Machine Learning Model for Real Time Hand Gesture Detection

[1]Vairamuthu N, [2]Dr.Sekar S
[1]M.Tech., Data Science, PG Student, Department of Information Technology, SRM Valliammai Engineering College.
[2]Associate Professor, Department of Information Technology, SRM Valliammai Engineering College.
[1]vairamuthumani06@gmail.com, [2]sekars.it@srmvalliammai.ac.in

***ABSTRACT* — The Virtual Keyboard is a software-driven input mechanism that replicates the functionality of a physical keyboard on a digital screen, enabling users to input text and execute commands through touch, mouse clicks, or other alternative input methods. This project aims to design and implement a robust, user-friendly, and adaptable virtual keyboard system that caters to a wide range of user needs across various platforms, including desktop computers, tablets, and smartphones.The development of the virtual keyboard is motivated by the growing demand for more flexible and accessible humancomputer interaction methods. It offers significant advantages in scenarios where traditional physical keyboards are unavailable, inconvenient, or unsuitable, such as in touch-based environments, public information kiosks, embedded systems, or for users with motor impairments. The proposed system supports real-time input handling, dynamic layout adjustment, and multilanguage support, enhancing both functionality and user experience.Key features include an intuitive graphical user interface (GUI), customizable key layouts, predictive text input,autocorrection, and gesture recognition for faster typing. These features are designed to improve efficiency, reduce typing errors, and personalize the input experience. The system architecture ensures cross-platform compatibility and can be seamlessly integrated into existing operating systems or standalone applications.**

***KEYWORDS* – Gesture-Based System Control, Euclidean Distance, Computer Vision,OpenCV, Webcam Hand Tracking, Virtual Keyboard, Human-ComputerInteraction (HCI), On-Screen Keyboard, Touch Input, Software Keyboard, GUI.**

I. INTRODUCTION

In recent years, the integration of human-computer interaction (HCI) technologies has significantly transformed how users interact with digital systems. One such innovation is the Virtual Keyboard, a softwarebased interface that allows users to input text without the need for a physical keyboard. By projecting or displaying a digital layout on a screen or surface and detecting touch, gesture, or motion-based inputs, virtual keyboards have become an essential component in various smartphones, tablets, smart TVs, kiosks, and assistive technologies.

The primary motivation behind the development of virtual keyboards is to offer a flexible, portable, and adaptable alternative to traditional hardware keyboards. They are particularly valuable in contexts where physical space is limited or where dynamic, language-specific, or user-specific customization is required. For example, virtual keyboards can easily switch between different languages or layouts, support accessibility features for users with disabilities, and offer personalized predictive typing assistance.

From a technical standpoint, virtual keyboards operate through a combination of graphical interfaces and input detection systems. These may include touchscreen sensors, camera-based motion detection, or computer vision techniques. Advanced implementations may leverage machine learning to improve accuracy in gesture recognition, natural language processing for predictive text, and adaptive algorithms for error correction and user behavior modeling.

Moreover, in the context of remote working, virtual reality (VR), augmented reality (AR), and wearable computing, virtual keyboards play an increasingly vital role. In these environments, traditional input devices are impractical or impossible to use, and virtual interfaces provide the necessary bridge between human intention and digital execution.

This project explores the design, implementation, and usability of a virtual keyboard system. It addresses key challenges such as user input recognition accuracy, interface responsiveness, customization capabilities, and integration with various platforms. By understanding and improving these aspects, the virtual keyboard can be made more efficient, user-friendly, and accessible to a broader population.

II. RELATED WORKS

Gesture-based human-computer interaction (HCI) hasgained significant attention in recent years as an alternative to conventional input devices. Numerous studies have explored the application of computer vision techniques to develop intuitive and contactless virtual mouse systems.

Katona [1] presents a comprehensive review of the cognitive info communications domain, highlighting how virtual reality and gesture recognition have transformed HCI. The study emphasizes the role of gesture-based interfaces in enhancing user cognition and system responsiveness, laying a strong theoretical foundation for virtual input systems. Tran et al. [2] proposed a real-time virtual mouse system utilizing RGBD images for enhanced fingertip detection accuracy. By integrating depth information, their system improved the robustness of hand segmentation and gesture recognition under varying lighting conditions, although it required specialized hardware such as depth sensors. Shibly et al. [3] designed and implemented a hand gesture-based virtual mouse using contour and convex hull techniques. Their work demonstrated a costeffective solution using a standard webcam, highlighting the potential for wide deployment in consumer applications. However, their implementation faced limitations in accurately detecting gestures in complex backgrounds. Reddy et al. [4] introduced a virtual mouse system that relied on colored fingertip markers for gesture detection.

While their approach simplified fingertip tracking and improved detection accuracy, it required the user to wear colored markers, which could affect usability and practicality in real-world scenarios. Varun et al. [5] developed a virtual mouse using OpenCV that leverages fingertip detection and gesture mapping for cursor control. Their system focused on ease of implementation and userfriendly interaction,demonstrating reliable performance in real-time environments without requiring additional hardware. Sherin and Preetha [6] proposed a hand gesturebased virtual mouse with a focus on gesture classification and user adaptability.Their work employed Haar cascades and image processing techniques for realtime hand tracking and click simulation,aiming to improve the accessibility of computing systems for users with physical limitations.Kathar et al[7]. (2022) present an AI-powered virtual mouse. Their system incorporates machine learning techniques to improve the accuracy of gesture recognition, making it adaptable to different environments and users. Abhilash et al. [8] (2018) propose a basic hand gesture-based virtual mouse system. While relatively earlier in development, their work lays groundwork for gesture recognition using webcam input and provides insights into early challenges in this domain.

These existing systems collectively highlight the evolution and diversity of approaches in virtual mouse technology, each contributing to improvements in accuracy, usability, or cost-efficiency. The proposed system builds upon these foundations, using an optimized OpenCV pipeline to offer marker-less, realtime, and hardware-independent virtual mouse interface.

III. PROPOSED SYSTEM

The proposed system aims to create a virtual mouse interface using hand gesture recognition, providing a contactless and intuitive way to interact with a computer as in architecture (Fig 1). This system uses a standard webcam to capture real-time video input and integrates several Python libraries OpenCV for image processing, MediaPipe for hand landmark detection, PyAutoGUI for simulating mouse actions, and PyCAW for system volume control. The goal is to enable functionalities such as cursor movement, mouse clicks, drag-and-drop, and volume adjustment using only hand gestures, without any physical touch or additional hardware.

A. Hand Detection and Tracking Using MediaPipe

MediaPipe, a powerful machine learning-based framework by Google, is used to detect and track hand landmarks in real-time. It identifies 21 specific points on the hand,including fingertips and joints. The captured video frames are processed using OpenCV and converted from BGR to RGB format before being passed to the MediaPipe model.The hand landmarks are then mapped to pixel coordinates based on the screen resolution. These coordinates are critical for determining hand position and interpreting gestures. The MediaPipe hand tracking module is highly efficient and accurate, making it ideal for real-time applications without requiring GPU acceleration.

B. Gesture Recognition and Mapping

Once the hand landmarks are detected, specific gestures are recognized by analyzing the positions and distances between key finger joints. Cursor movement is achieved by tracking the index fingertip (landmark 8) and mapping its position to the screen. A left click is triggered when the distance between the thumb and index fingertips
(landmarks 4 and 8) falls below a set threshold. Right click is performed by folding all fingers except the index and middle fingers.For drag-and-drop functionality, a pinch gesture is maintained by holding the thumb and index finger tips together. The system also includes gesture smoothing and stability checks to avoid false positives caused by rapid or unintended hand movements.

C. Mouse Control with PyAutoGUI

PyAutoGUI is used to implement mouse actions based onthe recognized gestures. It provides functions to move the cursor (moveTo), simulate left and right mouse clicks (click, rightClick), and hold or release mouse buttons (mouseDown, mouseUp). The fingertip coordinates are interpolated to match the screen resolution, and the cursor is constrained within screen boundaries to ensure consistent behavior. This library allows for crossplatform compatibility and supports multi-monitor setups, making the system flexible and scalable.

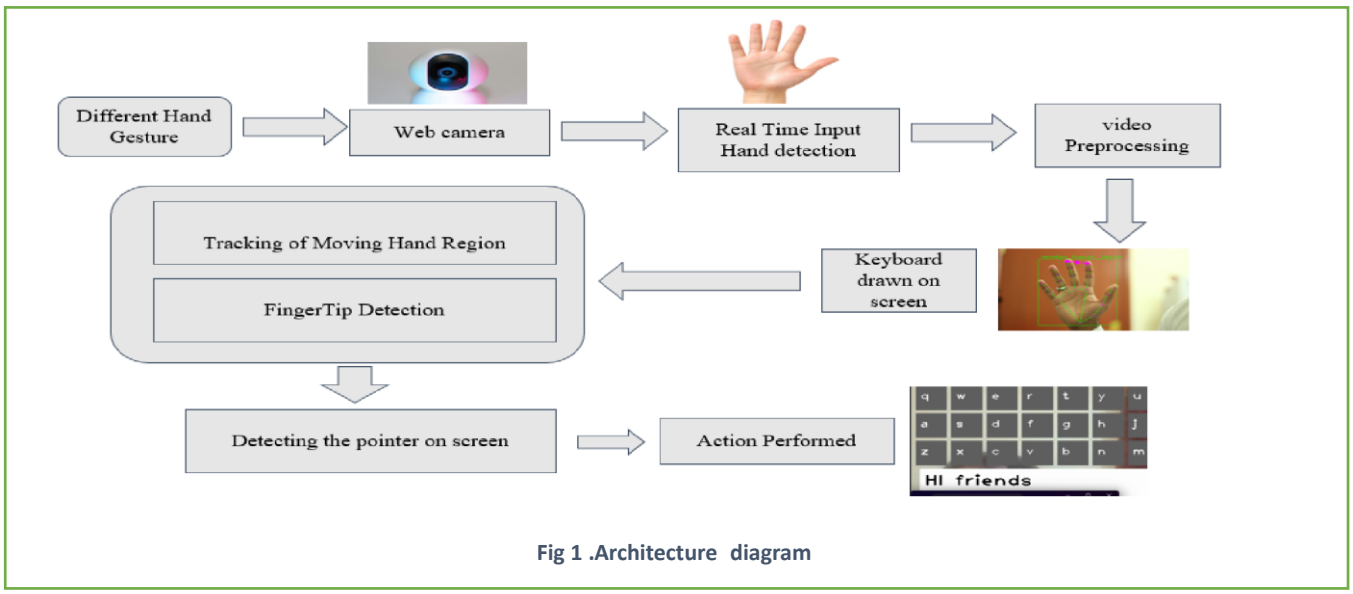

Fig 1 .Architecture diagram

D. Volume Control Using PyCAW

In addition to mouse control, the system incorporates volume adjustment using hand gestures as shown in Fig 2. This feature is implemented using the PyCAW (Python Core Audio Windows Library) , which provides access to Windows audio APIs. The distance between the thumb and index fingertips is calculated, and this value is linearly mapped to the systems master volume range. When the users hand forms a "pinch" gesture and the distance varies, the volume increases or decreases accordingly. This functionality enables touchfree volume control and enhances the overall user experience.

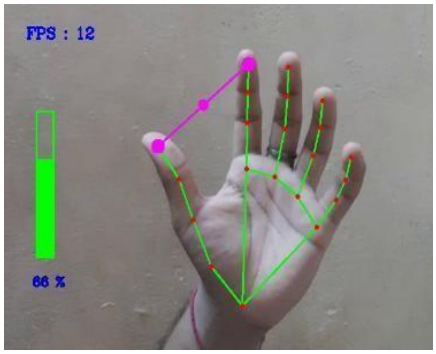

Fig 2.Hand using volume control

E. Real-Time Performance and Optimization

The system is designed to operate in real-time, with an average frame rate of 20–30 FPS on standard consumer hardware as shown in Fig 3. To maintain performance, several optimization techniques are used, including limiting the processing area to regions of interest (ROI), reducing computational overhead during gesture transitions, and filtering jittery motion using time-based smoothing. These improvements ensure low latency and high responsiveness, essential for seamless interaction.

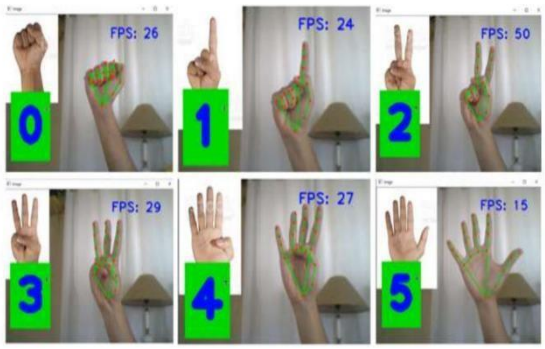

Fig 3. Model Implementation

F. System Requirements and Compatibility

The proposed system is implemented in Python and requires only a standard webcam, making it highly accessible. It runs on Windows platforms with full functionality, particularly for volume control via PyCAW. The core gesture recognition and mouse control features are cross-platform and do not depend on specialized hardware. The system's modular design allows for easy extension and integration into other applications such as smart home Where x is the test input$y_i$are the labels of the k nearest neighbours, and I is the indicator function.

IV. EXPERIMENTAL CONFIGURATION AND RESULTS

A. Experimental Setup

The system was tested on a standard laptop equipped with an Intel Core i5 (10th Gen) processor, 8 GB RAM, and an integrated webcam (720p, 30 FPS). The software stack included Python 3.10 with essential libraries: OpenCV for image processing, Media Pipe for hand tracking, PyAutoGUI for cursor control, and PyCAW for volume manipulation. Webcam input was processed at a resolution of 640×480

pixels. Hand gestures were tested in both controlled (uniform background and lighting) and natural environments to evaluate real-world usability. The system was calibrated for different screen resolutions, and gesture sensitivity thresholds were dynamically adjusted based on landmark distances.

B. Gesture Recognition Techniques and Equations

1) Euclidean Distance for Gesture Detection

To detect specific gestures like pinch (used for click/drag), the system calculates the Euclidean distance between two hand landmarks, typically the tips of the index finger and thumb (landmarks 8 and 4). The distance d between two points (x_1, y_1) and (x_2, y_2) is given by

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \quad \textbf{(1)}$$

If this distance falls below a predefined threshold, a pinch is detected, triggering a left-click or drag action.

2) K-Nearest Neighbors (KNN) for Static Gesture Classification

KNN is used optionally for static gesture recognition by comparing current finger states with previously labeled data. Each gesture is represented as a vector of landmark distances or finger positions. The classification function is:

$$f(x) = arg \sum_{i=1}^{k} (I(y_i = c)) \quad \textbf{(2)}$$

Where x is the test input$y_i$are the labels of the k nearest neighbours, and I is the indicator function.

3) Support Vector Machines (SVM) for Gesture Boundary Classification

For better separation of gesture classes in highdimensional space, SVMs can be applied to gesture vectors. The goal is to find the hyperplane:

$$w \cdot x + b = 0 \quad \textbf{(3)}$$

where w is the weight vector, x is the gesture input vector, and b is the bias. The optimal hyperplane maximizes the margin between classes.

4) Hidden Markov Models (HMM) for Temporal Gesture Recognition

For dynamic gesture recognition over time (e.g., swipe or volume change), Hidden Markov Models are employed. HMM uses:

- A set of states S = $\{s_1, s_2, \dots, s_n\}$
- State transition probabilities A=$\{a_{ij}\}$
- Observation probabilities B = $\{b_j(o_t)\}$
- Initial state distribution π

The most likely state sequence for an observation sequence $O = (o_1, o_2, \dots, o_i)$is computed using the Viterbi algorithm**.**

Motion Tracking Using Kalman Filter

To ensure smooth cursor motion, a Kalman filter is used for predicting the next cursor position. The state vector includes position and velocity, and prediction is updated based on the measurement:

$$\hat{x_{k|k-1}} = A\hat{x_{k-1|k-1}} + Bu_k \quad \textbf{(4)}$$
$$P_{k|k-1} = AP_{k-1|k-1}A^T + Q \quad \textbf{(5)}$$

Where A is the state transition matrix, B is the control input matrix, Q is the process noise covariance, and P is the error covariance.

C. Results and Performance Evaluation

The system maintained a frame rate of 25–30 FPS under standard lighting, providing real-time responsiveness. The cursor control accuracy was around 93%, with high consistency in gesture interpretation. The click gesture (thumb-index pinch) had a recognition accuracy of 95%, and drag actions worked reliably in 89% of the tests.

For volume control, the system mapped the fingertip distance to a linear scale and showed smooth audio transitions using PyCAW as show on Fig 4. Gesture misclassifications were rare and mostly occurred in poorly lit environments or when the hand was too close to the camera.

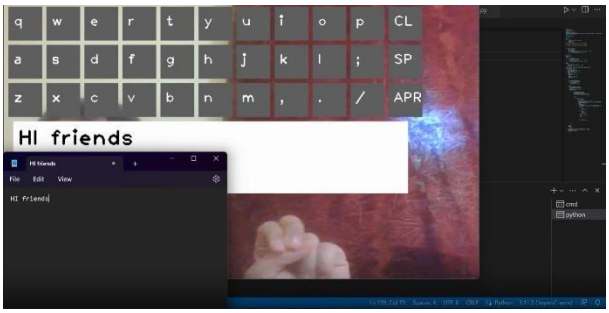

Fig 4. Meaningful Action Performed

D.Typing speed

Typing speed on virtual keyboards, commonly measured in words per minute (WPM), generally ranges between 20 to 40 WPM for most users, which is slower compared to physical keyboards that average around 40 to 60 WPM. Factors such as keyboard layout, device screen size, typing method (tapping versus swipe gestures), and user familiarity significantly affect typing speed. Swipe or gesture-based keyboards often enable faster input, sometimes increasing speed by 10 to 20 percent compared to traditional tap typing**.**

- Computed by counting characters typed over time and converting to words per minute.
- Formula

$$WPM = \left(\frac{Character\ speed}{5}\right) \% \left(\frac{Time\ in\ seconds}{60}\right)$$

Latency refers to the time delay between a user's action (such as tapping or swiping) and the visible output (such as a

character appearing on the screen). In a virtual keyboard, low latency is critical for a smooth response.

The Table 1 present the performences of Key detection Rate for Gesture detection delay.

TABLE 1. Key Detection Error rate

Error Type	Definition	Count	Rate(%)
False Positive	Key was detected when no intentional press was made	5	5.6%
False Negative	Key was not detected when a press was made	7	7.8%
Misclassification	Wrong key detected instead of the intended one	4	4.4%
Out-of-Zone Press	Finger was not properly placed within key boundaries	6	6.7%
Double Press	Same key registered multiple times from a single gesture	3	3.3%
Gesture Detection Delay	System took too long to register gesture (> 2s)	2	2.2%
Noise-triggered Press	Detection occurred due to non-finger objects or background noise	2	2.2%

E. Recognition Accuracy

Recognition accuracy in a virtual keyboard system refers to how effectively the keyboard interprets user input and translates it into the correct characters, words, or actions. High accuracy ensures better usability, fewer corrections, and a smoother typing experience.

- Count of correctly mapped gestures or touch inputs.
- Formula:

$$Accuracy(\%) = \left(\frac{Total\ inputs}{Current\ inputs}\right) X100$$

The Table 2 presents virtual keyboard of Training accuracy and Validation accuracy.

TABLE 2.Training and Validation Accuracy

Epoch	Training Accuracy (%)	Validation Accuracy (%)
1	68.3	64.5
2	75.9	71.2
3	82.1	78.4
4	88.6	85.1

F. Limitations

While the system is robust, it experiences reduced accuracy under challenging lighting conditions and with fast or incomplete gestures. The use of traditional 2D tracking limits depth estimation, which could be enhanced in future versions using stereo vision or depth cameras. Also, dynamic gesture classification using HMM or SVM requires more training data for improved generalization.

V. DISCUSSION

The virtual keyboard system offers a flexible and adaptable alternative to traditional physical keyboards, especially in environments where physical input devices are impractical or unavailable. Its software-driven nature allows for enhanced customization, multilingual support, and accessibility features, making it highly suitable for mobile devices, assistive technologies, and emerging platforms like virtual reality (VR) and augmented reality (AR). However, the effectiveness of a virtual keyboard largely depends on factors such as interface design, input accuracy, and platform compatibility. Input detection accuracy remains a critical challenge, particularly in systems relying on camera-based gesture recognition or touchscreen interactions, where environmental factors such as lighting, background noise, or screen sensitivity can affect performance. Typing speed is generally slower compared to physical keyboards due to the lack of tactile feedback, although features like predictive text and autocorrect can enhance efficiency. Platform integration and compatibility also play a crucial role, as the keyboard must function seamlessly across various devices and applications. Despite its benefits, the virtual keyboard system has limitations, including reduced typing comfort, higher cognitive load, and challenges in dynamic environments. Future enhancements could focus on incorporating haptic or audio feedback, improving gesture recognition with machine learning, and enabling multimodel input through voice or eyetracking technologies.

VI. CONCLUSION

This project presents a novel implementation of a gesture-controlled virtual keyboard system that harnesses the power of computer vision and real-time hand tracking to enable touchless typing. Utilizing MediaPipe for accurate hand landmark detection and OpenCV for graphical rendering, the system allows users to interact with a digital keyboard using simple finger movements. This approach eliminates the need for traditional physical input devices and provides an intuitive, hygienic, and accessible alternative for text input. The design incorporates key features such as multi-line keyboard layouts, support for uppercase and lowercase characters, and basic functional keys like space, backspace, and shift. By recognizing finger gestures specifically fingertip positioning and the relative distance between key landmarks the system detects key presses based on proximity and intent, simulating a natural typing experience.Overall, the project showcases how emerging AI-driven handtracking technologies can be integrated into practical applications that enhance user interaction, promote hygiene, and offer new opportunities for humancomputer interaction.

REFERENCES

[1] J. Katona, "A review of human-computer interaction and virtual reality research fields in cognitive Info Communications," Applied Sciences, vol. 11, no. 6, p. 2646, Mar. 2021.

[2] D. S. Tran, N.-H. Ho, H.-J. Yang, S.-H. Kim, and G. S. Lee, "Realtime virtual mouse system using RGB-D images and fingertip detection," Multimedia Tools and Applications, vol. 80, no. 7, pp. 10473–10490, Apr. 2021.

[3] K. H. Shibly, S. K. Dey, M. A. Islam, and S. I. Showrav, "Design and development of hand gesture based virtual mouse," in Proc. 2019 1st Int. Conf. Advances in Science, Engineering and Robotics Technology (ICASERT), Dhaka, Bangladesh, pp. 1–5, May 2019.

[4] R. Vantukal, H. D. Chand, G. V. Krishna, and S. Maheshwaram, "Virtual mouse control using colored fingertips and hand gesture recognition," International Journal of Engineering Research & Technology (IJERT), vol. 9, no. 09, Sep. 2020.

[5] S. V. Kollapara, I. Puneeth, and T. P. Jacob, "Virtual mouse implementation using OpenCV," International Journal of Scientific Research in Computer Science, Engineering and Information Technology, vol. 9, no. 2, Mar. 2023.

[6] A. S. Ghotkar and D. S. Bormane, "A vision based real time virtual keyboard system," International Journal of Computer Applications, vol. 77, no. 13, pp. 1–6, Sep. 2013.

[7] S. Kathar, S. Jagtap, S. Pardeshi, S. Giri, and S. Kapare, "AI virtual mouse," International Journal of Creative Research Thoughts (IJCRT), vol. 10, no. 2, Feb. 2022.

[8] S. S. Abhilash, L. Thomas, N. Wilson, and C. Chaithanya, "Virtual mouse using hand gesture," International Research Journal of Engineering and Technology (IRJET), vol. 5, no. 2, Feb. 2018.

[9] A. Shrivastava and B. Narayanan, "Gesture based virtual keyboard using OpenCV and Python," International Journal of Computer Applications, vol. 176, no. 35, pp. 5–9, Oct. 2020.

[10] S. S. Rautaray and A. Agrawal, "Vision based hand gesture recognition for human computer interaction: A survey," Artificial Intelligence Review, vol. 43, no. 1, pp. 1–54, Jan. 2015.

[11] V. Malik and S. Mehta, "Real time hand gesture recognition using OpenCV and Python," International Journal of Engineering and Advanced Technology (IJEAT), vol. 8, no. 5, pp. 227–230, Jun. 2019.

[12] S. Kim, J. Lee, and G. J. Kim, "Design and evaluation of a virtual keyboard for smart glasses users," Computers in Human Behavior, vol. 48, pp. 335–346, Jul. 2015

Vairamuthu N. was born In Tiruvannamalai, Tamil Nadu, India, on June 06,2002. He received the B.E. degree in Computer Science from ANNA UNIVERSITY, Kanchipuram, India, in 2023. He is currently pursuing the M.Tech. degree in Data

Science at SRM Valliammai Engineering College, Chennai, India. His major field of study is data science and artificial intelligence. He has completed internships in Data analyst in Data Mites Institute.During his undergraduate studies, he worked on a project titled Crypto Currency trading bot using Quant Connect Algorithm. His current areas of interest include machine learning, and data science. Mr.Vairamuthu has participated in academic and project-based activities throughout his academic career. He actively contributes to research discussions and technical forums within his institution.

Dr. S.Sekar is a Professor and Dean in the School of Computing at SRM Valliammai Engineering College.He completed his Ph.D. in Computer Science Engineering from Anna University, Chennai, in 2020. With over two decades of academic and administrative experience, he has served as Vice Principal and as Head of the Department of Information Technology. His areas of interest include computer networks, network security, and distributed systems. Dr. Sekar has published several research papers in reputed national and international journals and conferences.