

Edge-based Face Recognition Pipeline Optimization for Class Attendance System

¹Mutia Fadhilla*, ²Rizky Wandri, ³Des Suryani, ⁴Yudhi Arta

^{1,2,3,4}Program Studi Teknik Informatika, Fakultas Teknik, Universitas Islam Riau

*e-mail: tiadfadhilla@eng.uir.ac.id

(**received:** 1 September 2026, **revised:** 16 September 2026, **accepted:** 18 September 2026)

Abstract

Manual attendance recording remains inefficient and may enable proxy attendance, while cloud-based facial recognition introduces network dependency, latency, and biometric-data privacy concerns. Edge processing offers a local alternative, but limited computing resources require pipeline optimization. This study conducted a descriptive single-run evaluation of five optimization strategies on a Raspberry Pi 5: frame downscaling, static region of interest (ROI), prototype caching, sample interval tuning, and face detector comparison. The pipeline used FaceNet embeddings, cosine-similarity-based open-set recognition, and track-based temporal confirmation. Evaluation involved three prerecorded 1920×1080 classroom videos totaling 16 min 23 s, an enrollment database of 18 students, and 53 student-video attendance instances. Frame downscaling at 0.75 reduced runtime from 1,274.83 s to 947.72 s, equivalent to 25.66%, while confirmed attendance instances decreased slightly from 47 to 46. Effective sampled-frame throughput increased by 34.62%, and average RAM usage decreased by 6.86%, although CPU utilization and peak temperature increased. Prototype caching reduced preparation time but had limited effect on end-to-end runtime, whereas static ROI, sampling intervals above 5 s, and YuNet did not improve overall pipeline performance. Frame downscaling was therefore the most promising strategy in the evaluated setup. However, repeated trials and independent classroom-video testing are required before generalizing the results or considering the pipeline deployment-ready.

Keywords: attendance system, edge computing, face recognition, pipeline optimization, raspberry pi

1 Introduction

Accurate and efficient recording of student attendance is a crucial component of the learning process in higher education institutions [1]. To date, most institutions still rely on manual methods such as attendance sheets distributed during learning activities [2], [3]. While easy to implement, this method has several weaknesses that significantly impact the integrity of attendance data. One of the main problems is the vulnerability to cheating such as signing attendance sheets, where absent students are replaced by their classmates to sign the attendance sheet [4]. In addition, lecturers need to spend additional time verifying and digitizing attendance records, which poses a risk of data entry errors and takes up teaching time that could be used more productively [5]. Paper-based records are also prone to being lost, damaged, or becoming difficult to manage and archive in the long term.

Machine-learning methods have also been explored in higher-education contexts for student-related analytical tasks, including the identification of depression among university students [6]. In the administrative context, computer vision technology offers further opportunities to support institutional processes, including the automation of student attendance recording. In particular, the development of computer vision technology in the field of facial recognition, opens up opportunities for automating attendance recording without disrupting learning activities [7], [8]. Students can be passively identified from classroom video, potentially reducing administrative effort and limiting simple proxy attendance based on identity substitution. However, the present system does not incorporate presentation attack detection; therefore, resistance to spoofing attacks using photographs, recorded videos, or display devices is outside the scope of this study. On the other hand, most conventional facial recognition systems rely on cloud-based deep learning frameworks [9], [10]. This approach presents several practical challenges, including network latency that impacts system responsiveness, concerns about the privacy of biometric data sent to external servers, and dependence on stable internet connectivity [11]. These conditions are not always available in educational institutions with

<http://sistemasi.ftik.unisi.ac.id>

limited network infrastructure, making cloud-based solutions less practical for widespread implementation in real-world classroom environments.

Edge computing offers an alternative that addresses these limitations by enabling data processing to be performed directly on the local device [12]. This approach reduces latency, enhances privacy because biometric data never leaves the device, and eliminates the reliance on an internet connection [13]. Single-board computers such as the Raspberry Pi have become a popular choice for edge deployments due to their affordability, low power consumption, and mature development ecosystem. However, deploying facial recognition on edge devices faces challenges due to limited computing resources. Modern facial detection models such as MTCNN [14], RetinaFace [15], and variants of YOLO [16] typically demand significant GPU and memory capacity to maintain acceptable accuracy and speed. On low-power edge devices, the performance of these models often suffers significantly, leading to increased latency and thermal throttling. Therefore, for facial recognition systems to run effectively on edge devices, careful pipeline optimization is required to balance recognition accuracy and resource efficiency.

Various optimization strategies have been proposed in the literature to address resource limitations on edge devices. At the model level, some researchers have proposed lightweight architectures specifically designed for embedded devices, such as EdgeFace [17], which combines the power of CNNs and Transformers with a very small number of parameters, and YuNet [18], which offers a good accuracy-speed trade-off with only 75,856 parameters. Other approaches utilize pruning, quantization, and knowledge distillation techniques to reduce model size without significant accuracy loss [19], [20]. At the pipeline level, commonly used strategies include reducing frame resolution, limiting the processing area through regions of interest (ROIs), adjusting frame sampling intervals, and selecting lighter detectors. Previous research has also shown that intelligent frame analysis mechanisms can improve the throughput of facial recognition systems on embedded devices [21], and edge-cloud collaboration with cropping faces on edge devices before sending them to the cloud can significantly improve inference speed [11]. While each of these strategies can theoretically reduce the computational load, their impact on the end-to-end facial recognition pipeline on a specific edge device may vary depending on the hardware characteristics, usage scenarios, and interactions between pipeline stages. Empirical validation of each strategy in a concrete deployment context is essential to generate reliable configuration recommendations.

This study aims to evaluate five edge-based facial recognition pipeline optimization strategies for a class attendance system on the Raspberry Pi 5, namely frame downscaling, static ROI, prototype caching, sample interval tuning, and detector comparison. The main contributions of this study are: (1) a systematic evaluation of five facial recognition pipeline optimization strategies on the Raspberry Pi 5 edge device for class attendance scenarios; (2) identification of configurations that provide the best trade-off between recognition accuracy and system efficiency; and (3) empirical findings on the variability of the impact of optimization strategies on end-to-end pipeline performance, which can be a practical reference for developers of facial recognition-based attendance systems on edge devices.

2 Literature Review

The application of facial recognition technology in attendance systems has been widely investigated as an alternative to conventional attendance recording, particularly to improve automation, accuracy, and efficiency. Recent research has also explored edge-based processing to reduce dependence on centralized servers and network connectivity. Parhi et al. [22] proposed an Intelligent Online Attendance Tracking System (IOATS) based on facial recognition and edge computing for monitoring attendance in online classes. The system performs video capture, face detection, facial encoding extraction, and face verification directly on the student's laptop or personal computer, while only the final attendance information is submitted to a central database. This edge-based architecture is intended to reduce network utilization, protect facial data privacy, and accelerate processing by eliminating the need to continuously transmit images to a remote server. The system uses OpenCV for video capture, Dlib with Histogram of Oriented Gradients (HOG) for face detection, and a ResNet-based model to generate 128-dimensional facial encodings, which are compared using Euclidean distance. Attendance is determined from repeated face recognition during the class, with

<http://sistemasi.ftik.unisi.ac.id>

images captured at 10-second intervals. Experimental evaluation using a face recognition dataset containing 31 classifications and 2,652 face images reported an accuracy of 97.73%, precision of 99.60%, recall of 98.04%, and F1-score of 98.81%, with an AUC of 0.88. Although IOATS demonstrates the feasibility of combining facial recognition with edge computing for attendance tracking, its primary emphasis is on developing an online attendance mechanism and reducing network and privacy constraints rather than systematically optimizing the facial recognition pipeline on resource-constrained edge hardware. In contrast, the present study focuses on the optimization of the end-to-end recognition pipeline on a Raspberry Pi 5 by experimentally evaluating frame downscaling, static region of interest (ROI), prototype caching, sample interval tuning, and face detector selection. Thus, while IOATS establishes the importance of edge-based facial recognition for attendance, this study investigates how individual pipeline configurations affect the trade-off between recognition performance and computational efficiency.

Another relevant study was conducted by Kurniasari et al. [23], who developed an Intelligent Attendance Monitoring System using real-time face recognition and Raspberry Pi. The system was designed to automate attendance recording in educational and workplace environments using a Raspberry Pi with a camera and Wi-Fi connectivity. Its recognition process consists of image preprocessing by converting RGB images into grayscale, face detection using the Haar Cascade Classifier, and facial recognition using the Local Binary Patterns (LBP) algorithm. Detected faces are cropped from the video stream and compared with facial information stored in the database before the corresponding attendance record is generated. The researchers evaluated the system under different distances and illumination conditions. Ten individuals were used in the testing process, with seven registered in the database, and facial samples were repeatedly tested under different environmental conditions. The distance experiment showed average accuracies of 92.4%, 84.1%, and 52.7% at distances of 30, 50, and 80 cm, respectively. Similarly, the lighting experiment achieved average accuracies of 38.5%, 91.3%, and 76.7% at 20, 100, and 200 lux, respectively, indicating that distance and illumination substantially influence recognition performance. Although this study is closely related to the present research because both employ Raspberry Pi for real-time attendance and use Haar Cascade as a lightweight face detector, its evaluation mainly examines recognition accuracy under variations in distance and illumination. It does not systematically investigate the optimization of multiple stages of the recognition pipeline or quantify their effects on computational resources such as runtime, CPU utilization, RAM usage, and device temperature. The present study therefore extends this line of research by treating the recognition process as an end-to-end pipeline and experimentally comparing several optimization strategies on the Raspberry Pi 5. In addition to recognition accuracy, the proposed evaluation considers runtime, effective processing throughput, CPU, RAM, and temperature, enabling the selection of configurations based on the trade-off between recognition reliability and resource efficiency.

Recent studies have addressed edge-based face recognition through model-level and pipeline-level optimization. EdgeFace employs a lightweight hybrid CNN–Transformer architecture to reduce the complexity of face representation models while maintaining competitive recognition performance [17]. At the face-detection stage, compact architectures such as YuNet [18] and EResFD [24] have been developed to improve the trade-off between detection accuracy and computational efficiency. Model-compression studies have also investigated joint pruning and quantization [20], while more recent work has applied filter pruning specifically to lightweight face detectors [25]. At the pipeline level, CFSM employs selective frame analysis to improve face recognition throughput on embedded systems [21], whereas Oroceo et al. [11] distribute face detection and recognition tasks between edge and cloud resources to reduce inference latency and communication overhead. More recently, tracking and heterogeneous hardware utilization have been combined to avoid redundant face recognition operations on an accelerator-equipped edge platform [26]. Collectively, however, these studies optimize specific models, pipeline components, or hardware configurations and do not jointly compare multiple pipeline-level strategies in a CPU-based Raspberry Pi 5 attendance system using identical classroom videos and both recognition and system-resource metrics. Therefore, the research gap addressed in this study is not the absence of edge-based face recognition systems, but the limited end-to-end comparative evidence concerning the effects of frame downscaling, static ROI, prototype caching, temporal sampling, and detector selection in a resource-constrained classroom attendance scenario.

Based on these previous studies, facial recognition has demonstrated strong potential for automating attendance systems, while edge computing provides advantages in terms of reduced network dependency, privacy, and local processing. However, the two reviewed studies address different aspects of the problem: IOATS emphasizes edge-based attendance tracking and privacy for online classes, whereas Kurniasari et al. focus on the accuracy of real-time facial recognition on Raspberry Pi under different distance and lighting conditions. Neither study systematically evaluates how multiple optimization strategies across the facial recognition pipeline influence the overall performance of a class attendance system on a constrained edge device. This gap motivates the present study, which evaluates five pipeline optimization strategies frame downscaling, static ROI, prototype caching, sample interval tuning, and detector comparison on a Raspberry Pi 5. The study further evaluates both recognition-related and system-efficiency metrics to identify configurations that provide an appropriate balance between attendance recognition accuracy and computational efficiency.

3 Research Method

This section describes the experimental framework used to evaluate edge-based facial recognition pipeline optimization strategies. The description includes the testing system and environment, pipeline architecture, evaluated optimization strategies, test datasets and scenarios, and evaluation metrics used.

3.1 Testing System and Environment

Testing was conducted on a Raspberry Pi 5 as a target edge device with 8 GB RAM specifications and an active cooler module to maintain device temperature during processing. The operating system used was Raspberry Pi OS 64-bit Bookworm, with the pipeline running in a Docker container based on the python:3.11-slim image to maintain the reproducibility of the experimental environment. As a comparison, initial testing was also conducted on a local device with Apple M2 processor specifications (8-core), 16 GB RAM, and macOS Tahoe 26.4.1 operating system, to obtain a reference for pipeline performance on non-edge devices.

Python versions used were Python 3.9.6 on the local device and Python 3.11 in the Raspberry Pi 5 container. This difference in Python versions was based on practical considerations: 3.9.6 is the default version on macOS, which is used as the development environment, while 3.11 was chosen for the Docker container at the time of the study. Because all optimization strategy comparison experiments were conducted on the same edge device (Raspberry Pi 5 with Python 3.11), this version difference does not affect the validity of the comparative results between the configurations.

The main libraries used included OpenCV 4.10 for image processing and face detection, keras-facenet for the face embedding model, NumPy for vector operations and prototype storage, and psutil and vcgencmd for system resource monitoring (CPU, RAM, and temperature). All experiments were conducted on pre-recorded video to ensure each configuration was tested on identical input, preventing differences in test results solely from changes in pipeline configuration.

3.2 Facial Recognition Pipeline Architecture

The face recognition pipeline used in this study consists of four main stages: (1) frame sampling from the input video, (2) face detection, (3) face embedding extraction, and (4) cosine similarity-based identity matching. The pipeline flow is illustrated in Figure 1. In the frame sampling stage, the video is processed by capturing single frames at fixed time intervals (sample intervals) to reduce computational load without losing important temporal information. The sampled frames are then processed by a face detector to localize the face position in the form of a bounding box. The primary detector used is Haar Cascade from OpenCV, chosen because it is computationally lightweight and has long been the baseline for face detection on limited devices[27].

Each detected face is then cropped and resized to 160x160 pixels as input to the embedding model. The embedding model used is FaceNet from the keras-facenet library, which generates a 512-

<http://sistemasi.ftik.unisi.ac.id>

dimensional vector representation for each face. This embedding vector captures the characteristics of facial features in a high-dimensional space, so faces from the same individual will produce adjacent vectors in that space.

Student identification is performed by comparing the detected embedding vector with a database of registered student prototypes using cosine similarity. A high cosine similarity value (close to 1) indicates a high similarity between faces, while a low value indicates a difference in identity. The pipeline employs an open-set recognition scheme, whereby a student is identified only if their highest similarity value to the database exceeds a certain threshold; otherwise, the face is classified as unknown. Based on initial threshold testing, a value of 0.75 was chosen because it provided the best balance between known accuracy and unknown rejection for the context of class attendance.

To improve identification reliability, the pipeline implements a track-based confirmation mechanism. Each detected face is associated with a cross-frame track using a combination of Intersection over Union (IoU) and the center distance between consecutive bounding boxes, so that the same face across multiple consecutive frames is associated with a consistent track. A student is considered confirmed present only if their identity is successfully matched at least 3 times on the same track.

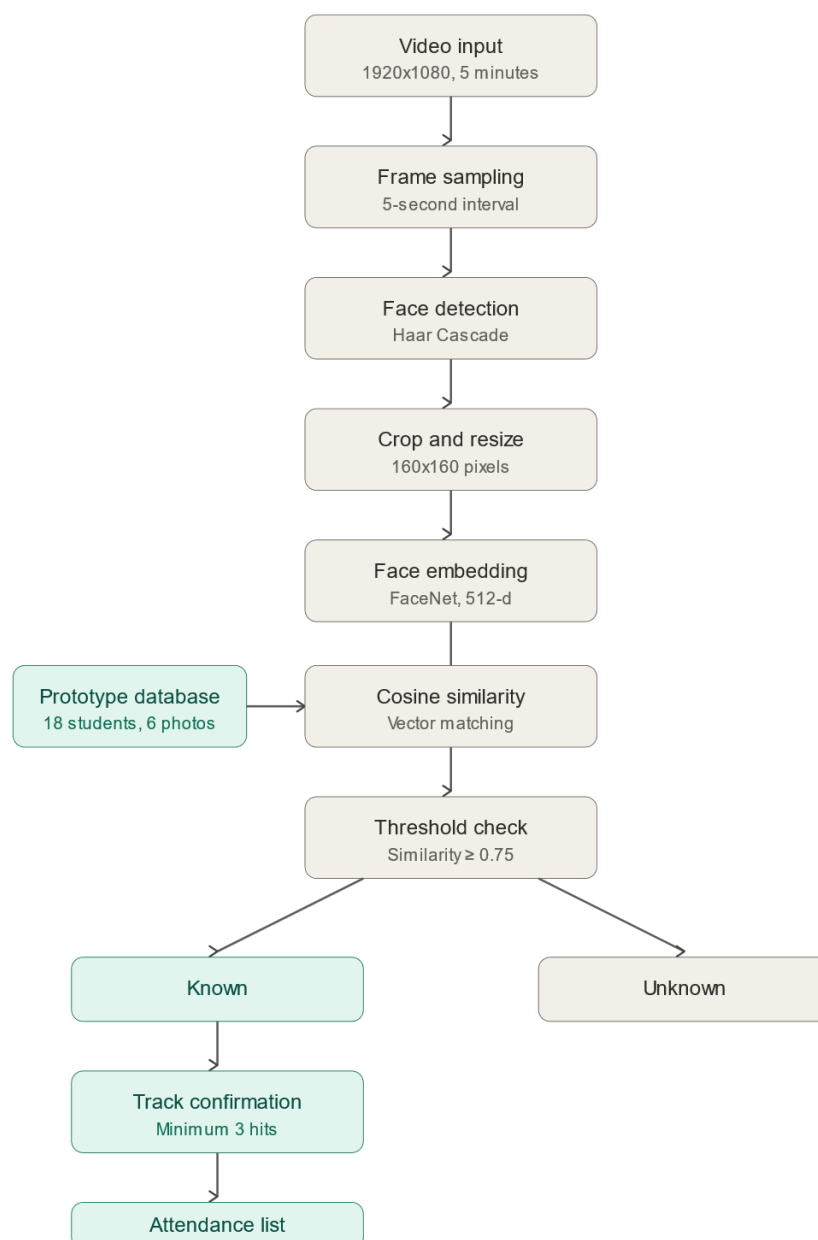


Figure 1 Facial recognition pipeline

3.3 Pipeline Optimization Strategy

Five pipeline optimization strategies were evaluated in this study, each tested separately with a baseline configuration as a comparison. The baseline configuration consisted of frame_scale=1.0, roi=full, sample_interval=5 seconds, a Haar Cascade detector, and prototype rebuilding at each execution (prototype_mode=rebuild).

- 1) **Frame Downscaling:** This strategy lowers the resolution of each captured frame before it is processed by the detector. Reducing the resolution reduces the number of pixels that must be processed, thus theoretically reducing the detector's inference time. The downscaling factors tested were 1.0 (baseline without downscaling), 0.75, and 0.5. The main trade-off with this strategy is that small faces may become difficult to detect at overly aggressive downscaling factors.
- 2) **Static Region of Interest (ROI):** This strategy limits the processing area to a fixed region within the frame that is assumed to always contain the student's face. Since the camera position in the classroom is static, the relevant region can be determined once at the beginning of the experiment without the need for dynamic adaptation. Two ROI configurations were tested, namely ROI q1 with coordinates (x1, y1, x2, y2) = (73, 330, 1836, 996) and ROI q2 with coordinates (76, 336, 1801, 956), both on a frame size of 1920x1080 pixels.
- 3) **Prototype Caching:** This strategy stores previously computed student embedding vectors in a cache file in .npy (NumPy binary) format, so that subsequent executions do not require recomputing the embedding from the reference image. Two modes were compared: rebuild mode, where the prototype is rebuilt each execution, and cache mode, where the prototype is loaded directly from the saved file.
- 4) **Sample Interval Tuning:** This strategy adjusts the time interval between frames processed by the pipeline. A larger interval reduces the number of frames processed, thus reducing total runtime. However, too large an interval risks missing brief student appearances in the video. Three sample interval values were tested: 5, 7, and 10 seconds.
- 5) **Detector Comparison:** This strategy compares two face detectors with different characteristics, which are Haar Cascade as a classical, computationally lightweight detector, and YuNet as a modern CNN-based detector specifically designed for edge devices. The YuNet implementation uses the cv2.FaceDetectorYN module of OpenCV DNN with built-in models available in the OpenCV Zoo repository.

3.4 Dataset and Testing Scenario

Table 1 Characteristics of the test video dataset

Video	Duration	Sampled	Registered Students	Face Detection
	(mm:ss)	Frame	Attended	Total
Class 01	05:16	64	17	683
Class 02	05:27	66	18	649
Class 03	05:40	69	18	560
Total	16:23	199	53	1.892

Testing was conducted using three prerecorded classroom videos captured during lecture sessions, with a resolution of 1920x1080 pixels and an average duration of 5 minutes and 28 seconds per video. The reference database consisted of 18 students, each with six reference photos taken in frontal conditions with normal room lighting. The reference photos were processed through the same detector and embedding model as the main pipeline to generate prototype vectors for each student. As ground truth, manual annotation was performed to record the appearance of each registered student in

each video. The annotation results showed that 17 registered students were present in the first video, 18 students in the second video, and 18 students in the third video, resulting in a total of 53 registered student instances used as the evaluation reference across the entire dataset. The characteristics of each video are summarized in Table 1.

The test scenario was divided into three stages. The first stage was to determine the optimal threshold and temporal parameters (sample interval and confirmation hits) through separate experiments. The second stage was to evaluate each optimization strategy individually using a baseline configuration as a comparison. The third stage was to determine the selected configuration based on a trade-off analysis between accuracy and efficiency.

Each pipeline configuration was executed once using the same prerecorded video inputs and software environment. Consequently, the reported runtime and resource measurements represent descriptive observations from the evaluated executions rather than statistical estimates across repeated trials. The terms average CPU, average RAM, and average temperature refer to averages across monitoring samples collected within a single execution, not averages across repeated experimental runs.

3.5 Evaluation Metrics

The optimal configuration is selected by considering the trade-off between accuracy and efficiency metrics. The best configuration is the one that maintains the highest possible number of confirmed students while significantly reducing runtime and resource consumption compared to the baseline.

- 1) Recognition accuracy metrics include known accuracy (the percentage of registered student faces that were successfully recognized with the correct identity), unknown rejection rate (the percentage of unregistered faces that were successfully rejected by the system as unknown), false accept rate (FAR), false reject rate (FRR), number of confirmed students (the total number of registered students that the system successfully verified across all test videos), number of unknown tracks (the count of facial tracks that were not successfully matched with any identity in the database), and number of face detections (the total count of face detections during the video processing).
- 2) System efficiency metrics include total runtime, average time per sampled frame, effective sampled FPS (the effective pipeline throughput calculated as the number of frames processed divided by the total runtime), average and peak CPU utilization, average and peak RAM usage, and average and peak device temperature.

4 Results and Analysis

This section presents the results of testing five facial recognition pipeline optimization strategies on the Raspberry Pi 5 along with their analysis. Prior to evaluating the optimization strategies, the main pipeline parameters—the cosine similarity threshold, sample interval, and confirm hits—were determined through preliminary experiments to ensure the optimization strategy testing was conducted with a valid parameter configuration.

4.1 Threshold Selection and Temporal Parameters

The threshold, sample interval, and confirmation-hit parameters were selected during the development stage using samples derived from the available classroom video collection. These parameters were fixed before the individual optimization configurations were compared. However, because no fully independent held-out video set was available, the parameter-selection and configuration-evaluation stages shared the same recording domain. Therefore, the reported results should be interpreted as an internal comparative evaluation rather than an unbiased estimate of generalization performance on unseen classroom videos.

The cosine similarity threshold was selected by testing three candidate values in the open-set recognition scheme: 0.65, 0.70, and 0.75. The test used 54 known student face samples and 16 unknown face samples cropped from class videos. The test results are presented in Table 2. The results in Table 2 demonstrate a classic trade-off between the false accept rate (FAR) and false reject rate (FRR) as the threshold increases. At a threshold of 0.65, the system achieved the highest known accuracy of 94.44%, but the FAR was also very high at 43.75%, meaning nearly half of the unknown

<http://sistemasi.ftik.unisi.ac.id>

faces passed as enrolled students. In the context of an attendance system, a high FAR is very dangerous because it can record the attendance of people who are not actually enrolled in class.

Table 2 Cosine similarity threshold test results

Threshold	Known Accuracy (%)	Unknown Rejection (%)	FAR (%)	FRR (%)	Overall Accuracy (%)
0.65	94.44	56.25	43.75	3.70	85.71
0.70	90.74	56.25	43.75	7.41	82.86
0.75	77.78	81.25	18.75	22.22	78.57

At a threshold of 0.70, the FAR remained at 43.75%, while the known accuracy actually decreased to 90.74%. This configuration was dominated by the 0.65 threshold and offered no benefits. Conversely, the 0.75 threshold showed the most balanced profile: the FAR dropped significantly to 18.75%, unknown rejections increased from 56.25% to 81.25%, and the number of misidentifications was zero. The consequence is an FRR increase to 22.22%, but in the context of video processing, the FRR per frame is largely mitigated by the temporal nature of the pipeline. Because each student may appear in multiple sampled frames, the temporal confirmation mechanism provides several opportunities to accumulate the three successful matches required for attendance confirmation.

In the attendance context, false acceptance was considered more costly than false rejection because it could incorrectly associate an unenrolled face with a registered identity. Therefore, threshold selection prioritized lower FAR and the absence of identity misclassification over maximum sample-level overall accuracy. Based on this criterion, a threshold of 0.75 was selected.

Table 3 Results of testing the combination of sample interval and confirm hits

Sample Interval (s)	Confirm Hits	Confirmed Students	Unknown Tracks	Coverage (%)
3	2	48	130	90.57
3	3	47	100	88.68
5	2	48	87	90.57
5	3	47	60	88.68
10	2	45	50	84.91
10	3	36	37	67.92

After the threshold was set at 0.75, the temporal parameters of sample interval and confirm hits were evaluated to determine the combination that provided the best balance between identification coverage and processing efficiency. Six combinations were tested with sample intervals of 3, 5, and 10 seconds, and confirm hits of 2 and 3. The test results are presented in Table 3.

The results in Table 3 reveal two important observations. First, configurations with 3-second and 5-second sampling intervals yield identical coverage for the same number of confirm hits. The pairs (3s, 2 hits) and (5s, 2 hits) both yield 48 confirmed students, and (3s, 3 hits) and (5s, 3 hits) both yield 47 students. This means that increasing the interval from 3 to 5 seconds does not compromise coverage at all. This suggests that sampling every 3 seconds is oversampling for the class attendance scenario; a 5-second interval is sufficient to capture every student's appearance. Second, the coverage decline only becomes significant at the 10-second interval, particularly with 3 confirm hits, which causes coverage to drop to 67.92%. At this interval, the system does not have enough opportunity to capture three confirmations for each student within the limited video duration.

The choice of confirm hits presents an interesting trade-off. At 5-second intervals, moving from 2 to 3 confirm hits resulted in only 1 confirmed student being excluded (from 48 to 47), but the number of unknown tracks dropped significantly from 87 to 60. The decrease in unknown tracks suggests that the stricter criteria removed more unconfirmed tracks. However, without track-level ground truth, these tracks cannot be conclusively attributed to unknown individuals, false detections, or track fragmentation. The (5s, 3 hits) configuration was chosen as the final temporal parameter configuration because it provided higher confirmation reliability at the cost of minimal accuracy, while also being consistent with the conservative philosophy adopted in selecting the 0.75 threshold.

4.2 Pipeline Optimization Strategy Evaluations

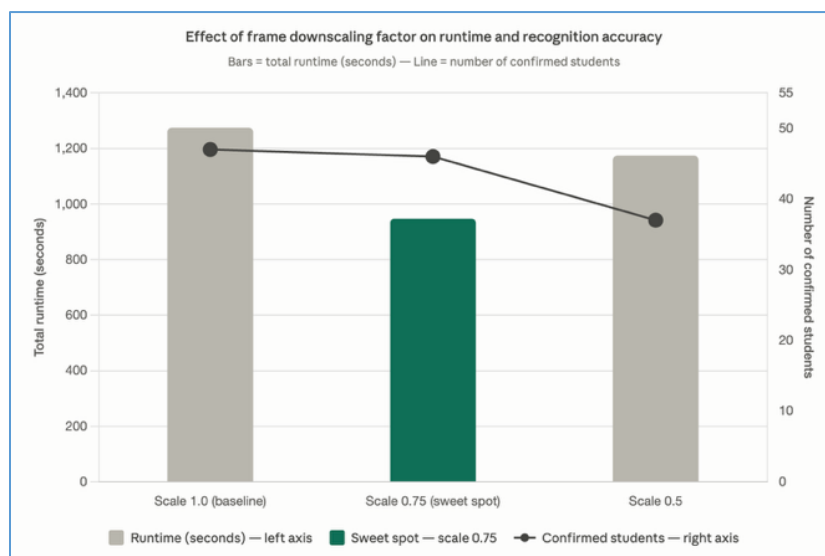


Figure 2 Effect of frame downscaling factor on runtime and recognition accuracy

Figure 2 presents the results of the frame downscaling test in the form of a dual-axis chart, with bars showing the total runtime and lines showing the number of confirmed students. At a scale of 1.0 (baseline), the pipeline produced a runtime of 1,274.83 seconds with 47 confirmed students. Downscaling to 0.75 reduced runtime from 1,274.83 s to 947.72 s, corresponding to a 25.66% reduction. The number of confirmed student-video attendance instances decreased from 47 to 46, equivalent to a 2.13% decrease. This configuration is marked as a sweet spot (teal bar) and was selected as the pipeline optimization component. However, further downscaling to 0.5 actually increased the runtime to 1,174.03 seconds and resulted in the loss of 10 confirmed students. The reason for the non-monotonic runtime at a scale of 0.5 cannot be determined from the aggregate measurements alone. Possible contributing factors include changes in detection behavior, tracking fragmentation, and downstream processing patterns, but per-stage profiling would be required to verify these explanations.

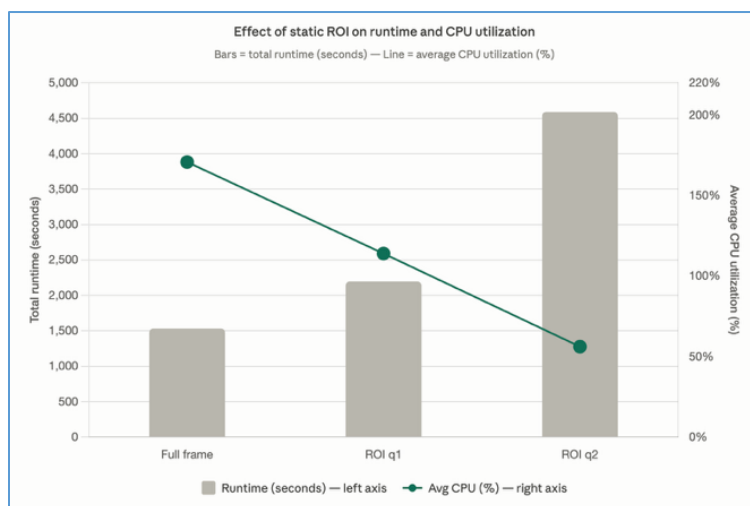


Figure 3 Effect of static ROI on runtime and CPU Utilization

Figure 3 shows the results of the static ROI test, which demonstrates findings contrary to theoretical expectations. Applying ROI q1 increased runtime from 1,532.21 s to 2,195.89 s, corresponding to a 43.32% increase relative to the full-frame configuration. ROI q2 further increased runtime to 4,589.71 s, corresponding to a 199.55% increase. Therefore, neither static ROI configuration produced a runtime improvement in the evaluated setup. The lower CPU utilization accompanied by a longer runtime suggests that the pipeline may have spent more time in serial, waiting, or non-CPU-intensive operations. However, because per-stage profiling was not conducted, the present results cannot determine whether the increase originated from cropping, video access, tracking, format conversion, or another pipeline component.

In rebuild mode, the prototype preparation phase took an average of 30.612 seconds (range: 22.021 to 47.042 seconds) because the pipeline had to recalculate the embedding from the reference photo with each execution. Meanwhile, cache mode, which loads the embedding directly from a saved .npy file, took an average of only 0.003 seconds (3 milliseconds), with very little variance. This approximately 11,000x speedup is dramatic for the preparation phase, but it needs to be understood in a broader context: the preparation phase only accounts for about 2–3% of the total runtime of the three-video processing pipeline. Therefore, prototype caching is more appropriately positioned as a deployment and system startup time optimization than an inference runtime optimization, and is therefore not included as a key component of the selected pipeline configuration, although it may be enabled in future implementations as a startup-time optimization.

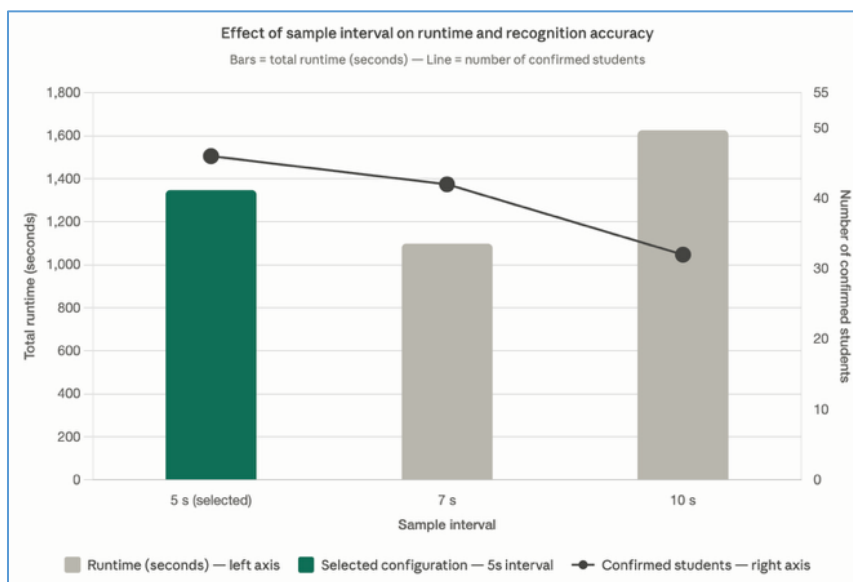


Figure 4 Effect of sample interval on runtime and confirmed attendance instances

Figure 4 shows the results of a sample interval tuning test with a pattern similar to frame downscaling: neither monotonic nor intuitive. At a 5-second interval (selected configuration, teal bar), the pipeline produced a runtime of 1,348.06 seconds with 46 confirmed students. A 7-second interval reduced the runtime to 1,099.16 seconds (an 18.46% decrease), but at the expense of 4 confirmed students (to 42). Conversely, a 10-second interval increased the runtime to 1,626.14 seconds with the loss of 14 confirmed students (a drop to 32). The longer runtime observed at the 10-second interval may be associated with video-seeking and decoding behavior. However, this explanation was not directly verified through decoding-time or per-stage profiling and should therefore be treated as a hypothesis. While the 7-second interval provides a speedup, losing 4 students is considered too large for the context of an attendance system. Therefore, the 5 second interval was retained as the chosen configuration, consistent with the conservative philosophy applied to the previous threshold and confirm hits selection.

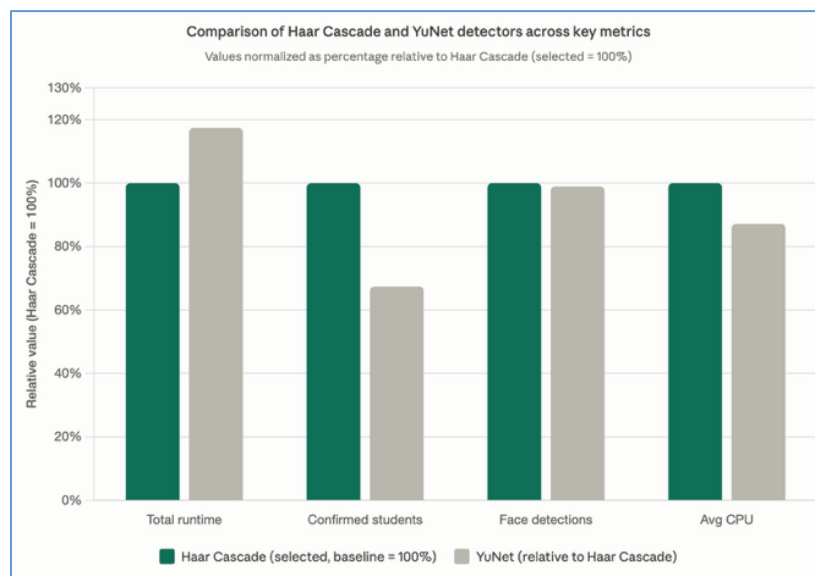


Figure 5 Comparison of Haar Cascade and YuNet detectors

Figure 5 presents a comparison of the Haar Cascade and YuNet detectors in the form of a grouped bar chart. Haar Cascade and YuNet produced a similar total number of face detections, with 1,528 and 1,512 detections, respectively. However, YuNet produced only 31 confirmed attendance instances compared with 46 for Haar Cascade. This indicates that detection count alone does not explain the difference in downstream confirmation performance. The present experiment did not evaluate bounding-box alignment, crop quality, genuine-similarity distributions, or tracking fragmentation. Therefore, the specific cause of YuNet's lower confirmation performance cannot be determined from the current results. Based on the observed end-to-end results, Haar Cascade was retained for the evaluated FaceNet pipeline. This finding should not be interpreted as evidence that Haar Cascade generally outperforms YuNet in other datasets or preprocessing pipelines.

4.3 Selected Pipeline Configuration and Trade-off Analysis

Based on the evaluation of five optimization strategies, the selected configuration for further evaluation on Raspberry Pi 5 was set with the following parameters: threshold cosine similarity 0.75, sample interval 5 seconds, confirm hits 3, frame scale 0.75, Haar Cascade detector, and prototype mode rebuild. This configuration incorporates the frame downscaling strategy as the main optimization component, while other evaluated strategies (static ROI, prototype caching for pipeline runtime, sample interval above 5 seconds, and YuNet detector) were not adopted because they did not provide significant net benefits on the test setup.

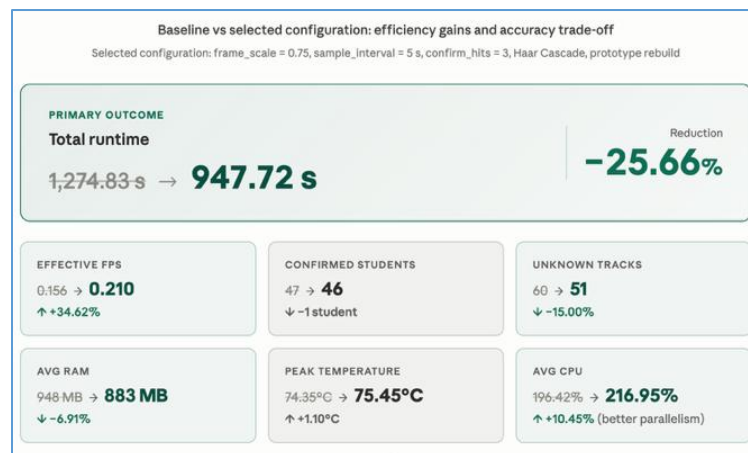


Figure 6 Comparison of baseline and selected configuration

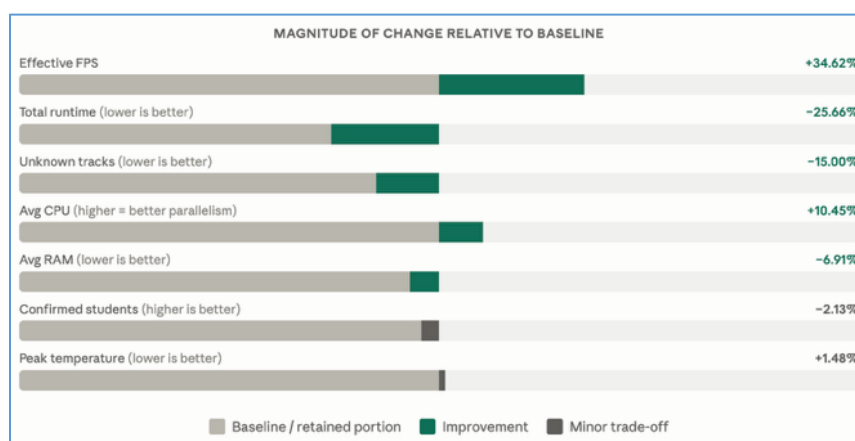


Figure 7 Magnitude of change relative to baseline

Figure 6 and 7 summarize the performance profile of the selected configuration compared to the baseline in three complementary visual sections. The selected configuration reduced runtime by 25.66%, increased effective sampled-frame throughput by 34.62%, and reduced average RAM usage by 6.86%. These improvements were accompanied by several trade-offs: average CPU utilization increased by 10.45%, peak temperature increased by 1.10°C, and confirmed attendance instances decreased from 47 to 46.

4.4 Research Limitations

This study has several limitations that should be considered when interpreting the reported findings. First, each pipeline configuration was evaluated using a single execution without independent experimental repetitions. Runtime and resource utilization may therefore have been influenced by temporary system conditions, including the initial device temperature, operating-system activity, container state, and system cache. Consequently, the reported differences, including the 25.66% runtime reduction achieved by frame downscaling at a factor of 0.75, represent descriptive observations from the evaluated executions rather than statistically validated performance gains. The reported average CPU utilization, RAM usage, and device temperature refer to averages across monitoring samples collected within a single execution and should not be interpreted as averages across repeated experimental runs. Repeated trials with randomized execution order and the reporting of mean and standard deviation are required to assess the stability of these results.

Second, the evaluation dataset was limited to three prerecorded classroom videos with a total duration of 16 min 23 s, recorded at a resolution of 1920 × 1080 pixels using a static camera under relatively similar classroom conditions. The enrollment database contained only 18 students, while the evaluation reference consisted of 53 student-video attendance instances across the three videos. Therefore, the findings may not generalize to larger student populations, longer recording durations,

<http://sistemasi.ftik.unisi.ac.id>

different camera positions, lower-resolution video, substantial illumination changes, severe facial occlusion, or more crowded classroom environments. Changes in these conditions may affect face size, detection reliability, similarity-score distributions, tracking stability, and the optimal recognition threshold.

Third, threshold and temporal-parameter selection were performed during the development stage using data from the same limited classroom-video domain employed in the configuration comparison. The cosine similarity threshold, sample interval, and confirmation-hit requirement were fixed before the individual optimization strategies were compared; however, no fully independent held-out classroom-video set was available for final validation. Consequently, the reported recognition coverage and selected configuration may be influenced by the characteristics of the available recordings and may provide an optimistic estimate of performance on unseen classroom environments. The results should therefore be interpreted as an internal comparative evaluation of pipeline configurations rather than as an unbiased estimate of generalization performance.

Fourth, several counter-intuitive results could not be explained conclusively using the collected aggregate measurements. These include the runtime increase at a frame scale of 0.5, the 43.32% and 199.55% runtime increases produced by ROI q1 and ROI q2, respectively, the longer runtime observed at a 10-second sampling interval, and the lower attendance-confirmation performance obtained using the YuNet detector. The current experiment did not measure execution time separately for video access, frame preprocessing, face detection, embedding extraction, tracking, identity matching, and monitoring operations. It also did not compare bounding-box alignment, crop quality, genuine-similarity distributions, or tracking fragmentation between Haar Cascade and YuNet. Therefore, explanations involving video-seeking overhead, cropping operations, detector bounding-box characteristics, or tracking behavior should be treated as hypotheses rather than established causal findings. Per-stage profiling and detector-specific crop analysis are required to determine the sources of these performance differences.

Finally, this study evaluated identity recognition but did not incorporate presentation attack detection or evaluate resistance to spoofing attempts using printed photographs, recorded videos, masks, or digital displays. Accordingly, the proposed pipeline cannot be considered an anti-spoofing or fraud-prevention system. Its potential contribution is limited to supporting identity-based attendance verification and potentially reducing simple proxy attendance involving identity substitution. In addition, the study focused on pipeline-level optimization and did not investigate model-level techniques such as quantization, pruning, knowledge distillation, or hardware acceleration. These limitations indicate that the selected configuration should be regarded as a candidate for further validation on the Raspberry Pi 5 rather than as a deployment-ready attendance solution.

5 Conclusion

This study conducted a descriptive single-run evaluation of five pipeline-level optimization strategies for an edge-based facial recognition attendance system on a Raspberry Pi 5. The evaluated strategies included frame downscaling, static region of interest (ROI), prototype caching, sample interval tuning, and face detector comparison. The pipeline employed FaceNet for facial embedding, cosine similarity for open-set identity matching, and track-based temporal confirmation. The evaluation used three prerecorded classroom videos with a resolution of 1920×1080 pixels, a total duration of 16 min 23 s, and an enrollment database containing 18 students. Across the three videos, 53 student-video attendance instances were used as the reference for attendance confirmation.

Among the evaluated strategies, frame downscaling at a factor of 0.75 provided the most favorable observed trade-off between recognition coverage and computational efficiency. Compared with the baseline, this configuration reduced runtime from 1,274.83 s to 947.72 s, corresponding to a 25.66% reduction, while the number of confirmed student-video attendance instances decreased only slightly from 47 to 46. Effective sampled-frame throughput increased by 34.62%, and average RAM usage decreased by 6.86%. These improvements were accompanied by an increase of 10.45% in average CPU utilization and an increase of 1.10°C in peak temperature. Therefore, higher CPU utilization was treated as a resource trade-off rather than as a performance improvement.

The remaining optimization strategies produced more varied results. Prototype caching substantially reduced prototype preparation time and may therefore be useful as a system-startup optimization, although its contribution to the end-to-end video-processing runtime was limited. Static ROI did not improve runtime in the evaluated setup; ROI q1 and ROI q2 increased runtime by 43.32% and 199.55%, respectively, relative to full-frame processing. Increasing the sample interval beyond 5 s reduced attendance-confirmation coverage, while YuNet produced fewer confirmed attendance instances than Haar Cascade despite generating a similar number of face detections. Because per-stage profiling, bounding-box analysis, and genuine-similarity distribution analysis were not conducted, the specific causes of these counter-intuitive results cannot be determined from the current measurements. Based on the observed end-to-end comparison, the selected pipeline configuration used a cosine similarity threshold of 0.75, a 5-second sample interval, three confirmation hits, a frame scale of 0.75, and the Haar Cascade detector. Prototype rebuilding was retained during the comparative experiments for consistency, although prototype caching may be enabled to reduce preparation time in an implementation setting.

Overall, the findings indicate that the selected configuration was computationally feasible within the evaluated prerecorded-video scenario on a Raspberry Pi 5. However, the results should not be interpreted as evidence of general deployment readiness because each configuration was executed only once, the dataset was limited to three videos and 18 enrolled students, and no fully independent held-out classroom-video set was used for final validation. In addition, the system did not incorporate presentation attack detection and therefore cannot be considered resistant to spoofing attacks involving photographs, recorded videos, or digital displays. Future work should conduct repeated trials with randomized execution order and report mean and standard deviation, evaluate the pipeline using independent classroom recordings and larger enrollment databases, perform per-stage profiling to identify computational bottlenecks, analyze detector-specific crop and embedding quality, integrate presentation attack detection, and investigate model-level optimization techniques such as quantization, pruning, and knowledge distillation. Consequently, the selected configuration should be regarded as a candidate for further deployment validation rather than a deployment-ready attendance solution.

Acknowledgement

The authors would like to express their sincere gratitude to the Direktorat Penelitian dan Pengabdian Masyarakat Universitas Islam Riau (DPPM UIR) for providing financial support, which enabled this research to be conducted successfully.

References

- [1] R. Wandri, M. Fadhilla, D. F. Qurniawan, and E. D. Saputra, "Implementation of a Face Recognition API for an Automated Web-based Attendance System using Extreme Programming," *Sistemasi: Jurnal Sistem Informasi*, Vol. 15, No. 6, Jun. 2026, [Online]. Available: <http://sistemasi.ftik.unisi.ac.id>
- [2] R. Rahmayani, S. Saniman, and T. Tugiono, "Perancangan Sistem Sidik Jari Absensi Siswa SMP dengan menggunakan Node MCU yang Terhubung dengan Telegram," *Jurnal Sistem Komputer Triguna Dharma (JURSIK TGD)*, Vol. 2, No. 2, pp. 132–138, Mar. 2023, DOI: 10.53513/jursik.v2i2.7193.
- [3] Asep Mahpudin and Agam Hamdani, "Perancangan Sistem Informasi Absensi Sekolah berbasis Web," *ICT Learning*, Vol. 6, No. 2, Dec. 2022, DOI: 10.33222/ictlearning.v6i2.2766.
- [4] H. Harizahayu, F. Friendly, R. W. Sembiring, and P. H. Putra, "Perancangan dan Penerapan Sistem Pencatatan Kehadiran Siswa pada SMK Yayasan Pendidikan Mulia Kecamatan Medan Selayang Sumatera Utara," *SWARNA: Jurnal Pengabdian Kepada Masyarakat*, Vol. 3, No. 2, pp. 134–140, Feb. 2024, DOI: 10.55681/swarna.v3i2.997.
- [5] J. Budiasto, H. Jayawardana, and F. A. K. Dewi, "Sistem Informasi Pencatatan Absensi Siswa berbasis Website pada SMA Negeri 1 Kurik," *Musamus Journal of Technology & Information*, Vol. 5, No. 02, pp. 066–071, Apr. 2023, DOI: 10.35724/mjti.v5i02.5384.

- [6] M. Fadhillah, R. Wandri, A. Hanafiah, P. R. Setiawan, Y. Arta, and S. Daulay, "Analisis Performa Algoritma *Machine Learning* untuk Identifikasi Depresi pada Mahasiswa," *Journal of Informatics Management and Information Technology*, Vol. 5, No. 1, 2025.
- [7] X. Li, S. Lai, and X. Qian, "DBCFace: Towards Pure Convolutional Neural Network Face Detection," *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 32, No. 4, pp. 1792–1804, Apr. 2022, DOI: 10.1109/TCSVT.2021.3082635.
- [8] C. Oinar, B. M. Le, and S. S. Woo, "KappaFace: Adaptive Additive Angular Margin Loss for Deep Face Recognition," *IEEE Access*, Vol. 11, pp. 137138–137150, 2023, DOI: 10.1109/ACCESS.2023.3338648.
- [9] G. Vardakis, G. Tsamis, E. Koutsaki, K. Haridimos, and N. Papadakis, "Smart Home: Deep Learning as a Method for Machine Learning in Recognition of Face, Silhouette and Human Activity in the Service of a Safe Home," *Electronics (Switzerland)*, Vol. 11, No. 10, May 2022, DOI: 10.3390/electronics11101622.
- [10] G. Rajeshkumar et al., "Smart Office Automation via Faster R-CNN based Face Recognition and Internet of Things," *Measurement: Sensors*, Vol. 27, Jun. 2023, DOI: 10.1016/j.measen.2023.100719.
- [11] P. P. Oroceo, J. I. Kim, E. M. F. Caliwag, S. H. Kim, and W. Lim, "Optimizing Face Recognition Inference with a Collaborative Edge–Cloud Network," *Sensors*, Vol. 22, No. 21, Nov. 2022, DOI: 10.3390/s22218371.
- [12] H. Hua, Y. Li, T. Wang, N. Dong, W. Li, and J. Cao, "Edge Computing with Artificial Intelligence: A Machine Learning Perspective," *ACM Comput. Surv.*, Vol. 55, No. 9, Sep. 2023, DOI: 10.1145/3555802.
- [13] A. Koubaa, A. Ammar, A. Kanhouch, and Y. Alhabashi, "Cloud Versus Edge Deployment Strategies of Real-Time Face Recognition Inference," *IEEE Trans. Netw. SCI. Eng.*, Vol. 9, No. 1, pp. 143–160, 2022, DOI: 10.1109/TNSE.2021.3055835.
- [14] S. S. Khan, D. Sengupta, A. Ghosh, and A. Chaudhuri, "MTCNN++: A CNN-based Face Detection Algorithm Inspired by MTCNN," *Visual Computer*, Vol. 40, No. 2, pp. 899–917, Feb. 2024, DOI: 10.1007/s00371-023-02822-0.
- [15] S. Sony Priya and R. I. Minu, "Augmenting Face Detection in Extremely Low-Light CCTV Footage using the EDCE Enhancement Model," *Traitement du Signal*, Vol. 40, No. 6, pp. 2741–2750, Dec. 2023, DOI: 10.18280/ts.400634.
- [16] M. A. Hasan, "Facial Human Emotion Recognition by using YOLO Faces Detection Algorithm," *JOINCS (Journal of Informatics, Network, and Computer Science)*, Vol. 6, No. 2, pp. 32–38, Nov. 2023, DOI: 10.21070/joincs.v6i2.1629.
- [17] A. George, C. Ecabert, H. O. Shahreza, K. Kotwal, and S. Marcel, "EdgeFace: Efficient Face Recognition Model for Edge Devices," *IEEE Trans. Biom. Behav. Identity SCI.*, Vol. 6, No. 2, 2024, DOI: 10.1109/TBIOM.2024.3352164.
- [18] W. Wu, H. Peng, and S. Yu, "YuNet: A Tiny Millisecond-Level Face Detector," *Machine Intelligence Research*, Vol. 20, No. 5, 2023, DOI: 10.1007/s11633-023-1423-y.
- [19] S. Minakova and T. Stefanov, "Memory-Throughput Trade-off for CNN-based Applications at the Edge," *ACM Transact. Des. Autom. Electron. Syst.*, Vol. 28, No. 1, Dec. 2022, DOI: 10.1145/3527457.
- [20] Q. Qi, Y. Lu, J. Li, J. Wang, H. Sun, and J. Liao, "Learning Low Resource Consumption CNN through Pruning and Quantization," *IEEE Trans. Emerg. Top. Comput.*, Vol. 10, No. 2, pp. 886–903, 2022, DOI: 10.1109/TETC.2021.3050770.
- [21] S. L. Chu, C. F. Chen, and Y. C. Zheng, "CFSM: A Novel Frame Analyzing Mechanism for Real-Time Face Recognition System on the Embedded System," *Multimed. Tools Appl.*, Vol. 81, No. 2, 2022, DOI: 10.1007/s11042-021-11599-0.
- [22] M. Parhi, A. Roul, B. Ghosh, and A. Pati, "IOATS: An Intelligent Online Attendance Tracking System based on Facial Recognition and Edge Computing," *Original Research Paper International Journal of Intelligent Systems and Applications in Engineering IJISAE*, Vol. 2022, No. 2, pp. 252–259, 2022, DOI: 10.1039/b000000x.
- [23] A. Agus Kurniasari, I. G. Wiryawan, T. Rizaldi, P. S. D. Puspitasari, D. M. P. Ernanta, and S. P. Sari, "Intelligence Attendance Monitoring System using Real-Time Face Recognition and Raspberry Pi 'Intelligence Attendance Monitoring System using Real-Time Face Recognition

- and Raspberry Pi,” *Matrix: Jurnal Manajemen Teknologi dan Informatika*, Vol. 15, No. 2, pp. 102–113, 2025, DOI: 10.31940/matrix.v15i2.102-113.
- [24] J. Jeong, B. Kim, J. Yu, and Y. Yoo, “EResFD: Rediscovery of the Effectiveness of Standard Convolution for Lightweight Face Detection,” in *Proceedings - 2024 IEEE Winter Conference on Applications of Computer Vision*, WACV 2024, 2024. DOI: 10.1109/WACV57701.2024.00103.
- [25] K. Gkrispanis, N. Gkalelis, and V. Mezaris, “Filter-Pruning of Lightweight Face Detectors using a Geometric Median Criterion,” in *Proceedings - 2024 IEEE Winter Conference on Applications of Computer Vision Workshops*, WACVW 2024, 2024. DOI: 10.1109/WACVW60836.2024.00037.
- [26] A. Baobaid and M. Meribout, “Edge-GPU based Face Tracking for Face Detection and Recognition Acceleration,” *IEEE Internet Things J.*, 2026, DOI: 10.1109/JIOT.2026.3701825.
- [27] O. A. Naser, S. Mumtazah, K. Samsudin, M. Hanafi, S. M. B. Shafie, and N. Z. Zamri, “Comparative Analysis of MTCNN and Haar Cascades for Face Detection in Images with Variation in Yaw Poses and Facial Occlusions,” *Journal of Communications Software and Systems*, Vol. 21, No. 1, 2025, DOI: 10.24138/jcomss-2024-0084.