.

**Relevance:** Sets the foundation of the baseline usability of OpenCV in favor of more user-friendly access and adaptive gesture recognition.

1. **Hand Gesture-Based Virtual Mouse Using OpenCV**

**(G. Jagnade, M. Ikar, N. Chaudhari)**

This research examined the use of hand gestures to control a virtual mouse. While the system was designed to operate in real-time, it lacked collaboration and user customization functionalities.  
**Relevance:** DrawWave (AI Virtual Painter) gesture control, as determined by the user, and support for multiple users enhance OpenCV interactive gesture usage usability further.

1. **Real-Time Signal Processing Using AI Integrated Framework**

**(G. Lavania, V. Arya, N. Sharma)**

Utilized OpenCV with AI in gesture control in sketching environments. It did showcase how AI could enhance real-time performance but did not address accessibility, environmental flexibility, and inclusiveness.

**Relevance:** Illustrates the promise of AI incorporation; DrawWave (AI Virtual Painter) completes with real-world testing, cloud synchronization, and user-accessibility enhancements.

### 5.1. Research Gaps Identified

1. **Scalability** – Most of the models were limited to static, controlled conditions with limited real-world usableness.
2. **Multi-User Collaboration** – Collaboration capabilities were lacking or constrained to one device or user.
3. **Accessibility** – Limited systems adapted users with disabilities or alternative input requirements.
4. **Environmental Adaptability** – Performance often reduced under changing lighting or background circumstances.
5. **Persistence & Sharing** – Many models lacked session memory or cloud-based sharing implements.

### 5.2. Conclusion of Review

Although hand gestures in interactive programs have been detected by previous research using OpenCV and MediaPipe, it also emphasizes that there is a need for more robust, scalable and inclusive solutions. The DrawWave (AI Virtual Painter) explicitly answers those needs with the addition of:

* Real-time collaboration via WebSockets
* Multi-environment testing and lighting adaptability
* Accessibility features such as voice commands and gesture customization
* Local and cloud-based persistence for drawing sessions

This project is therefore one significant step towards developing gesture-based drawing systems from one-user demonstrations towards professional-level, accessible, and cooperative systems that are capable of being used in various environments.

# Method of Approach

The DrawWave (AI Virtual Painter) project was implemented using a systematic framework taken from the Waterfall model. The systematic approach enabled accurate planning, serial progress, and greater sustainability in subsequent projects.

## 6.1. Functionality

#### System Advancement

* Creation of a multiple platform virtual sketching system based on hand gesture recognition technology.
* Developing a desktop application for standalone usage and in parallel, a web application that allows synchronous collaboration.
* The addition of hand tracking through webcam using OpenCV and MediaPipe enables gesture manipulation.
* Modular design for flexible feature integration and platform-specific customization.

#### Gesture-Based Drawing and Interaction

* The analysis of hand landmarks enables the recognition of gestures such as drawing, erasing, and choosing colors.
* Enabling digital sketching activities without the need to use hardware input devices like a stylus or mouse.
* The implementation of intuitive gesture mapping in virtual instruments greatly enhances usability for users with varied skill sets.

#### Multi-User Collaboration (Web App)

* Real-time synchronization of drawing activities by using WebSockets.
* Enabling simultaneous participation of multiple users in one common shared workspace during a joint session.
* Backend handling of gesture commands, canvas updates, and session management.

#### Drawing Session Management

* The desktop app provides preservation and recovery support for drawings by utilizing SQLite to support local data storage.
* The web application uses MongoDB to store session and user data in order to enable scalability for potential cloud-based integrations.
* Structured data flow storing, retrieving and handling user drawn drawings across sessions.

#### User Interfaces and Experiences

* Simple, recursive UI. Live webcam display, drawing canvas, gesture status indicator.
* Toggle color palette, eraser toggle and mode (gesture / mouse) to let users for better UX.
* Adaptable designs suitable for both desktop and web environments seek to enhance usability and accessibility.

#### Accessibility Features

* The desktop client also has an optional mouse mode to help people with physical gesture impairments.
* Prioritize universal design that has minimal hardware requirements, employing only the webcam.

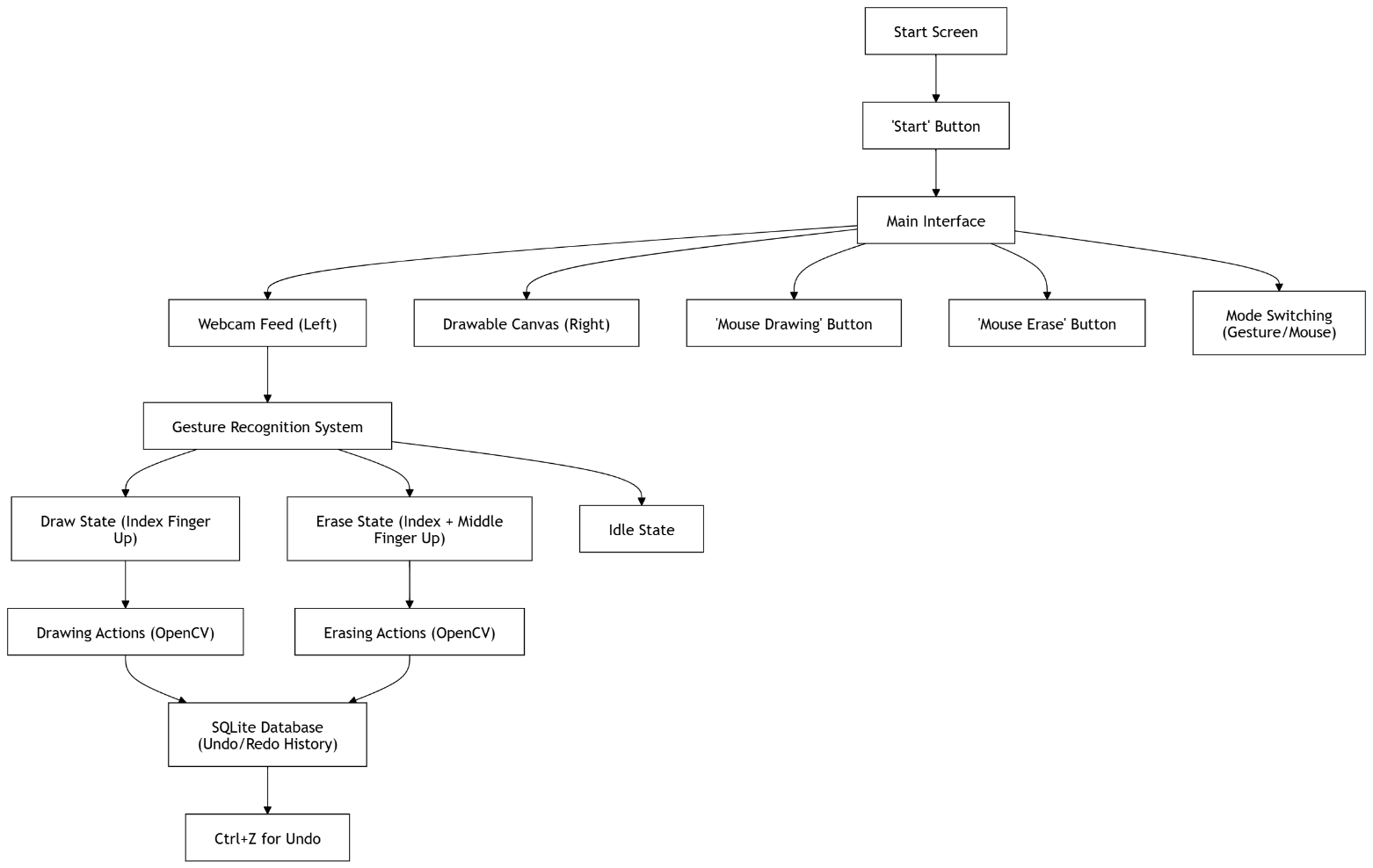
#### Feedback and Visualization

* Analysis and Evaluation Graphical display of current gesture commands and edits inside tools.
* Real-time gesture overlays and stroke tracking enable user interactions.
* Dynamic performance optimization (frames per second regulation) in the online version on multiple hardware configurations.

The first phase was focused on identifying and strengthening basic functionalities:

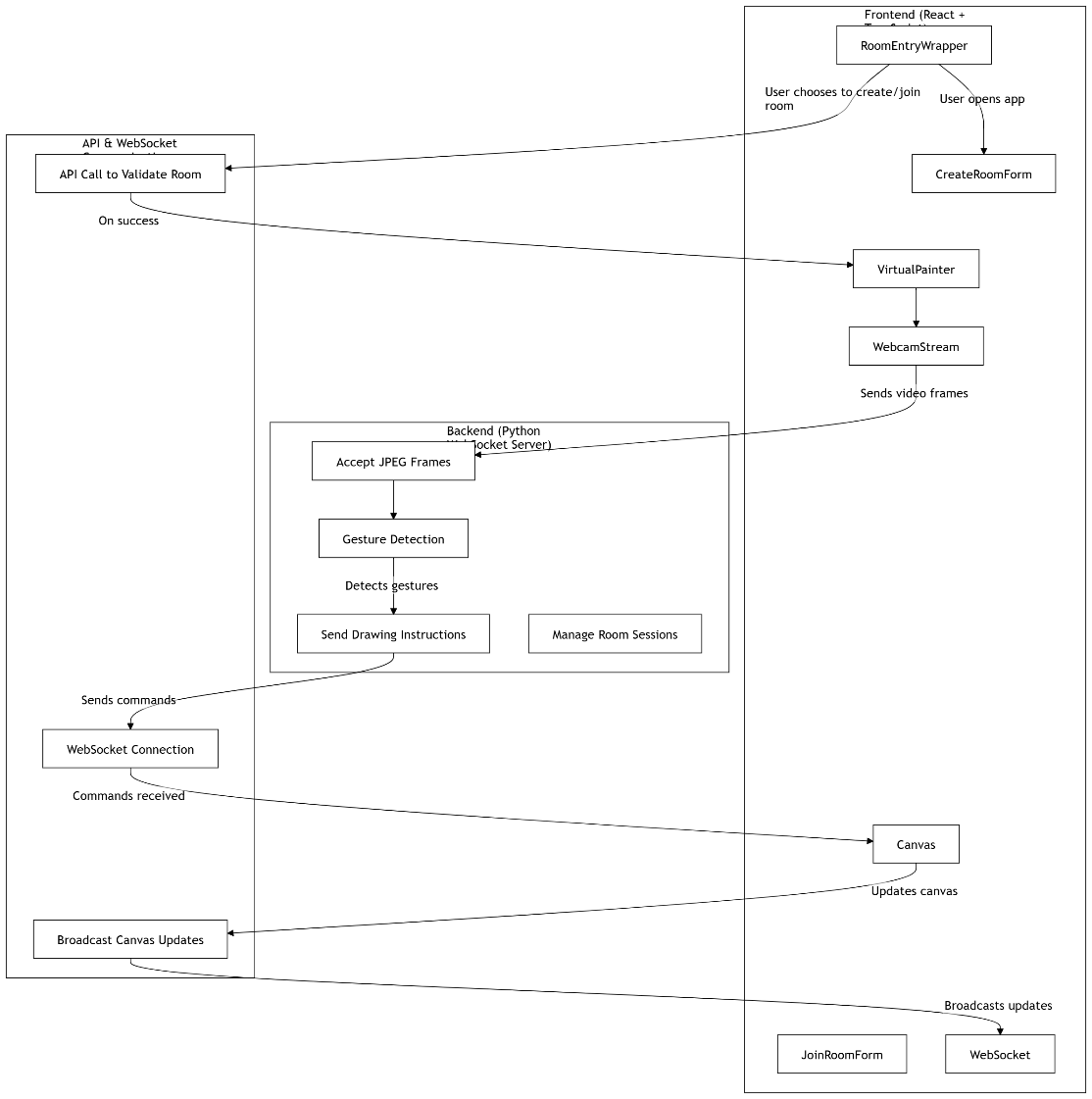
* Real-time gesture recognition using a webcam
* Canvas interaction using gestures (drawing, erasing, color selection)
* Mouse-based drawing as an alternative mode
* Multi-user synchronization and real-time collaboration (web version)
* User-friendly interfaces and intuitive controls

Features were chosen with care according to their utility, user expectations, and importance to the defined use cases (education, therapy, co-design, entertainment ).



**System Flow Overview:**

The flowchart shown here describes the core workflow of the DrawWave (AI Virtual Painter) desktop program. The workflow starts from a Start Screen prior to transitioning in to a dual-pane layout, encompassing both live webcam input as well as a canvas. The live webcam input is analyzed in real-time using MediaPipe for identifying the hand landmarks. The gesture recognition component handles the landmarks for determining the mode of the user as either in drawing, erasure, or idle mode. Depending on the recognized gesture, drawing or erasure operations are performed on the canvas using the help of OpenCV. Additionally, users have the option of switching between mouse-based input through specific drawing and erasure buttons. All the performed operations are logged methodically in an SQLite database for the purpose of enabling undo and redo features (e.g., using Ctrl+Z). The system design is carefully optimized for mode switch-over as well as for drawing handling.



**System Flow Overview:**

The Virtual Painter system allows instant drawing sessions using mouse controls and gesture-based system interaction. The system allows users to begin by either establishing new rooms or joining existing ones through its frontend. The authentication process through a backend API allows users to access the shared canvas environment which uses WebSockets for operation. Background video frames from the webcam get streamed to a backend server where gesture recognition occurs. All users in the same room receive drawing commands that derive from recognized gestures through a server transmission. The real-time updates to the canvas propagate simultaneously to every user in the session. The architectural design separates various features by assigning them separate roles for user interface input and real-time networking while processing gestures which leads to efficient multi-user collaboration experiences.

## 6.2. Data Collection

For training the system do not use traditional training datasets or machine learning classifiers, instead:

* MediaPipe’s pretrained hand landmark model is leveraged
* Landmarks get processed in real time and translated into gestures by custom logic
* Frame-by-frame input is used transiently and is not stored

No personal data is collected, it respects user privacy. There may be future iterations where consent based data collection is done to make the gesture models better.

## 6.3. Implementation

#### Desktop Application:

* developed in Python using PyQt5 for GUI and OpenCV + MediaPipe for gesture recognition
* Gesture logic is embedded in a custom module
* Drawn drawings are rendered on a NumPy array and saved with SQLite

#### Web Application:

* **Frontend**: React + TypeScript with HTML5 Canvas for drawing
* **Backend**: Python for gesture classification; Node.js for WebSocket management
* **Communication**: WebSocket protocol for real-time data exchange between users
* Frames are sent as base64 encoding, classified, and replies are broadcast to all clients in the session

## 6.4. Maintenance

The project is designed to be modular and extensible (so that it can support:

* Bug tracking and issue logging for stable post-release updates
* Modular components (gesture processing, UI rendering, communication) for isolate improvements
* Features roadmap, like voice control, cloud sync and gesture tailoring
* Long-term adaptability with open-source tools and minimal hardware dependency

At the future updates we will focus on scaling up the backend, adding platform support and improving the user experience across devices.

# Requirements

## 7.1. Functional Requirements

Artificial intelligence Virtual Painter system interprets user gestures for digital drawing and collaborative collaboration 0 below are the core functional requirements

* **Gesture Detection and Classification:** The system must detect hand landmarks using MediaPipe and classify them into commands (draw, erase, color change, idle).
* **Canvas Interaction:** users should be able to draw with gestures such as applying strokes, deleting sections and picking colour from a palette.
* **Real-Time Webcam Integration:** Both desktop and web applications should continuously capture frames from the webcam to ensure fluid hand tracking.
* **User Interface Interaction:** The UI should provide controls for toggle modes (gesture / mouse), color pick, undo / redo and save files.
* **Multi-User Collaboration:** Web version should enable multi-user interaction on the same canvas (by synchronized WebSocket communication).
* **Session Persistence:** Drawings on the desktop should store locally in SQLite, and preferably support memory persistence, with future possibilities for cloud storage.
* **Fallback Input Methods:** (To make it more accessible ) users should be able to switch to mouse drawing on the desktop version.

## 7.2. Non-Functional Requirements

Non-functional requirements describe the quality attributes of the system:

* **Performance:** Motion recognition and canvas updates should occur at a speed of less than 100 ms for running at real-time responsiveness.
* **Accuracy:** The gesture recognition module must have high accuracy in varying light and background conditions.
* **Scalability:** The backend must be able to support concurrent WebSocket sessions with limited resource contention.
* **Accessibility:** the accessibility needs should be addressed by gestures in adjustable modes and future voice command integration.
* **Security:** not storing the data without user consent. In the web app too all communication should be encrypted in HTTPS/WSS.
* **Portability:** the desktop app should be portable to Windows environments (with remarkably little setup) and the web app should support all modern browsers.
* **Reliability:** The system should be able to recover smoothly from an unanticipated system failure ( network failure, webcam failure, etc. ).

## 7.3. Hardware & Software Requirements

### 7.3.1. Hardware Requirements:

* **Input Device:** Standard USB Webcam (720p or higher recommended)
* **Processor:** The minimum two core processor needed for desktop app and multi core server for back end hosting
* **Memory:** At least 4 GB RAM for local use; scalable server memory based on active sessions
* **Display:** Minimum 1280x720 resolution for usable canvas space

### 7.3.2. Software Requirements:

#### Desktop Application

* Python 3.8+
* PyQt5 (GUI)
* OpenCV & MediaPipe (Gesture Recognition)
* SQLite3 (Persistence)
* PyInstaller (Packaging)

#### Web Application

* Frontend: React, TypeScript, Vite, Tailwind CSS
* Backend: Python , Node.js (Express.js)
* Communication: WebSockets
* Optional: Docker (Deployment), Redis (Future caching/persistence layer)

This covers the minimum requirements for making the system run smoothly at low cost with accessible technologies. The next version may have more intensive requirements regarding AI model training or integration into cloud storage.

# 8. End Project Report

## 8.1. Project Summary

DrawWave (AI Virtual Painter) was originally developed as a gesture-based virtual painting tool that allows the user to draw in real-time using standard webcams. It is built as a standalone desktop application and as a collaborative Web based platform that lets the user paint, erase and interact with a digital canvas via intuitive hand gestures. It combines a portfolio of technologies including OpenCV, MediaPipe, React and WebSockets to provide a low-cost, scalable alternative to traditional interactive web applications like smartboards.

There are a number of motivations for this project: the aim was to solve the problems of both cost (commercial hardware-based systems are non-universal) and usability (there are obstacles to accessibility of existing commercial products), as well as user experience (AI-based hand tracking and chat-based interaction in real time) and wider accessibility.

## 8.2. Achievements

* **Fully functional desktop app** using PyQt5, with gesture and mouse drawing, undo/redo, color selection, and save/load using SQLite.
* **Real-time collaborative web apps** built with React, Python, and WebSockets, enabling multi-user canvas interaction.
* **Use of MediaPipe in gesture recognition** guarantees consistent performance in multiple lighting conditions and assorted hand configurations.
* **a design composed of reusable components** (gesture logic, canvas control, communication layer).
* **A user-friendly and intuitive interface** that displays real-time webcam feedback and gesture pictures.
* **Secure communication** with HTTPS/WSS and preparing in advance to use cloud storage and user login in the future.

## 8.3. Objectives (Achieved vs. Planned)

|  |  |  |
| --- | --- | --- |
| **Objective Type** | **Planned (PID & Interim)** | **Achieved in Final** |
| Real-time drawing via gestures | Yes (OpenCV + Mediapipe) | Yes |
| Gesture to command mapping | Yes (Draw, Erase, Color) | Yes |
| Desktop app with persistence | Yes (SQLite) | Yes |
| Web app collaboration | Optional | Fully Implemented |
| Voice control support | Planned | Not Implemented |
| Accessibility via gesture/mouse toggle | Yes | Yes |
| Cloud save/load | Future feature | Not Implemented |
| Undo/Redo | Planned | Not Implemented (Desktop only) |

## 8.4. Changes to Project Scope

During the examination of the PID, Interim Report and Final Implementation documents the following issues were identified in relation to the project:

#### 8.4.1. Feature Additions:

* **Mouse Drawing Mode** added to desktop app for accessibility.
* **Undo/Redo and Save-to-Image** also implemented in the desktop version (not covered by original scope).
* **Room-based WebSocket architecture** inherently optimized for better multi-user control (as opposed to global session).
* **Frame processing optimizations** in the web app to reduce latency and fluid performance.

#### 8.4.2. Scope Removed or Deferred:

* **Voice Control** was originally proposed for accessibility, but because of time and complexity in integrating speech recognition, it was postponed.
* **Cloud Storage (e.g., AWS S3)** was designed but not implemented fully.
* **Gesture Customization UI** remains a future enhancement (not implemented).

#### 8.4.3. Design Changes:

* Desktop UI was moved from a very minimalist prototype to an easier to use layout with intuitive control over color and mode.
* Web UI added FPS tuning and gesture guide overlays that were not part of original plans.

These changes are reflection of iterative refinements based on technical feasibility, constraints on development and improved understanding of user requirements.

**5. Conclusion**

DrawWave (AI Virtual Painter) successfully achieved its major goal of providing a low-cost AI-powered drawing tool with standalone (desktop) and collaboration (web) capabilities. It successfully demonstrated the viability of using real-time gesture recognition to enable effortless digital interacting and collaborative canvas-centered sketching.

One of the strength of the project is that it's modular in design, uses open source libraries and can be programmed to handle whatever user environment you may have. Some features (cloud support and voice command capability) didn't make it out of version one, but the minimum framework is a good place to start.

Shortly put, the DrawWave (AI Virtual Painter) is not only a technological display, but moreover a scalable, portable and creative tool suitable for classroom use, therapeutic settings and co-creative design studios.

**Project Post-Mortem**

**1. Project Objectives**

The DrawWave (AI Virtual Painter) project established its primary goal as creating a gestural drawing interface which operated through both standalone computer mode and collaborative internet mode. Core functionalities comprised real-time hand detection, drawing capabilities, erasing functions, color selector and desktop session retention and web-based WebSocket-realized real-time drawing synchronization. The project design focused on accessibility and usability through its elimination of hardware limitations as well as its development of straightforward control mechanisms.

**2. Product Specification**

The final product consists of two major components:

* **Desktop Application** written in Python and PyQt5 with gesture and mouse drawing, undo/redo functionality and local saving/loading by SQLite.
* **Web Application** The system architecture involves a web framework that is based on React, backed by Python and Express.js, allowing users to draw using gestures and also supporting the recognition of gestures from webcam recordings.

They offer functionalities like dynamic gesture-mapping methods and responsive User Interface that changes in accordance with user activity.

**3. Development Process**

The project used the Waterfall methodology, which has been described as: Strategically and sequentially, with requirement gathering, design, implementation, and testing. Initial phases were intensive in research and planning, followed by sequential deployment and iteration of the UI. Documentation and modular design allowed the system to be maintained and extended in the future. Failing formal user testing, the system was functionally tested continuously throughout the process.

**4. Chosen Technologies**

* **OpenCV & MediaPipe (Python)** for real-time gesture recognition and landmark tracking.
* **PyQt5** for GUI development in desktop applications.
* **React + TypeScript** for modern frontend development.
* **Python + Express.js** for gesture classification and WebSocket control.
* **SQLite** for local persistence; Docker was planned for deployment.

Technologies were chosen for their compatibility, open-source nature, and strong community support, allowing cost-effective and scalable solutions.

**5. Own Performance**

The group developed the process by themselves and handled each component on their own. They strictly complied with the design principles and completed key tasks on time. Managing time better on those parts that required backend enhancement and availability would have been helpful. The group definitely excelled in technical work as well as in performing user interface and low-level processing tasks.

**6. Lessons Learned**

* Modular planning facilitates easier debugging, refactoring, and future scalability.
* Gesture based systems require early and diversified environment testing.
* Sub-dividing feature development into sprints would have been easier to balance parallel development jobs.
* System performance is determined in large part by frame processing efficiency; the input pipeline optimization is important.
* For collaborative tools (WebSocket events) exact state management is required so that race conditions or lag doesn't happen.

**7. Project Successes (Strengths & Accomplishments)**

* **Technical Robustness**: Successfully integrated OpenCV and MediaPipe for real-time gesture detection.
* **System Modularity**: Clean architectural separation allowed simultaneous development on desktop and web platforms
* **Collaborative Capabilities**: The WebSocket-based communication model successfully combined for time synchronization of multi-user drawing sessions.
* **User Experience Design**: Better interfaces were intuitive, adaptive and responsive in order to foster user engagement.
* **Scalability**: The system is completely scalable to other gestures, collaborative rooms and cloud storage.

**8. Limitations & Issues Encountered**

* **Incomplete Features**: Some features such as voice control and cloud storage along with undo/redo capabilities for web were not completely developed.
* **Environmental Limitations**: Some lighting conditions created bigger difficulties with gesture detection than developers initially predicted.
* **Resource Overlap**: Managing two platforms at once caused a split of available resources thus reducing progress occasionally.

The project achieved its main goals along with reliable technical implementation which provides opportunities for continued development. The deployment creates an operating base which supports gesture-based real-time interaction systems on different platforms.

**Future Implementations**

Several potential enhancements and extensions have been identified to expand the DrawWave (AI Virtual Painter) system's functionality and capabilities:

**1. Cloud Storage Integration**

Give users option to store and access their work using cloud storage services. This will offer persistent storage across sessions and cross-device accessible.

**2. Gesture Customization Interface**

Users should have the capability to develop their own gestures by utilizing a step-by-step configuration system. The system will customize itself to individual needs which benefits users with distinct hand movements for more accessible control.

**3. Voice Command Support**

The implementation of speech recognition APIs enables users to operate application modes along with selecting colors and giving commands by speaking natural language instructions. This application will provide special utility to those who experience limited movements within their bodies.

**4. Shape Detection and Smart Drawing**

A system that employs artificial intelligence enables it to detect manual drawings of shapes such as rectangles or circles to change them into smooth vector-based images automatically. The structured visuals would help designers and educators during their work.

**5. Collaborative Enhancements**

* The software provides real-time chat and annotation functions within its web application.
* The system allows users to record their drawing collaborations followed by session playback functions.

**6. AI Model Upgrades**

Transition from rule-based gesture classification approaches to neural network models (like TensorFlow Lite models) to enable more complex, adaptive, and extensible gesture recognition

**9. Plugin Support and API Access**

The system enables third-party connection through a public API along with a plugin interface for LMS platforms and design tools and multiplayer educational games.

These enhancements serve to upgrade the toolset to a production-grade level allowing its use in educational and professional creative fields.

**Conclusion**

The DrawWave (AI Virtual Painter) project successfully showcases the potential of gesture-based interaction in desktop and web applications by, using low-cost, easily accessible technologies like OpenCV, MediaPipe, React and WebSockets, closing the gap between complex hardware based drawing programs and intuitive camera-based digital interaction systems.

The main goal of this project through the entire development lifecycle was to develop a responsive, real-time drawing system using gestures as well as build a multi-user collaborative web canvas with tools for real-time collaboration and coloring. Also in the desktop application, real-time draw features such as color selection, undo/redo and erase were implemented along with the offline performance and local session saving of content.  
As well as technical achievements, the system follows a set of principles of accessibility, scalability, and inclusivity—including user-friendly controls, low hardware requirements, and flexible functionality that works for different needs. While a number of advanced features—such as voice control and cloud syncing—are still in development, the foundation has been laid solidly for development.

The structured development process (Waterfall model) and modular architecture, real-time system design brought efficiency in integration and flexible implementation of the solution and the relevant challenges like gesture accuracy in an environment with changing environment, handling multiple users interactions and interface design were successfully solved.

In Conclusion, DrawWave (AI Virtual Painter) is not just a working prototype: it is a creative, inclusive and scalable platform for gesture based expressive art. With future development efforts centered on better accessibility, cloud integration and smart shape recognition, DrawWave will offer real value both to education and design.

**References**

**Bibliography**

(Vidhate, Khadse and Rasal, no date; Lavania *et al.*, 2022; Jagnade *et al.*, 2023; Gokul *et al.*, 2024; Patil *et al.*, 2024; Shanthi, Azeez and Kumar, 2024)

Gokul, S. *et al.* (2024) *INTERNATIONAL JOURNAL OF MULTIDISCIPLINARY RESEARCH AI Virtual Drawing Using OpenCV and Mediapipe*, *SCIENCE, ENGINEERING AND TECHNOLOGY*. Available at: www.ijmrset.com.

Jagnade, G. *et al.* (2023) ‘Hand Gesture-based Virtual Mouse using Open CV’, in *IDCIoT 2023 - International Conference on Intelligent Data Communication Technologies and Internet of Things, Proceedings*. Institute of Electrical and Electronics Engineers Inc., pp. 820–825. Available at: https://doi.org/10.1109/IDCIoT56793.2023.10053488.

Lavania, G. *et al.* (2022) ‘Real-Time Signal Processing using AI Integrated Framework for Color and Drawing in Gesture Recognition’, in *Proceedings of 5th International Conference on Contemporary Computing and Informatics, IC3I 2022*. Institute of Electrical and Electronics Engineers Inc., pp. 473–478. Available at: https://doi.org/10.1109/IC3I56241.2022.10072448.

Patil, Y. *et al.* (2024) ‘Virtual Painting with Opencv Using Python’. Available at: https://doi.org/10.32628/IJSRST205831.

Shanthi, R., Azeez, S.A. and Kumar, N.D. (2024) ‘Virtual Painting through Hand Gestures: A Machine Learning Approach’, *Journal of Ubiquitous Computing and Communication Technologies*, 6(1), pp. 39–49. Available at: https://doi.org/10.36548/jucct.2024.1.004.

Vidhate, P., Khadse, R. and Rasal, S. (no date) *VIRTUAL PAINT APPLICATION BY HAND GESTURE RECOGNITION SYSTEM*, *International Journal of Technical Research and Applications*. Available at: www.ijtra.com.

**Appendices**

**User Guide: Desktop Application**

**Overview**

DrawWave (AI Virtual Painter) is a desktop application that enables users to draw and erase on a digital canvas using either hand gestures detected via a webcam or traditional mouse input. The application supports gesture-based drawing and erasing, mouse-based interactions, undo/redo functionality, and persistent storage of drawing actions using a local SQLite database.

**System Requirements**

* Operating System: Windows 10 or later
* Python Environment: Python 3.8+ (for development)
* Hardware: Webcam (internal or external) & Mouse or touchpad

**Installation & Setup**

1. Install all required Python packages (if using source):

**pip install -r requirements.txt**

1. Run the application:

**python main.py**

1. Alternatively, run the bundled Painter.exe.

**Application Startup**

* Upon launching the application, the user is presented with a **Start Screen**.
* Clicking **Start** initializes the webcam and loads the main drawing interface.

**Interface Layout**

The main interface is split into two main panes:

* **Left Panel**: Live webcam feed with hand tracking overlay.
* **Right Panel**: Drawing canvas where input is rendered.

A control bar is provided below the panels with buttons for:

* Mouse Drawing
* Mouse Erase
* Gesture Drawing
* Save
* Clear Canvas
* Undo (also accessible via Ctrl + Z)
* Redo (also accessible via Ctrl + Shift + Z)
* Color Picker
* Back to Start

**Input Modes**

**Gesture Drawing**

* Default mode on startup.
* Raise **index finger only** to draw.
* Raise **index + middle fingers** to erase.
* Raise **all fingers** to reset gesture input.

**Mouse Drawing**

* Click **Mouse Drawing** to enter mouse drawing mode.
* Left-click and drag to draw lines.

**Mouse Erasing**

* Click **Mouse Erase** to enter mouse erasing mode.
* Left-click and drag to erase using a white brush.

***Note: Gesture recognition is paused while in mouse mode.***

**Saving Work**

* Click **Save** to export the current canvas as an image.
* The image files saved can be stored locally.

**Database Persistence**

The application uses virtual\_painter.db (SQLite) to:

* Log drawing and erasing actions.
* Manage undo/redo history.
* Store saved canvas states.
* Persist user settings like selected color and brush size.

**Error Handling & Fallbacks**

* If the webcam is unavailable or fails to initialize, the application will notify the user and disable gesture features.
* The canvas and mouse features remain usable independently.

**Best Practices**

* Ensure consistent lighting for accurate gesture detection.
* Avoid occluding the camera during gesture mode.
* Use mouse mode for precision edits.

**User Guide: Web Application**

**1. Introduction**

Virtual Painter is a collaborative web-based drawing application that allows multiple users to draw on a shared canvas in real time. It supports both traditional mouse-based drawing and gesture-based input through webcam video streaming, which is processed by a backend server for gesture recognition. Users can create or join private rooms to collaborate securely.

**2. Accessing the Application**

The application is browser-based and accessible via a modern web browser (preferably Chrome or Firefox). No installation is required. To begin using the app, navigate to the project’s deployment URL.

**3. Creating or Joining a Room**

**3.1 Create Room**

1. Launch the application.
2. On the left panel, under **Create Room**, enter your name.
3. Click **Generate** to create a unique Room ID.
4. (Optional) Click **Copy** to copy the Room ID to clipboard.
5. Click **Create Room** to create the room and enter the drawing environment.

**3.2 Join Room**

1. On the right panel under **Join Room**, enter your name.
2. Enter the Room ID provided by another participant.
3. Click **Join Room** to enter the drawing environment.

***Note: If the Room ID is invalid or the room does not exist, an error message will be displayed.***

**4. Drawing Interface**

Once in the drawing interface:

**4.1 Canvas Area**

* The canvas is the main area for collaborative drawing.
* All drawing events are synchronized across users in the same room.

**4.2 Webcam Stream**

* Your webcam stream is activated in the background.
* Video frames are sent to the backend for real-time gesture analysis.
* The application uses detected hand gestures to control drawing actions (e.g., draw, erase, stop).

**4.3 Controls**

* **Clear Canvas:** Erases the current canvas for all users.
* **Mouse Mode Toggle:** Enables or disables drawing with the mouse.
* **Color Picker:** Allows selection of drawing color.
* **FPS Selector:** Adjusts the frame rate (in frames per second) for webcam video streaming.

**5. Gesture Controls**

* **Draw:** Extend the index finger.
* **Erase:** Extend both the index and middle fingers.
* **Stop Drawing:** Extend all five fingers (open palm).

Gestures are interpreted by the backend and translated into drawing commands sent back to all connected clients in the room.