Department of Computer Science
Faculty of Technology, University of Delhi

Project Report

Vision-Based Gesture Control System for Hands-Free Human-Computer Interaction

Course Name: Skill Enhancement Course
Semester: V Semester

Team Members:
Taufiq Ansari (23293916007)
Lakshay Chaudhary (23293916034)
Ibrahim Ibn Tanweer Almadani (23293916044)
Krishna Yadav (23293916045)
Anubhav Pathania (23293916002)

Supervisor:
Dr. Sangeeta Yadav
Department of Computer Science
Faculty Guide

Date of Submission: November 24, 2025

CERTIFICATE

This is to certify that the project report entitled

"Vision-Based Gesture Control System for Hands-Free Human-Computer Interaction"

submitted by

Taufiq Ansari (23293916007)
Lakshay Chaudhary (23293916034)
Ibrahim Ibn Tanweer Almadani (23293916044)
Krishna Yadav (23293916045)
Anubhav Pathania (23293916002)

in partial fulfillment of the requirements for the

Skill Enhancement Course, Semester V
Department of Computer Science,
Faculty of Technology, University of Delhi

is a bona fide record of the work carried out under my supervision.

Supervisor:
Dr. Sangeeta Yadav
Department of Computer Science
Faculty of Technology, University of Delhi

Declaration

We hereby declare that the project titled *“Vision-Based Gesture Control System for Hands-Free Human-Computer Interaction”* is an original work carried out by us as part of the Skill Enhancement Course (Semester V), Department of Computer Science, Faculty of Technology, University of Delhi.

We affirm that this report has not been submitted to any other institution or university for any academic credit. All sources of external information have been properly acknowledged and referenced.

Team Members:
Taufiq Ansari (23293916007)
Lakshay Chaudhary (23293916034)
Ibrahim Ibn Tanweer Almadani (23293916044)
Krishna Yadav (23293916045)
Anubhav Pathania (23293916002)

Contents

Vision-Based Gesture Control System for Hands-Free Human-Computer Interaction

Abstract—**This paper presents a comprehensive touchless human-computer interaction framework comprising two complementary systems: a Virtual Air Keyboard for text input and a Gesture-Based System Controller (AI Mouse) for cursor manipulation and system actions. The Virtual Air Keyboard utilizes MediaPipe for hand tracking, OpenCV for image processing, and a custom Convolutional Neural Network (CNN) trained on the EMNIST ByClass dataset to achieve real-time handwritten character recognition from air-drawn gestures. Users draw characters using their index finger, with control managed by specific hand poses (e.g., "Draw," "Commit," "Backspace"). The Gesture-Based System Controller employs a robust, mode-based gesture system to recognize predefined hand poses. This maps gestures to a rich set of actions including cursor control, discrete clicks, continuous system scroll, and application zoom, while a thumb-closed anchor enables stable cursor motion. Both systems operate in real-time on commodity webcams, demonstrating the practical viability of touchless interfaces for accessibility, hygienic computing, and hands-free operation.**

Index Terms—**Hand gesture recognition, computer vision, deep learning, CNN, MediaPipe, touchless interface, EMNIST, character recognition, human-computer interaction, gesture control, virtual mouse, rule-based classification, PyAutoGUI, air writing**

I. Introduction

Touchless interfaces have gained momentum due to needs in hygiene, accessibility, and immersive computing. Traditional peripherals (keyboard, mouse) can be limiting where hands-free operation is desirable, e.g., clinical settings, public kiosks, or assistive technologies. Air writing enables character input by tracking finger trajectories in free space, differing from static gesture recognition which focuses on discrete hand poses. This work combines both paradigms: air-drawn text entry via a CNN classifier and rule-based gesture control for system actions, powered by real-time hand landmark tracking with MediaPipe [1].

A. Motivation

Demand for contactless input rose during COVID-19. Beyond hygiene, gesture interfaces offer intuitive control for AR/VR, smart devices, and accessibility. We target two complementary use cases:

1) **Text Input**: Natural air-based character entry without physical keyboards.
2) **System Control**: Hand-pose-based commands for cursor and desktop automation.

B. Research Challenges

Challenges include: (i) dataset mismatch for air writing styles; (ii) real-time constraints on CPUs; (iii) generalization across users and environments; (iv) variability in writing speed, size, and position; (v) trajectory noise and jitter; and (vi) balancing accuracy (deep models) and latency/complexity (rule-based).

C. Contributions

- An end-to-end air keyboard using CNN-based recognition of segmented air-drawn characters.
- A lightweight AI mouse with rule-based gesture recognition and an anchored cursor motion model.
- Integration of MediaPipe landmarks with OpenCV preprocessing and PyTorch inference.
- A gesture-based control system for drawing, committing, and correcting text (backspace, clear all).
- A clear experimental plan for accuracy, latency, and ablation studies.

II. Related Work

A. Hand Tracking and Gesture Interfaces

Early solutions relied on instrumented gloves [2]. Depth sensors (e.g., Kinect, RealSense, Leap Motion) provide 3D tracking but require dedicated hardware. RGB-only approaches advanced with deep learning; MediaPipe Hands [1] offers efficient 21-landmark detection on commodity devices, enabling robust real-time gesture input.

B. Handwritten Character Recognition

EMNIST [3] extends MNIST to letters and digits and is widely used for benchmarking character recognition. CNNs are standard for handwritten character classification; aligning air-drawn trajectories to this domain requires appropriate preprocessing (normalization, orientation correction, scaling) to bridge the distribution gap between free-space strokes and pen-and-paper images.

III. Literature Review

A. Overview and Context of Touchless HCI

Recent advances in Human-Computer Interaction (HCI) have accelerated the shift toward touchless interfaces driven by demands for hygiene, accessibility, and intuitive interaction in immersive environments such as Virtual and Augmented Reality (VR/AR). Traditional peripherals (keyboard, mouse) become limiting in clinical settings, public kiosks, industrial environments, and assistive technology contexts.

To address these challenges, the presented work integrates two complementary systems—a Virtual Air Keyboard for high-accuracy text input and a Gesture-Based System Controller (AI Mouse) for real-time desktop control. Modern research consistently demonstrates that a **software-only stack** using commodity webcams provides practical, accessible, and scalable solutions. This stack is powered by Python, OpenCV for image processing, and MediaPipe Hands for efficient real-time detection of the 21 landmark coordinates needed for gesture recognition and air writing.

B. Foundation: Hand Tracking and Gesture Interfaces

Gesture-based HCI has evolved from early approaches using instrumented gloves, which offered precise tracking but were impractical for everyday use. Dedicated depth-sensing hardware such as Kinect, Intel RealSense, and Leap Motion improved usability but still required specialized equipment.

The recent shift toward **RGB-only deep learning approaches**—particularly MediaPipe Hands—has enabled robust 3D hand landmark detection using standard webcams. MediaPipe consistently demonstrates superior accuracy, speed, and generalization compared to earlier hardware-based trackers. Its 21 3D landmarks form the geometric backbone for cursor control, air-writing trajectory extraction, and static hand pose classification.

C. Gesture-Based System Control (AI Mouse)

Vision-based cursor control systems map real-time hand motion and distinct gestures to OS-level commands using interaction libraries such as PyAutoGUI and Pynput. A major usability challenge documented in literature is the **"Midas Touch" problem**, where unintentional movements trigger commands. Modern systems overcome this through structured, mode-based interaction.

1) Mode-Based Interaction: The AI Mouse follows a **mutually exclusive, mode-based state machine**, improving reliability and eliminating gesture conflicts. Each mode is activated through a unique, recognizable hand pose:

- **Scroll Mode:** Triggered by forming the "OK" sign (thumb tip touching index tip). Subsequent extension or curling of the remaining fingers controls scroll direction.
- **Zoom Mode:** Activated using a "Peace" sign (index and middle fingers extended), with vertical hand motion mapped to zoom in/out commands.
- **Screenshot Mode:** Mapped to the "Hang Loose" sign (thumb and pinky extended) to ensure non-overlapping gesture activation.

This structured design, supported by HCI literature, improves system robustness and scalability while simplifying the user's mental model.

2) Cursor Motion and Discrete Clicks: Cursor motion typically tracks the fingertip position of the index finger (Landmark 8). However, literature highlights the challenge of cursor instability during re-engagement. The presented system applies an **anchor-offset cursor motion model**, anchoring the position when the thumb (Landmark 4) closes toward the index finger's base knuckle (Landmark 5), thus preventing cursor jumps.

Discrete actions rely on lightweight, rule-based detection of finger flexion:

- Index flexion $\rightarrow$ Left Click
- Middle finger flexion $\rightarrow$ Right Click
- Dual flexion $\rightarrow$ Double Click

These actions are intentionally designed for low latency and high responsiveness, as supported by findings in virtual pointing research.

D. Vision-Based Text Entry

Text entry in mid-air remains a challenging domain with two dominant paradigms: Mid-Air QWERTY keyboards and Trajectory-Based Air Writing.

1) Mid-Air QWERTY vs. Trajectory-Based Approaches: Research consistently highlights a trade-off:

- **Mid-Air QWERTY Keyboards** offer conversational speeds of 20–30 WPM through probabilistic decoders but result in higher user fatigue.
- **Trajectory-Based Air Writing** offers superior per-character accuracy (often above 92%) at the cost of reduced speed due to segmentation overhead.

The Virtual Air Keyboard in this work adopts the air-writing paradigm, aligning with applications that require precision over speed, such as short text entry, commands, or authentication tasks.

2) CNN-Based Air-Writing Pipeline: The implemented air writing workflow incorporates several stages that align with current literature on trajectory recognition:

1) **Drawing Control:** Activated based on proximity between the thumb tip and the index MCP (Landmark 5). The adaptive threshold

$$\max(\alpha \cdot W, \tau)$$

ensures robust performance across varying resolutions.
2) **Segmentation and Tracking:** Once drawing pauses, the canvas is segmented using Otsu thresholding, morphological closing, contour extraction, and intelligent bounding-box merging. Inter-frame association via IoU matching supports dynamic letter replacement—an important feature for naturalistic use.
3) **CNN Recognition:** Cropped characters are padded, resized to 28×28, rotated 90° clockwise, and horizontally mirrored to match EMNIST orientation. A custom CNN architecture (two convolutional layers, two fully connected layers) trained on EMNIST letters performs final classification.

This approach matches the best practices in air-writing literature for bridging the domain gap between in-air strokes and handwritten datasets.

E. Synthesis and Design Challenges

The literature highlights several challenges that directly inform the system design:

- **Ergonomics:** Mid-air input is associated with higher perceived exertion. This suggests long-term usage requires low-fatigue design or multimodal assistance.

- **Dataset Mismatch:** A natural discrepancy exists between freehand air-writing trajectories and structured datasets like EMNIST. Careful preprocessing (rotation, padding, normalization) is essential.
- **Latency vs. Accuracy:** Deep learning-based recognition provides accuracy but increases latency; therefore cursor control relies on rule-based methods for real-time responsiveness.
- **Environmental Sensitivity:** Lighting variability and occlusions impact vision-based HCI systems, motivating future work in domain adaptation and sensor fusion.

F. Conclusion of Literature Review

Overall, prior research supports the integrated approach of combining **deep learning for high-precision air writing** with **mode-based rule-driven gesture control**. MediaPipe Hands provides a reliable backbone, enabling real-time performance using commodity cameras. The reviewed literature validates the architectural decisions of the presented system and outlines future directions such as multimodal fusion, temporal models, and improved robustness under environmental variability.

IV. System Overview

Our system consists of two applications sharing a common hand-tracking backbone:

- **Air Mouse** (`air_mouse.py` + `util.py`): A mode-based controller for full system interaction. It features anchor-offset cursor motion, discrete clicks, and dedicated modes for system scroll (via "OK" sign) and application zoom (via "Peace" sign).
- **Air Keyboard** (`air_keyboard_screenshots.py`): Index finger draws on a virtual canvas when a "Draw" gesture is made; letters are recognized via a PyTorch CNN (trained on EMNIST ByClass) when a "Commit" gesture is made.

Both use OpenCV for video I/O and visualization; PyAutoGUI and Pynput provide OS-level mouse and keyboard actions.

V. Methods

A. Hand Tracking Backbone

We employ MediaPipe Hands (`mp.solutions.hands`) with a single-hand configuration and confidence thresholds suited for real-time desktop use. The pipeline provides 21 landmarks per hand in normalized image coordinates which we project to pixel or screen coordinates depending on the task.

B. Air Mouse: Mode-Based Gestures and Motion

1) Mathematical Formulation for Distance and Angle Computation: The gesture classification module of the AI Mouse relies on two fundamental geometric primitives derived from MediaPipe's 21 hand landmarks: (i) point–to–point distances and (ii) joint angles formed by consecutive finger segments. These quantities determine whether fingers are extended, curled, or forming specific configurations such as the OK sign or the peace sign.

a) Euclidean Distance.: Given two normalized landmark coordinates $\mathbf{p}1 = (x1, y_1)$ and $\mathbf{p}2 = (x2, y_2)$, the Euclidean distance is computed as

$$L = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}.$$

Since MediaPipe outputs coordinates in $[0, 1]$, the raw distance is mapped to a fixed range $[0, 1000]$ via linear interpolation:

$$L_{\text{scaled}} = \text{interp}(L;\ 0\to 0,\ 1\to 1000).$$

This normalization enables robust threshold-based gesture decisions, such as:

- **Thumb–Index Knuckle Distance** for activating Move Mode.
- **Thumb–Index Tip Distance** for OK-sign detection.
- **Thumb Open/Closed State** for disambiguating gestures.

b) Joint Angle Computation.: To determine whether a finger is extended or curled, we compute the angle at joint $\mathbf{b}$ formed by points $\mathbf{a}$ and $\mathbf{c}$. The vectors are:

$$\mathbf{u} = \mathbf{a} - \mathbf{b}, \qquad \mathbf{v} = \mathbf{c} - \mathbf{b}.$$

The signed angles of these vectors are

$$\theta_u = \text{atan2}(u_y, u_x), \qquad \theta_v = \text{atan2}(v_y, v_x),$$

and the resulting joint angle is

$$\theta = |\theta_v - \theta_u| \cdot \left(\frac{180}{\pi}\right).$$

This corresponds directly to the implementation:

$$\text{angle} = |\text{atan2}(c_y - b_y,\ c_x - b_x) - \text{atan2}(a_y - b_y,\ a_x - b_x)|\,.$$

Empirical thresholds classify each finger as

$$\theta > 90^\circ \Rightarrow \text{extended}, \qquad \theta < 50^\circ \Rightarrow \text{curled}.$$

These angle features enable accurate recognition of the *peace sign*, *left/right click* gestures, and the *hang-loose* screenshot gesture.

c) Integration into Gesture State Machine.: The computed distances and angles serve as inputs to a priority-based finite-state machine that selects exactly one active mode among {Move, Scroll, Zoom, Idle/Click}. This mathematical foundation ensures robustness to scale differences, lighting variation, and viewpoint changes.

d) Cursor Motion with Anchor Offset.: The core motion model employs a dynamic anchor offset to prevent cursor jumps. When the thumb (landmark 4) closes toward the index-finger base knuckle (landmark 5), the system records an anchor offset between the finger coordinates and the current cursor location. Subsequent finger movements drive the cursor relative to this offset, yielding smooth and stable motion.

e) Mode-Based Gesture System.: The AI Mouse uses a mutually exclusive, mode-based state machine. Each gesture corresponds to exactly one macro-action:

- **Move Mode:** Activated when the thumb pinches the index knuckle. This mode has highest priority and enables smooth cursor control.
- **Scroll Mode:** Triggered by the **OK sign** (thumb tip touching index tip). Extending the other three fingers scrolls **up**, while curling them scrolls **down**.
- **Zoom Mode:** Activated by the **Peace sign**. Upward hand motion performs "Zoom In" (`Ctrl +`), and downward motion performs "Zoom Out" (`Ctrl -`).
- **Idle/Click Mode:** When no mode is active, discrete finger flexions trigger:
  - Left Click (index flex)
  - Right Click (middle flex)
  - Double Click (index + middle flex)
- **Screenshot:** The **Hang Loose** gesture (thumb + pinky extended) triggers a screenshot action.

Actions are executed using `pyautogui` and `pynput`. The modal design eliminates gesture conflicts and significantly improves practical usability.

C. Air Keyboard: Gesture-Based Drawing and Recognition

Drawing Control. We render strokes onto a single-channel canvas (`canvas`) when the user performs a specific **"Draw"** gesture: bringing the thumb tip (landmark 4) close to the index finger's base (landmark 5). This allows the user to draw strokes with their index finger (landmark 8). When the gesture is released, drawing stops, and the system awaits an action command.

Action Control with State-Lock. To prevent repeated or accidental triggers, a state-lock (`action_locked`) is employed. An action can only be triggered once per gesture. The lock is only reset when the user performs the "Draw" gesture again. The following gestures are mapped to specific system actions:

- **Commit Letter ("OK" sign):** Pinching the thumb tip [4] to the index tip [8] triggers `process_and_commit_letter`. This processes the current drawing on the canvas, predicts the character, and types it.
- **Commit + Space ("Peace" sign):** Index and middle fingers are extended. This first calls `process_and_commit_letter` and then types a space, ideal for ending words.
- **Backspace ("Open Hand"):** All five fingers are extended. This triggers a backspace key press, removing the last character from `text_output`.
- **Clear All ("Fist"):** All four fingers are curled down. This clears the entire `text_output` and the virtual canvas.

CNN Recognition. When a commit action is triggered, the `canvas` is processed as described in Section IV-D. The resulting $1\times1\times28\times28$ tensor is fed into our CNN. The CNN, trained on the **62-class EMNIST ByClass dataset** (digits, uppercase letters, and lowercase letters), predicts the character. If the prediction confidence exceeds 60%, the character is appended to the output string and typed.

D. Canvas Preprocessing for Character Recognition

To convert the free-form air-drawn strokes into an input compatible with the EMNIST-trained CNN, we design a dedicated preprocessing pipeline. The virtual canvas is a single-channel binary image updated whenever the draw gesture is active. Once the user performs a commit gesture (OK or Peace), the following steps are applied:

1) Thresholding and Noise Suppression: The canvas is smoothed using a 3×3 Gaussian filter and binarized using Otsu thresholding:

$$T = \arg\max_{\tau} \sigma_B^2(\tau),$$

yielding a clean binary silhouette of the drawn character. Small contour fragments (area < 100 px) are discarded.

2) Unified Bounding Box Extraction: Since characters such as "A", "K", or "H" may contain multiple disconnected strokes, all valid contours are merged to produce a single bounding box:

$$B = \bigcup_{i=1}^{N} B_i,$$

where B_i is the bounding box of contour i. Bounding boxes with total merged area below a minimal threshold (500 px) are ignored to avoid accidental triggers.

3) Square Padding and Centering: The cropped region is padded into a square canvas of size $\max(h, w)$ to preserve aspect ratio. Let h and w denote the height and width of the cropped letter. A square canvas $S \in \mathbb{R}^{m\times m}$, where $m = \max(h, w)$, is created and the crop is centered:

$$S[y_0 : y_0 + h,\ x_0 : x_0 + w] = \text{crop}.$$

4) Resizing to 28×28: The padded square is resized using area interpolation to match the EMNIST input resolution:

$$I_{28} = \text{Resize}(S,\ 28 \times 28).$$

5) Orientation Alignment (Rotate + Flip): Air-written trajectories have a different canonical orientation than EMNIST handwritten characters. To align distributions, we apply:

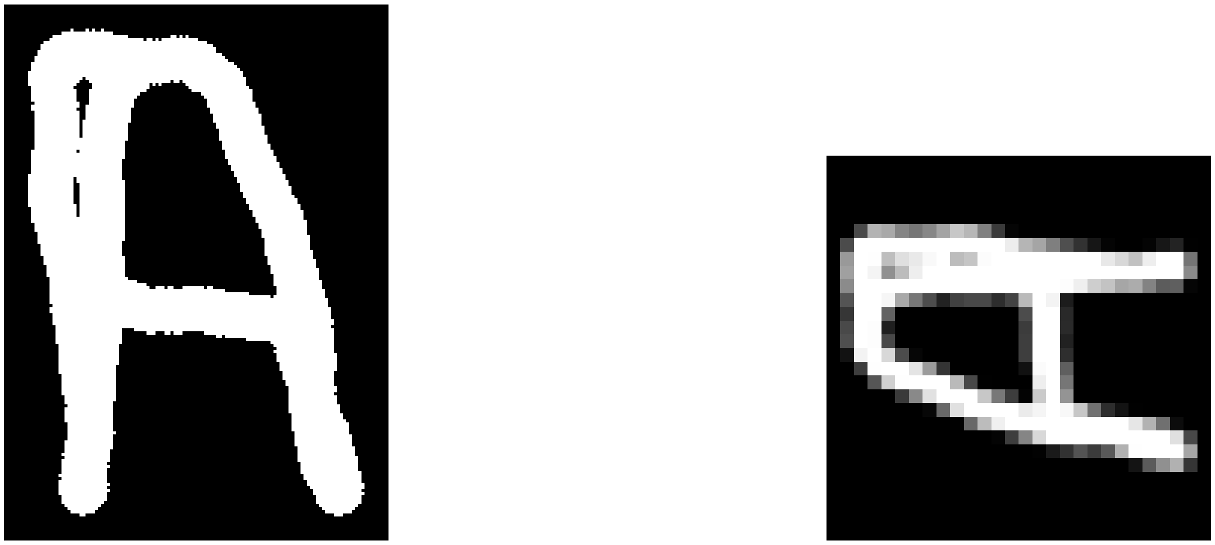
1) 90° clockwise rotation,
2) horizontal mirroring.

This transformation was empirically found to significantly improve CNN accuracy.

6) Normalization: Pixel intensities are normalized to the range $[-1, 1]$ to match the model's training distribution:

$$I_n = \frac{I_{28}}{255}, \qquad I_n = \frac{I_n - 0.5}{0.5}.$$

The final tensor has shape $1 \times 1 \times 28 \times 28$ and is passed to the CNN for inference. This entire preprocessing pipeline is visualized in Figure 1.



(a) Raw Crop (Before) (b) Preprocessed (After)

Fig. 1: The visual pipeline for preprocessing a character. (a) The raw, cropped image from the virtual canvas. (b) The same image after being centered in a square, resized to 28x28, rotated 90 degrees clockwise, and flipped horizontally to match the EMNIST dataset's orientation.

VI. Selected Listings

A. Geometry Utilities

```python
import numpy as np

def get_angle(a, b, c):
    radians = np.arctan2(c[1]-b[1], c[0]-b[0]) \
            - np.arctan2(a[1]-b[1], a[0]-b[0])
    return abs(np.degrees(radians))

def get_distance(landmark_list):
    (x1, y1), (x2, y2) = landmark_list[0],
        landmark_list[1]
    L = np.hypot(x2-x1, y2-y1)
    return np.interp(L, [0, 1], [0, 1000])
```

Listing 1: Angles and distances on normalized landmarks (`util.py`)

B. AI Mouse: Mode-Based Gesture Logic

```python
def get_finger_angles(landmarks_list):
    """Helper to get angles for all 4 fingers..."""
    angles = {}
    if len(landmarks_list) < 21: return None

    # Calculate angles for all four fingers
    angles['idx'] = util.get_angle(landmarks_list
        [5], landmarks_list[6], landmarks_list[8])
    angles['mid'] = util.get_angle(landmarks_list
        [9], landmarks_list[10], landmarks_list[12])
    angles['ring'] = util.get_angle(landmarks_list
        [13], landmarks_list[14], landmarks_list
        [16])
    angles['pinky'] = util.get_angle(landmarks_list
        [17], landmarks_list[18], landmarks_list
        [20])
    return angles

def is_peace_sign(angles, thumb_index_distance):
    """Peace Sign (Zoom Mode): Index/Middle UP,
       Ring/Pinky DOWN, Thumb OPEN"""
    return (angles['idx'] > 90 and
            angles['mid'] > 90 and
            angles['ring'] < 50 and
            angles['pinky'] < 50 and
            thumb_index_distance > 50)
```

Listing 2: Core logic for the mode-based gesture system (`air_mouse.py`)

C. Air Keyboard: Preprocessing and CNN

```python
def preprocess_crop_for_model(crop_gray):
    # pad to square
    h, w = crop_gray.shape
    size = max(h, w)
    sq = np.zeros((size, size), dtype=np.uint8)
    y_off = (size - h) // 2
    x_off = (size - w) // 2
    sq[y_off:y_off+h, x_off:x_off+w] = crop_gray

    # resize to 28, rotate + flip
    img28 = cv2.resize(sq, (28, 28), interpolation=
        cv2.INTER_AREA)
    img28 = cv2.rotate(img28, cv2.
        ROTATE_90_CLOCKWISE)
    img28 = cv2.flip(img28, 1)

    # normalize and create tensor
    img28 = (img28.astype(np.float32)/255.0 - 0.5)
        /0.5
    return torch.from_numpy(img28).unsqueeze(0).
        unsqueeze(0).float().to(device)

class CNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1,32,3,1,1)
        self.conv2 = nn.Conv2d(32,64,3,1,1)
        self.pool = nn.MaxPool2d(2,2)
        self.fc1 = nn.Linear(64*7*7,128)
        self.fc2 = nn.Linear(128,62) # UPDATED for
            ByClass
        self.relu = nn.ReLU()
    def forward(self, x):
        x = self.pool(self.relu(self.conv1(x)))
        x = self.pool(self.relu(self.conv2(x)))
        x = x.view(x.size(0),-1)
        x = self.relu(self.fc1(x))
        return self.fc2(x)
```

Listing 3: Crop preprocessing and CNN head (`air_keyboard_screenshots.py`)

a) Model Architecture: The CNN architecture, visualized in Figure 2, is a standard design for 28×28 image classification. It consists of two convolutional blocks followed by two fully-connected layers. The total number of trainable parameters is 428,350, calculated as follows:

- **conv1**: $((3 \times 3 \times 1) + 1) \times 32 = 320$ parameters.
- **conv2**: $((3 \times 3 \times 32) + 1) \times 64 = 18,496$ parameters.
- **fc1**: $((64 \times 7 \times 7) + 1) \times 128 = 401,536$ parameters.
- **fc2**: $(128 + 1) \times 62 = 7,998$ parameters.

This lightweight design is crucial for achieving real-time inference on a CPU.

VII. Implementation Details

A. Model Training

We train for 5 epochs with Adam (10^{-3}) and cross entropy on the **EMNIST ByClass** dataset (62 classes) using standard normalization. The training and validation loss curves are shown in Figure **??**. The trained weights are saved as `emnist_byclass_cnn.pth`.

VIII. Experimental Strategy

A. Dataset and Evaluation Protocol

We utilized the **EMNIST ByClass** dataset, which represents the most challenging partition of the EMNIST suite. It contains 814,255 training images across 62 unbalanced classes (digits 0–9, uppercase A–Z, lowercase a–z). All models were trained using the following hyperparameters:

- **Optimizer**: Adam (CNNs) with $\eta = 0.001$ / SGD (ResNet) with Momentum=0.9.
- **Loss Function**: Cross Entropy Loss.
- **Input Resolution**: 28×28 Grayscale (Normalized $[-1, 1]$).
- **Augmentation**: Random Rotation ($\pm 10°$) and Affine Translations to simulate air-writing jitter.

B. The "Merged Accuracy" Metric (47 Classes)

A major challenge with EMNIST ByClass is visual ambiguity. In a low-resolution 28×28 pixel grid, a handwritten 'o' (lowercase) is often pixel-identical to 'O' (uppercase) or '0' (digit). Penalizing the model for these indistinguishable cases does not reflect real-world usability, as a user's intent is usually disambiguated by context.

To address this, we introduced a **Strict vs. Merged** evaluation protocol. While the model is strictly trained on all 62 classes to learn subtle features, we map the predictions to 47 distinct visual classes during evaluation. The mapping logic merges visually overlapping lowercase letters into their uppercase counterparts, as detailed in Table I.

TABLE I: Class Merging Logic (62 $\rightarrow$ 47 Classes)

Category	**Action**
Digits (0–9)	Kept distinct
Uppercase (A–Z)	Kept distinct
Distinct Lowercase	Kept distinct (a, b, d, e, f, g, h, n, q, r, t)
Merged Lowercase	**Mapped to Uppercase ID** c, i, j, k, l, m, o, p, s, u, v, w, x, y, z

C. Model Architecture Evolution

We conducted a progressive ablation study across three architectures to solve the trade-off between inference latency and recognition accuracy.

1) Experiment 1: Baseline CNN: Our initial attempt used a lightweight 2-layer CNN (32 $\rightarrow$ 64 filters).

- **Observation:** The model suffered from significant underfitting. It struggled to capture the morphological differences between complex characters (e.g., 'g', 'q', 'y').
- **Result:** High latency in convergence and poor generalization on the test set.

2) Experiment 2: WiderCNN: To address underfitting, we increased the network width (64 $\rightarrow$ 128 $\rightarrow$ 256 filters) and introduced Batch Normalization after every convolutional layer.

- **Observation:** This stabilized training and significantly improved convergence speed. However, the shallow depth (only 3 layers) limited its ability to learn hierarchical abstractions.
- **Result:** Reached $\approx 91\%$ merged accuracy, but plateaued.

3) Experiment 3: Custom ResNet-18: We adapted the ResNet-18 architecture. Standard ResNet is designed for 224×224 images; applying it directly to 28×28 inputs causes information loss due to aggressive pooling. We modified the first layer:

$$\text{Conv2d}(1, 64, k = 3, s = 1, p = 1)$$

and removed the initial MaxPool layer. This preserved spatial fidelity while allowing us to leverage deep residual connections.

IX. Results and Discussion

A. Quantitative Analysis

Table II summarizes the performance across all experiments. The transition to ResNet-18 yielded the highest performance, proving that deeper residual connections are necessary to distinguish complex characters which shallower networks confused.

B. Visual Error Analysis

To diagnose specific failure cases, we analyzed the Confusion Matrices and Per-Class Accuracy distributions. We utilized full-width visualizations to accommodate the high dimensionality (62 classes) of the dataset.

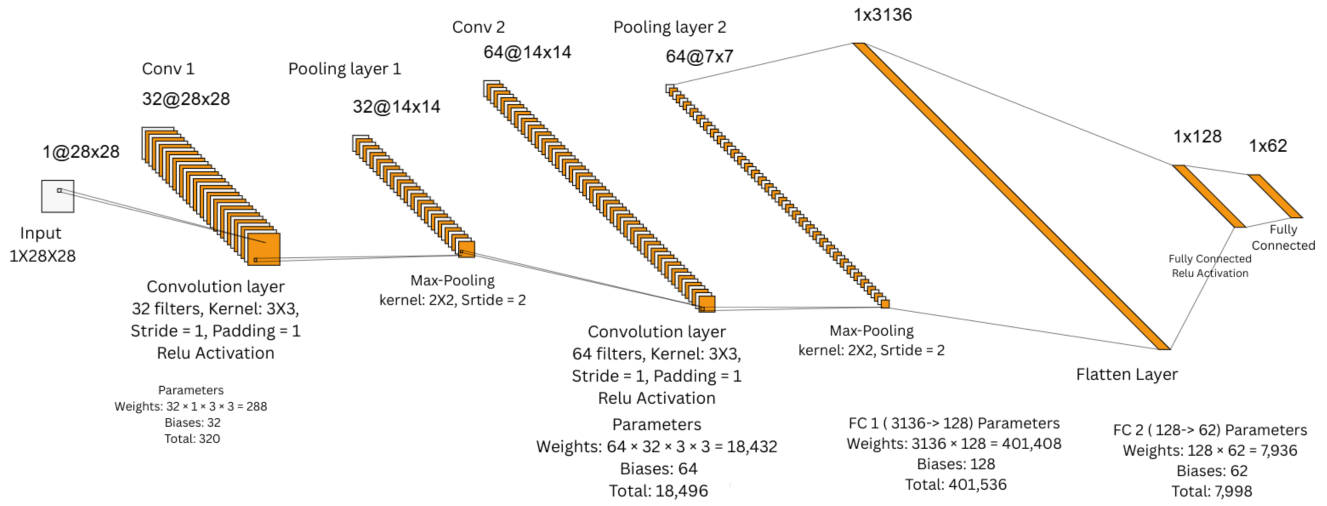


Fig. 2: A diagram of the CNN, showing the flow from a 1x28x28 input to a 62-class output. The image is from the NN-SVG tool, illustrating the downsampling at each pooling layer and the final fully connected layers.

TABLE II: Final Accuracy Comparison

Model	Strict Acc.	Merged Acc.	Status
Baseline CNN	86.1%	-	Deprecated
WiderCNN	88.03%	91.37%	Good
ResNet-18	**90.10%**	**91.80%**	**Deployed**

1) Intermediate Performance (WiderCNN): Figure 3 illustrates the performance of the WiderCNN. While the model performs well on digits, the Confusion Matrix reveals scattered off-diagonal noise, indicating frequent misclassifications between structurally similar letters (e.g., 'U' and 'V').

2) Final Performance (ResNet-18): Figure 4 demonstrates the superiority of the ResNet architecture.

- **Confusion Matrix:** The matrix shows a sharp, clean diagonal with minimal background noise. The deep residual layers effectively separated the feature space of conflicting characters.
- **Per-Class Accuracy:** The bar chart shows consistent performance ($> 95\%$) across almost all classes. The only remaining drops are in the intrinsically ambiguous '1', 'I', and 'l' cluster, which approaches the Bayes error rate for this dataset without semantic context.

C. System Latency and Voice Integration

The multimodal system achieved real-time performance metrics suitable for desktop control.

- **Vision Latency:** The anchor-offset cursor model operated at ≈ 30 FPS on a standard CPU.
- **Voice Latency:** The wake-word detection introduced a 1.2s latency. This was deemed acceptable for high-level mode switching (e.g., opening the keyboard) where immediate reaction is not critical.

X. System Refinement and Multimodal Extensions

Following the initial implementation of the vision-only keyboard and mouse, we identified a critical gap in user experience: the lack of feedback. To address this, we developed a **Multimodal Architecture** integrating a Voice Assistant ("Jarvis") and auditory feedback for the Air Mouse.

A. The Voice Assistant ("Jarvis")

To eliminate the need for physical interaction when switching modes, we implemented a voice command center using Python's `speech_recognition` and `subprocess` libraries.

1) Command Orchestration: The assistant runs in a continuous listening loop (energy threshold $= 4000$) to filter background noise. Upon detecting a wake word, it spawns independent processes for the vision tools. This decoupling ensures that if the vision script hangs, the voice assistant remains active.

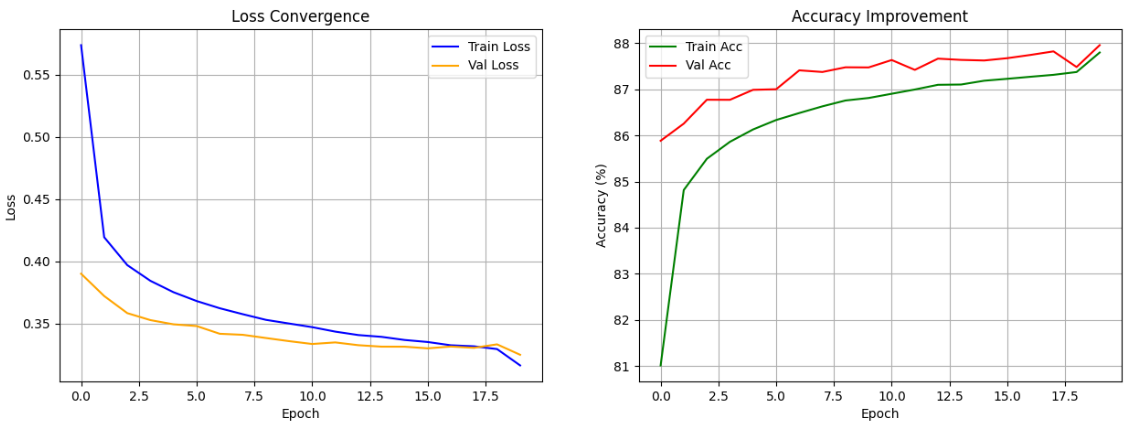
- **"Open Virtual Mouse"**: Launches `air_mouse_2.py`
- **"Open Virtual Keyboard"**: Launches `key.py`
- **"Terminate"**: Kills child processes via `SIGTERM`.

2) Robust Speech Synthesis (TTS): We utilized Windows Native Automation (VBScript) via a Python wrapper for Text-to-Speech. This approach was chosen over standard libraries to ensure speech works even in restricted environments where Python audio drivers might fail.

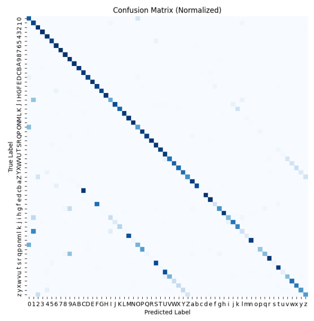
B. Voice-Enabled Air Mouse

We enhanced the standard gesture controller with concurrent threading to provide real-time auditory confirmation of actions.

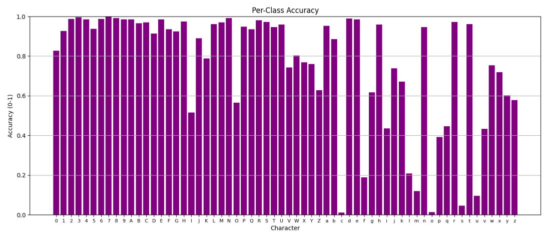
1) Threaded Feedback Loop: A naive implementation of TTS (Text-to-Speech) blocks the main execution thread, causing the video feed to freeze for 1-2 seconds while the computer



(a) Loss vs. Epoch



(b) Confusion Matrix



(c) Per-Class Accuracy

Fig. 3: **Experiment 2 Results (WiderCNN):** The model achieves 91.37% merged accuracy. Note the dip in accuracy for letters like 'l', 'I', and '1' in the bar chart, and the visible noise in the confusion matrix background.

speaks. We solved this by offloading the `speak()` function to a daemon thread.

$$\text{Latency}_{\text{Video}} = \Delta T_{\text{Process}} + \underbrace{\Delta T_{\text{TTS}}}_{\approx 0 \text{ (Threaded)}} \quad (1)$$

This ensures the cursor continues moving smoothly while the system announces "Scrolling Down" or "Right Click."

2) *Smart Cooldown Mechanism:* To prevent auditory spam (e.g., the system repeating "Scrolling... Scrolling..." ten times a second), we implemented a `SPEECH_COOLDOWN` of 3 seconds. The system checks the timestamp of the last spoken event:

$$\text{if } (T_{\text{current}} - T_{\text{last}}) > 3.0 \implies \text{Trigger TTS}$$

This creates a natural interaction flow where the system confirms the start of an action but remains silent during continuous execution.

XI. Results and Discussion

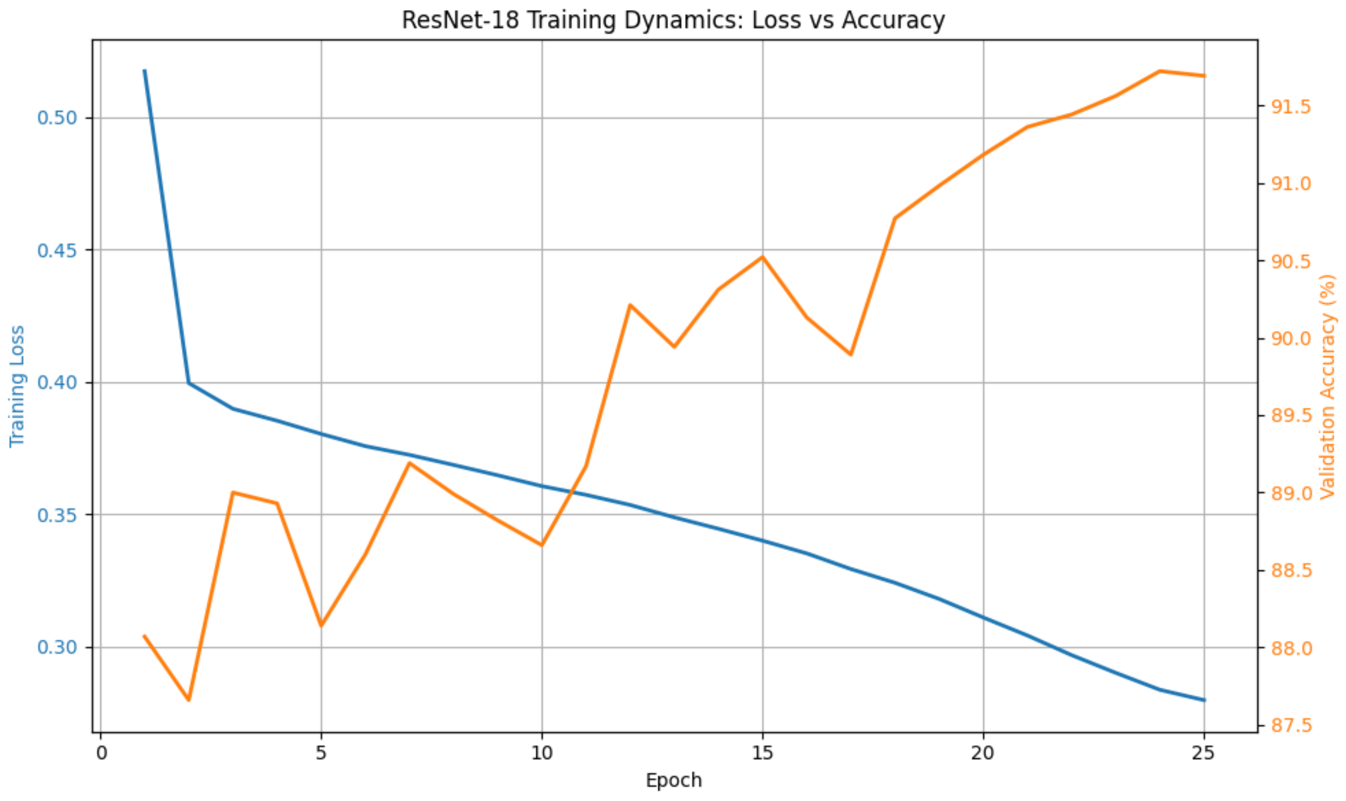
We will report quantitative accuracy on air-drawn test sets and qualitative demos. Preliminary observations: (i) the anchor-offset cursor model mitigates re-engagement jumps; (ii) segmentation with morphological closing plus horizontal gap merge is robust to varied stroke spacing; (iii) rotate+flip alignment substantially improves CNN accuracy versus raw crops.

XII. Limitations and Future Work

Limitations include sensitivity to extreme lighting/occlusions, limited multi-hand support, and reliance on EMNIST which differs from true air-writing distributions. Future work will explore domain adaptation, temporal models (e.g., CNN+RNN or transformers) for sequence recognition, multi-language support, and multimodal fusion with inertial sensors for enhanced robustness.

XIII. Conclusion

We presented a practical, real-time touchless interaction system comprising a CNN-based air keyboard and a rule-based AI mouse. Leveraging MediaPipe landmarks, OpenCV preprocessing, and PyTorch inference, our approach delivers accurate, low-latency interaction on commodity hardware. The proposed experimental plan targets systematic evaluation and extensions toward richer, more robust hands-free interfaces.



(a) Train Loss vs Val Accuracy

Fig. 4: **Experiment 3 Results (ResNet-18):** The final model achieves 91.72% merged accuracy.

Acknowledgment

We express our heartfelt gratitude to our faculty guide, Dr. Sangeeta Yadav, for her continuous guidance, valuable feedback, and encouragement throughout the development of this project. Her insights helped us refine our system and understand the deeper technical challenges involved in gesture-based interaction.

We are also thankful to the Department of Computer Science, Faculty of Technology, University of Delhi, for providing us with the resources and academic environment required for meaningful research and experimentation.

Finally, we would like to thank our classmates and friends who provided feedback during testing and helped us validate the performance and practicality of our Virtual Air Keyboard and Gesture-Based System Controller.

References

[1] C. Lugaresi, J. Tang, H. Nash *et al.*, "Mediapipe: A framework for building perception pipelines," in *Proc. CVPR Workshop on Computer Vision for AR/VR*, 2019.

[2] D. J. Sturman and D. Zeltzer, "A survey of glove-based input," *IEEE Computer Graphics and Applications*, vol. 14, no. 1, pp. 30–39, 1994.

[3] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik, "Emnist: Extending mnist to handwritten letters," *IJCNN*, 2017.