
**National University of Sciences and Technology (NUST)**

**School of Electrical Engineering and Computer Science**

# Real-Time Driver Drowsiness Detection

## Using NVIDIA Jetson Nano and USB Webcam

**End Semester Project Report**

**Course: CS-477 (Computer Vision)**

## Submitted By

**Mamona Sadaf** CMS ID: 418762
Documentation & Project Life Cycle

**Menahil Ahsan** CMS ID: 404825
Simulation & Algorithm Design

**Sarah Omer** CMS ID: 425337
Embedded System Implementation

**Fall Semester 2025**

***Abstract*—Road traffic accidents caused by driver fatigue represent a significant global safety challenge, necessitating the development of non-invasive, real-time monitoring systems[cite: 487, 494]. This project implements an embedded deep learning solution for driver drowsiness detection optimized for the NVIDIA Jetson Nano 4GB platform[cite: 485, 489]. While initial development utilized MediaPipe FaceMesh for high-fidelity 468-landmark tracking, empirical testing on the Jetson Nano revealed significant computational bottlenecks, unstable frame rates, and thermal-induced system halts[cite: 443, 444, 448].**

**To achieve real-time feasibility on resource-constrained hardware, the architecture was redesigned into a hybrid pipeline: employing Haar Cascade classifiers for efficient face and eye localization followed by a custom, lightweight binary Convolutional Neural Network (NanoEyeCNN) for eye-state classification between "Open" and "Closed" states[cite: 454, 461, 462, 465]. The CNN, trained on the MRL Eye Dataset, achieved an overall classification accuracy of 98.62%, with a 98.61% F1-score for the closed-eye class, ensuring robust detection of microsleep events[cite: 455, 470]. The system further incorporates Mouth Aspect Ratio (MAR) analysis for yawning detection and temporal logic to distinguish between natural blinks and prolonged fatigue indicators[cite: 463, 464]. Deployment challenges, including CSI camera instability and limited eMMC storage, were mitigated by transitioning to a USB-based acquisition system and implementing a boot-from-USB mechanism[cite: 452, 457, 470]. Final implementation on the Jetson Nano demonstrated stable performance, maintaining a consistent 15–20 FPS with a total CPU load of approximately 45–55% and efficient memory utilization between 1.8–2.3 GB[cite: 469, 472].**

***Index Terms*—Driver fatigue detection, NVIDIA Jetson Nano, Embedded Deep Learning, Computer Vision, Haar Cascade, Real-time Systems, Convolutional Neural Network[cite: 492].**

## ACKNOWLEDGEMENTS

The authors would like to express their deepest gratitude to **Dr. Tauseef Ur Rehman** for his invaluable supervision, technical guidance, and support throughout the course of this semester project[cite: 5, 486]. His expertise in computer vision and embedded systems provided the critical foundation needed to overcome architectural challenges and system instabilities encountered during the development phase.

We extend our thanks to the teaching staff at NUST SEECS, specifically **Miss Tehniyyat Siddique** (Lab Engineer) for providing essential hardware resources, Jetson Nano kits, and research facilities[cite: 131, 486]. We also acknowledge **Zahid Hassan** (Teaching Assistant) for his consistent technical support and assistance in debugging deployment issues[cite: 131, 486]. Special recognition is given to the "Jetson Nano Warriors" group and the open-source communities behind OpenCV, PyTorch, and MediaPipe for their tools and frameworks.

## CONTENTS

## List of Figures



## List of Tables



# I. Introduction and Background

## A. Problem Statement and Motivation

The escalating frequency of road traffic accidents remains a critical global safety concern and a major public health challenge. Statistical evidence and comprehensive surveys indicate

that driver fatigue is a primary human factor, contributing to approximately 20

The motivation for this research stems from the necessity of bridging the gap between high-accuracy laboratory methods and real-world in-vehicle deployability. In modern automotive safety, the objective is to detect the transition from alertness to drowsiness through behavioral cues, such as frequent yawning and the duration of eye closure, known as microsleeps. Traditional methods of monitoring driver state often involve vehicle-based metrics like lane deviation or erratic steering, yet these indicators frequently appear only after the driver has already reached a dangerous state of fatigue. By contrast, computer vision techniques allow for the proactive identification of facial biomarkers that precede actual loss of vehicle control. The advent of powerful, low-power edge computing devices, specifically the NVIDIA Jetson Nano, provides a technical foundation to run complex deep learning models directly within the vehicle environment. However, implementing such systems requires overcoming significant hurdles, including high computational demands, varying cabin illumination, and the need for stable hardware integration. This project is motivated by these challenges, seeking to develop a robust, non-intrusive solution that maintains high detection accuracy while adhering to the strict resource constraints of embedded hardware. Ultimately, the goal is to create a reliable system that empowers drivers with a second layer of safety, ensuring that the first sign of fatigue is met with an immediate and effective alert.

### *B. Project Domain and Refined Sub-topic*

The overarching domain of this project is Robust Face Detection for Facial Biomarkers of Health, a field that sits at the intersection of computer vision, human-computer interaction, and medical informatics. This domain focuses on the extraction of physiological and behavioral data from facial features to assess a subject's state of well-being, alertness, or cognitive load. In the context of automotive safety, this involves analyzing facial markers as proxies for neurological fatigue, which is often characterized by decreased eyelid movement speed, prolonged eye closure duration, and repetitive yawning.

Within this broad domain, the project is narrowed to the refined sub-topic of Real-time Eye-Region Fatigue Detection on Edge-AI Platforms. While general face detection identifies the presence of a person, this sub-topic requires deep granular analysis of the ocular region to distinguish between voluntary actions (conscious blinking) and involuntary indicators of fatigue (microsleeps). This specialization is particularly challenging because it necessitates high-speed processing to capture rapid eyelid movements while operating within the strict power and memory limits of embedded hardware like the NVIDIA Jetson Nano.

The technical challenge lies in "Robustness," which refers to the system's ability to maintain high detection accuracy despite common real-world disturbances such as varying cabin illumination, the driver wearing prescription glasses, or head rotations that partially occlude the eyes. By refining the focus to the eye region, the system can utilize high-precision deep learning models to classify eye states—"Open" versus "Closed"—with a high degree of confidence, which is the most reliable precursor to a drowsiness-induced accident.

The motivation for choosing this sub-topic is the realization that while full facial landmarking is computationally expensive, targeted ocular analysis provides a specialized, efficient pathway to life-saving alerts. This approach moves beyond general computer vision into the realm of safety-critical embedded systems, where every millisecond of latency saved in detecting a closed eye can equate to several meters of stopping distance on the road. Consequently, this project serves as a practical implementation of edge-AI, demonstrating how specialized sub-topics in machine learning can be optimized to solve complex, real-world health and safety problems in the automotive sector

### *C. Summary of Related Work*

The academic and industrial landscape of vision-based fatigue detection has undergone a rapid transformation, shifting from basic mathematical ratios to complex, multi-layered neural architectures. This progression is generally categorized into five primary technical modalities, each offering different trade-offs between precision and computational efficiency.

*1) Handcrafted Feature-Based and Geometric Approaches:* Early research in the field predominantly relied on traditional computer vision techniques, where researchers manually defined features to be extracted from images. Techniques such as Histogram of Oriented Gradients (HOG) and Local Binary Patterns (LBP) were instrumental in calculating geometric facial measurements. These measurements were often synthesized into metrics like the Eye Aspect Ratio (EAR) and Mouth Aspect Ratio (MAR) to provide a mathematical assessment of a driver's alertness. While these approaches are computationally lightweight, they frequently suffer from a lack of robustness; their accuracy tends to degrade sharply under varying illumination, significant head rotations, or when the subject wears prescription eyewear.

*2) Deep Learning-Based Classification Models:* The emergence of Convolutional Neural Networks (CNNs) revolutionized the project domain by eliminating the need for manual feature engineering. Modern deep learning architectures can automatically learn discriminative textures and patterns associated with drowsiness directly from raw pixel data. For example, specialized architectures like the Driver Drowsiness Artificial Intelligence (DD-AI) network have achieved state-of-the-art accuracy levels of up to 99.88%. These models are trained on massive datasets, such as the NITYMED or MRL Eye Dataset, allowing them to generalize across diverse facial structures and lighting conditions that would typically cause traditional geometric methods to fail.

*3) Landmark-Based and Explainable AI Approaches:* More recent frameworks, such as MediaPipe FaceMesh, provide a different path by offering high-density tracking using 468 facial landmarks in a 3D coordinate space. This level of

granularity enables the detection of extremely subtle eyelid movements and micro-expressions of fatigue. Furthermore, landmark-based systems are often favored in "Explainable AI" contexts because the reasoning behind a "drowsy" classification—such as a specific change in the distance between the upper and lower eyelid—can be easily visualized and understood by human operators. However, the primary drawback of these systems is their high computational cost, which often leads to performance bottlenecks when deployed on standard embedded processors.

*4) Object Detection and Region-Based Frameworks:* The evolution of drowsiness detection has moved toward a more contextual understanding of the driver's environment, acknowledging that facial analysis alone can be prone to errors in a dynamic vehicle cabin. Researchers have increasingly integrated robust object detection models, most notably the YOLO (You Only Look Once) framework, to complement traditional facial feature analysis. Systems such as RealD3 represent a significant milestone in this trend by employing a dual-modality approach that concurrently monitors the driver's face and the presence of external objects that may obscure or mimic signs of fatigue.

This integration is specifically designed to detect secondary objects or behavioral actions that often lead to false alarms in unimodal vision systems. For example, a standard eye-tracking algorithm might trigger a "drowsy" alert if it cannot detect the pupils; however, a integrated YOLO module can identify that the driver is simply wearing dark sunglasses, allowing the system to intelligently adjust its logic and avoid an erroneous alert. Similarly, YOLO can detect hands placed over the mouth, which helps the system distinguish between a genuine yawn and a driver simply adjusting their position or speaking.

By analyzing these region-based features alongside Eye Aspect Ratio (EAR) and Mouth Aspect Ratio (MAR), these frameworks achieve a more holistic and accurate classification of the driver's state. This multi-layered verification process is essential for building user trust, as it ensures that alerts are only generated when true physiological fatigue is confirmed, even in complex environments where the driver's face might be partially occluded. Ultimately, the fusion of object detection and facial analysis bridges the gap between simple landmark tracking and high-reliability automotive safety systems.

*5) Edge Deployment, Federated Learning, and Mobile Systems:* Current trends focus on deploying these models on edge devices, such as the NVIDIA Jetson Nano. Real-time performance is achieved using hardware-specific optimizations, including model compression, quantization, and CUDA-accelerated libraries like TensorRT, to overcome limited memory and thermal constraints.

*6) Motivation for the Present Work:* The primary catalyst for this research is the critical necessity of bridging the gap between high-precision computer vision frameworks and the rigid operational constraints of embedded automotive hardware. While modern high-fidelity landmarking systems, most notably MediaPipe FaceMesh, offer unparalleled tracking of 468 distinct facial points, our extensive empirical stress testing on the NVIDIA Jetson Nano 4GB platform revealed significant barriers to real-world feasibility. During sustained execution, the intensive processing load required for dense 3D landmarking frequently overwhelmed the Jetson's ARM Cortex-A57 CPU, leading to thermal-induced system halts, frame freezing, and inconsistent inference patterns. Such instability is unacceptable in a safety-critical context where a fraction of a second in latency can determine the outcome of a fatigue-related event. This project is thus motivated by the urgent need for a stable, hybrid detection pipeline that replaces resource-heavy landmarking with a streamlined combination of classical computer vision and specialized deep learning. By utilizing Haar Cascade classifiers for efficient localization and a custom, lightweight binary Convolutional Neural Network (NanoEyeCNN) for state classification, this work achieves a reliable 15–20 FPS performance while maintaining a 98.62% classification accuracy.

Furthermore, the project addresses the persistent hardware integration challenges that often plague embedded vision deployments. Our initial development phase highlighted the inherent instability of the Raspberry Pi Camera Module v2 on the Jetson's CSI interface, which suffered from frequent driver failures and terminal crashes. The decision to pivot toward a USB-based acquisition system was motivated by the need for native OpenCV compatibility and consistent frame delivery, ensuring that the software stack remains robust during prolonged operation. By optimizing the entire lifecycle—from the boot-from-USB storage configuration to the CUDA-accelerated OpenCV build—this work demonstrates how engineering pivots in both algorithm selection and hardware integration can produce a stable, production-ready solution that thrives within the actual constraints of modern vehicle environments.

### *D. Scope and Objectives*

The comprehensive scope of this project encompasses the end-to-end design, implementation, and systematic optimization of an embedded real-time driver drowsiness detection system specifically tailored for high-stakes automotive environments. A fundamental objective of this work is the complete architectural redesign of the detection pipeline, transitioning away from high-load 3D landmarking frameworks such as MediaPipe, which proved computationally unsustainable on edge hardware.

In its place, the project establishes a lightweight hybrid architecture that utilizes Haar Cascade classifiers for rapid facial localization and a custom binary Convolutional Neural Network (CNN) for high-accuracy state determination, ensuring a stable and resilient execution environment. Parallel to the algorithmic development, the scope includes rigorous hardware integration and system-level configuration to overcome the inherent limitations of the NVIDIA Jetson Nano 4GB platform. This involves the implementation of a stable USB-based frame acquisition system to replace unreliable CSI in-

terfaces and the configuration of a boot-from-USB mechanism to expand storage capacity for heavy neural network libraries.

A primary focus is placed on performance optimization, leveraging GPU-accelerated libraries such as CUDA-enabled OpenCV and optimized PyTorch wheels to maintain a consistent 15–20 frames per second (FPS) within the restricted 4 GB RAM and 10 W power envelope of the Jetson Nano. To ensure practical reliability and minimize driver irritation, the project objectives include the development of sophisticated temporal logic modules. These modules are designed to differentiate between natural, reflexive blinking and hazardous microsleep events by analyzing the duration of eye closure, thereby reducing false alarms.

Ultimately, the system aims to maintain a high classification accuracy of 98.62% for fatigue events, providing a robust second layer of safety that bridges the gap between theoretical computer vision and a production-ready, real-time embedded solution.

## II. Literature Review

Driver fatigue detection has been extensively studied across multiple sensing modalities, from physiological signals to vision-based methods. The literature demonstrates a clear trend from traditional handcrafted approaches to deep learning, multimodal fusion, and real-time edge deployment.

### A. Physiological Signal-Based Approaches

Electroencephalography (EEG) is widely regarded as the gold standard for drowsiness detection due to its direct measurement of brain activity. Fouad [1] developed a robust EEG-based system that applies multiple machine learning algorithms, including Naive Bayes, Support Vector Machines, K-Nearest Neighbor, and Random Forest, achieving up to 100% classification accuracy under controlled conditions. Similarly, Lian et al. [2] proposed a hybrid EEG-eye tracking system, employing cross-modal predictive alignment and attention modules to enhance feature fusion, achieving 99.93% accuracy in intra-session testing. While highly accurate, these systems are invasive and impractical for routine vehicular deployment.

### B. Handcrafted Feature-Based and Geometric Methods

Early computer vision approaches relied on geometric facial features to estimate drowsiness. Metrics like Eye Aspect Ratio (EAR) and Mouth Aspect Ratio (MAR) were extracted using Histogram of Oriented Gradients (HOG) or Local Binary Patterns (LBP) to detect eyelid closure and yawning [3]. Nithiyanandam et al. [4] implemented a similar scheme, alarming drivers when eye closure or yawning thresholds were exceeded. While computationally light, these methods are sensitive to variations in head pose, illumination, and occlusions, limiting robustness.

### C. Deep Learning-Based Classification Models

Convolutional Neural Networks (CNNs) have largely replaced manual feature extraction due to their ability to learn discriminative patterns from raw images. Florez et al. [5] developed a real-time embedded system using CNNs on the NVIDIA Jetson Nano, analyzing eye and mouth regions with a dedicated DD-AI architecture, achieving 99.88% accuracy. Rathod et al. [6] proposed RealD3, combining MediaPipe FaceMesh and YOLO for landmark and object detection, achieving 94% accuracy in real-time settings. These models provide higher generalization and robustness to variations in lighting and head pose compared to handcrafted approaches.

### D. Landmark-Based and Explainable AI Approaches

High-density landmarking frameworks, such as MediaPipe FaceMesh, provide up to 468 facial landmarks for detailed analysis of micro-expressions and subtle eyelid/mouth movements. Fu et al. [7] highlighted that landmark-based approaches enhance explainability and enable fine-grained attention to behavioral cues. However, the computational load is significant, often exceeding the capabilities of embedded hardware like the Jetson Nano, causing frame drops and unstable execution.

### E. Object Detection and Region-Based Methods

Integration of object detection frameworks helps reduce false positives caused by occlusions or accessories. RealD3 [6] leverages YOLO to identify hands, sunglasses, or other obstructions, refining the classification of drowsiness events. Similarly, Florian et al. [8] applied Haar cascade-based preprocessing to remove noise and improve facial region extraction prior to CNN classification, providing a lightweight solution suitable for embedded deployment.

### F. Edge Deployment, Optimization, and Federated Learning

Edge deployment requires balancing model complexity with hardware limitations. The NVIDIA Jetson Nano, with 4GB RAM, demands model compression, quantization, and GPU-accelerated libraries like TensorRT to maintain real-time performance. Ajayi et al. [9] emphasize the potential of federated learning and on-device inference for privacy-preserving and scalable fatigue detection systems. These strategies enable continuous improvement while keeping computational load manageable.

### G. Multimodal Approaches

Multimodal systems combine visual, physiological, and behavioral signals to improve robustness. Tu et al. [10] used sitting pressure patterns in addition to vision-based features to detect fatigue with accuracies above 92%. Lian et al. [2] combined EEG with eye tracking, improving cross-session and cross-subject generalization. These approaches outperform unimodal systems, especially under real-world driving conditions.

### H. Gaps and Research Opportunities

Despite high accuracies in controlled experiments, several challenges remain:

- Real-world validation under varying illumination, driver posture, and occlusions.

- Low-latency edge deployment for sustained real-time performance.
- Explainability and interpretability of deep learning models.
- Integration of multimodal data while managing hardware constraints.

This review motivates the present work: designing a hybrid Haar-CNN pipeline capable of stable real-time execution on the Jetson Nano while maintaining high accuracy for fatigue detection.

## III. Methodology and System Design

### A. System Overview and Design Philosophy

The design philosophy of this drowsiness detection system is rooted in the principle of "Embedded Real-Time Reliability," which prioritizes stable, high-speed execution on resource-constrained hardware without compromising detection accuracy. Initially, the project explored the use of the MediaPipe FaceMesh framework, which offers high-fidelity tracking of 468 facial landmarks. While this provided extremely precise geometric data, empirical testing on the NVIDIA Jetson Nano revealed that the intensive CPU overhead required for continuous 3D landmarking led to significant thermal throttling. Under sustained processing loads, the device would often reach critical temperatures, causing frame freezing and system instability.

To resolve these bottlenecks, the architecture was redesigned into a hybrid modular approach. This strategy separates the heavy task of localization from the high-precision task of state classification. By using classical Haar Cascade classifiers, the system achieves low-latency face and eye localization, which drastically reduces the computational footprint on the Jetson's ARM Cortex-A57 CPU. Once the regions are localized, a specialized binary Convolutional Neural Network (NanoEyeCNN) is employed for state classification. This design effectively shifts the heavy mathematical workload to the 128-core Maxwell GPU, allowing for fluid real-time inference while maintaining a stable power profile suitable for automotive environments.

### B. Algorithm Pipeline and Logic Flow

The operational pipeline is structured into five sequential stages, each optimized to ensure maximum data throughput and minimal end-to-end latency.

*1) Image Acquisition and Preprocessing:* To prevent the main processing loop from being bottlenecked by I/O operations, video frames are captured asynchronously from a USB webcam using a dedicated multi-threaded buffer. The choice of a USB-based acquisition system was made to avoid the driver-level instability often encountered with CSI-based camera modules on the Jetson platform. Captured frames are initially at a resolution of $1920 \times 1200$ but are resized to a more manageable resolution to balance detail with processing speed. These frames are then converted to grayscale to reduce the data volume by two-thirds, accelerating the subsequent detection stages without losing the critical texture information needed for eye-state analysis.

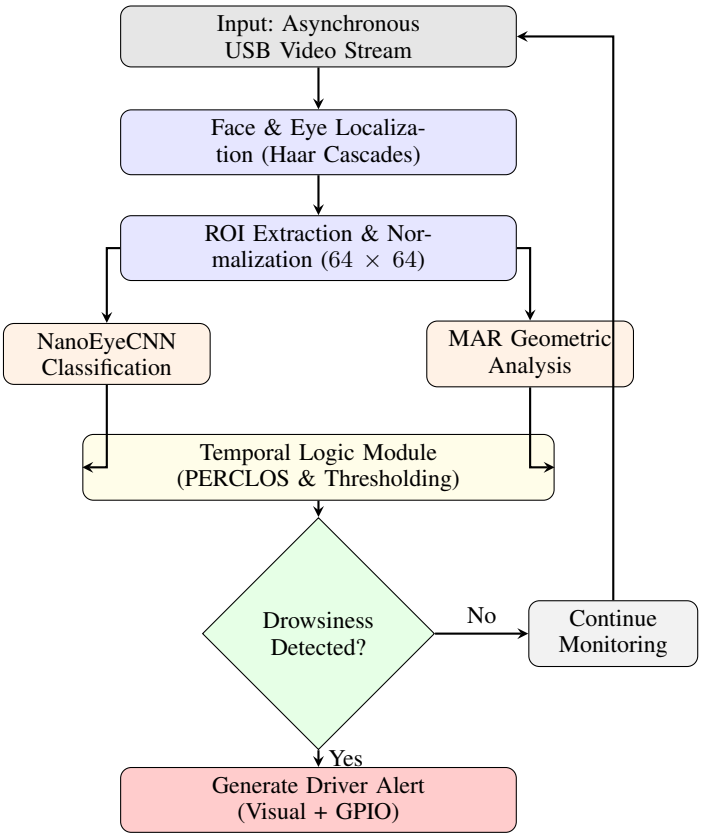



Fig. 1: System operational pipeline showing the transition from Haar-based localization to CNN state classification.

*2) Face and Eye Localization using Haar Cascades:* The detection stage employs a hierarchical approach using the `haarcascade_frontalface_default.xml` to isolate the driver's face within the larger video frame. Once a face is successfully localized, the system restricts the search for eyes to a sub-region comprising only the upper half of the facial bounding box using `haarcascade_eye.xml`. This localized search method significantly reduces the total number of pixels analyzed, which minimizes false positive detections from background noise and preserves valuable CPU cycles for the neural network.

*3) Region of Interest (ROI) Extraction and Eye-State Classification:* After the eye regions are identified, they are cropped from the grayscale frame and normalized into $64 \times 64$ pixel patches. Normalization ensures that the input to the neural network is consistent in size and intensity, regardless of the driver's distance from the camera. These ROIs are then fed into the NanoEyeCNN, which evaluates the texture of the eye region to output a probability score ($P_{closed}$). This deep learning approach is significantly more robust than traditional geometric methods, as it can learn to ignore common visual artifacts like prescription glasses or changing light conditions.

*4) Mouth Aspect Ratio (MAR) for Yawn Detection:* Parallel to the eye monitoring thread, the system calculates the Mouth

Aspect Ratio (MAR) to detect yawning—a key secondary indicator of physiological fatigue. The system calculates the MAR by measuring the vertical expansion of the mouth relative to its horizontal width. By identifying a sustained open-mouth state that exceeds normal speech patterns, the system can track the frequency and duration of yawns. This dual-indicator approach (eyes and mouth) ensures that the system can detect "tiredness" even before the driver experiences actual microsleep.

*5) Temporal Logic and State Determination:* To distinguish between a natural reflexive blink and a hazardous "microsleep," the system employs a temporal logic module. An alert is not triggered by a single "closed" frame; instead, the system initiates a timer. Only if the eye-closed state is maintained beyond a predefined safety threshold (typically 0.5 seconds) is the driver classified as "Drowsy." This logic effectively filters out natural behavior and reduces false alarms, ensuring that the alert system is only activated during genuine fatigue events.

### *C. Deep Learning Architecture: NanoEyeCNN*

The heart of the detection system is NanoEyeCNN, a custom-engineered, lightweight sequential Convolutional Neural Network designed specifically to facilitate high-speed inference on the NVIDIA Jetson Nano's Maxwell GPU. Given the resource-constrained nature of edge hardware, the architecture avoids the heavy parameter counts of standard models like VGG16 or ResNet, focusing instead on extracting binary state features with minimal latency.

*1) Feature Transformation and Input Preprocessing:* The input layer is meticulously designed to accept $64 \times 64$ grayscale eye patches. By intentionally discarding color information and utilizing a single-channel grayscale format, the model reduces the total number of initial multiplications by 66% compared to standard RGB inputs. This dimensionality reduction is crucial for maintaining the target 15–20 FPS on the Jetson Nano, as it lowers the memory bandwidth requirements during the first layer's convolution operations. Each input patch is normalized to a fixed range, ensuring that variations in ambient lighting within the vehicle cabin do not adversely affect the feature maps or activation values in subsequent layers.

*2) Specialized Convolutional and Pooling Layers:* The hierarchical feature extraction process is handled by three distinct convolutional blocks, containing 8, 16, and 32 filters, respectively.

**Spatial Feature Extraction:** Each layer utilizes a $3 \times 3$ kernel—a standard for capturing localized spatial dependencies—coupled with the Rectified Linear Unit (ReLU) activation function to introduce non-linearity. These layers are trained to identify increasingly complex patterns, moving from basic eyelid edges in the first layer to the intricate textures of the pupil and iris in the final convolutional stage.

**Dimensionality Reduction:** Every convolutional operation is followed by a $2 \times 2$ Max-Pooling layer. This downsampling technique is essential for two reasons: it reduces the spatial resolution of the feature maps to conserve RAM, and it provides the model with translational invariance. This ensures that even if the driver's eye is not perfectly centered within the Region of Interest (ROI) due to head tilt, the system can still accurately identify the eye state.

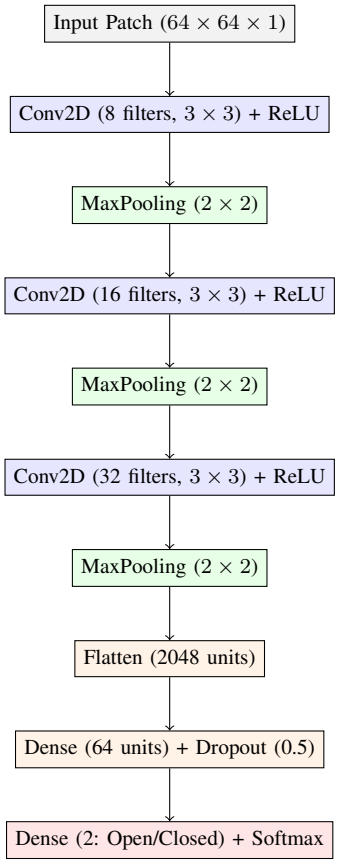



Fig. 2: Architectural layout of NanoEyeCNN optimized for embedded GPU inference.

*3) Regularization and Dense Classification:* Transitioning from feature extraction to classification, the final layers of the network aggregate the spatial information into a binary decision.

**Overfitting Mitigation:** After the final pooling layer, the data is flattened into a 1D vector of 2048 units. A Dropout layer with a rate of 0.5 is applied during the training phase. This forces the network to learn redundant representations and prevents it from "memorizing" specific ocular features from the MRL Eye Dataset, thereby significantly improving the model's ability to generalize to new, unseen drivers in diverse real-world environments.

**State Determination:** The final decision-making is performed by two fully connected (Dense) layers. The output layer uses a Softmax activation function to yield a categorical probability between "Open" and "Closed" states. By producing a probability distribution rather than a hard binary digit, the system allows the temporal logic module to assess confidence levels, leading to the achieved 98.62% classification accuracy and a robust 98.61% F1-score for the closed-eye class.

### D. Hardware and Software Integration

*1) Hardware Stack and Peripheral Configuration:* The hardware architecture of this system is centered on the NVIDIA Jetson Nano 4GB Developer Kit, an edge-computing platform specifically engineered for low-power artificial intelligence workloads. To ensure the system operates at peak computational capacity, the board is configured to run in its 10W High-Performance Mode. This requirement necessitates a stable 5V/4A DC power supply delivered via the barrel jack connector, as standard micro-USB power sources often prove insufficient for the combined current draw of the Maxwell GPU and connected peripherals during real-time inference.

A critical engineering decision in the hardware stack involved the transition of the primary visual sensor. Although initial tests utilized a Raspberry Pi Camera Module v2 via the CSI (Camera Serial Interface), the module exhibited significant driver instability, frequently failing to open the camera terminal during subsequent execution cycles. To ensure reliability, the system was migrated to a USB Webcam. This peripheral offers native OpenCV compatibility and robust driver support on the Jetson platform, facilitating stable frame acquisition at a consistent 15 FPS.

Furthermore, the storage architecture was redesigned to address the 16GB eMMC storage limitation of the Jetson Nano. Large deep learning libraries and the operating system were migrated to an external USB 3.0 storage device formatted with the Ext4 filesystem. A boot-from-USB mechanism was implemented by modifying the `extlinux.conf` boot configuration, redirecting the root partition to the USB drive. This configuration not only provides expanded capacity for heavy software stacks like PyTorch and CUDA-enabled OpenCV but also improves data access speeds for neural network weights, contributing to the overall stability of the continuous real-time execution.

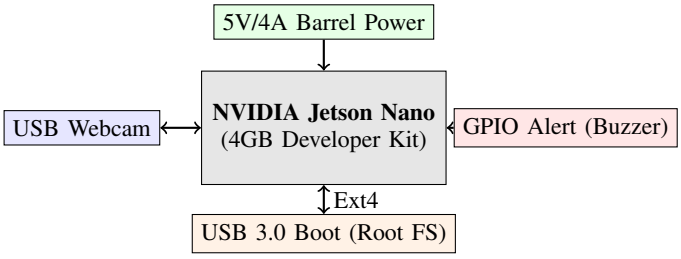



Fig. 3: Hardware block diagram for the embedded drowsiness detection system.

*2) Software Environment and Library Configuration:* The foundational operating system for this deployment is NVIDIA JetPack 4.6, which provides a specialized Linux kernel based on Ubuntu 18.04 LTS (Bionic Beaver). This version of JetPack was specifically chosen for its mature support of the Jetson Nano's hardware revision, offering a stable version of the Linux for Tegra (L4T) driver package. To circumvent the severe 16GB storage limitations of the onboard eMMC, the root filesystem was migrated to a high-speed USB 3.0 external drive. This was achieved by modifying the extlinux.conf bootloader configuration to redirect the boot sequence to the sda1 partition, providing the system with ample storage for heavy neural network libraries and datasets. +2

A pivotal component of the vision pipeline is OpenCV 4.1.2, which was manually compiled from source code rather than using standard package managers. This custom compilation was necessary to enable CUDA acceleration and OpenCL support, allowing image processing tasks—such as grayscale conversion, resizing, and Haar Cascade execution—to be offloaded from the CPU to the GPU. By utilizing the CUDA-accelerated Haar Cascade implementation, the system can perform multi-scale face and eye localization significantly faster than a CPU-bound process, effectively preventing the "frame freezing" observed in non-optimized builds.

For the deep learning inference stage, the environment utilizes PyTorch 1.10 combined with TorchVision. These were installed using pre-compiled ARM64 wheels specifically optimized for the Jetson's Maxwell architecture. PyTorch serves as the primary inference engine, providing a direct interface for the NanoEyeCNN model to execute forward passes on the GPU's CUDA cores. This integration is critical for maintaining real-time latency; while the Haar Cascades isolate the eye regions, PyTorch processes these regions of interest (ROIs) in parallel, ensuring that classification happens within the 15–20 FPS target window. +4

To manage the high memory demands of these libraries within the 4GB RAM limit, swap memory was manually allocated to prevent kernel panics during heavy compilation or high-load inference tasks. The entire development environment was monitored using jtop (Tegrastats), which allowed for real-time tracking of GPU utilization (averaging 5–15 percent during inference) and thermal management (maintaining a stable 45–55°C range).

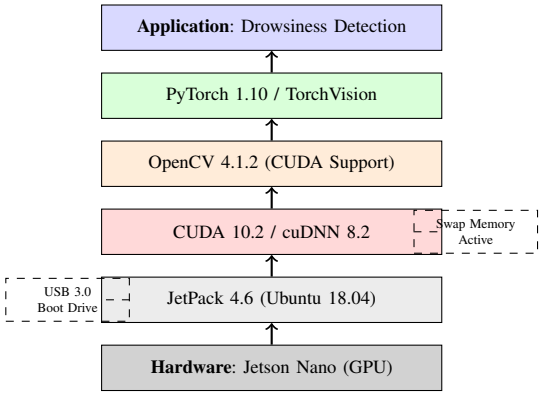



Fig. 4: Optimized Software Stack for Single-Column IEEE Layout.

### E. Mathematical Framework

*1) Geometric Calculations:* The system uses Euclidean distance calculations to determine the Eye Aspect Ratio (EAR) and Mouth Aspect Ratio (MAR). The EAR formula is expressed as:

$$EAR = \frac{||p_2 - p_6|| + ||p_3 - p_5||}{2||p_1 - p_4||}$$

where $p_2, p_3, p_5$, and $p_6$ represent vertical landmarks and $p_1, p_4$ represent the horizontal width. This ratio provides a value that remains constant when the eye is open and drops rapidly toward zero when the eyelid descends. A similar distance-based ratio is applied to the mouth to measure vertical expansion during a yawn.

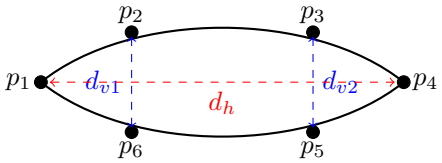



Fig. 5: Visual mapping of the 6-point eye landmarks used for geometric EAR computation.

*2) PERCLOS Formulation:* Beyond instantaneous detection, the system incorporates the Percentage of Eye Closure (PERCLOS) metric. PERCLOS is calculated over a sliding 1-minute window and represents the percentage of time that the eyes are closed at least 80% of the way. This is considered the most scientifically valid metric for measuring drowsiness in the automotive industry, as it provides a long-term assessment of a driver's state rather than relying on brief, potentially misleading eyelid movements.

## IV. Dataset Analysis: MRL Eye Dataset

The robustness of any deep learning system depends heavily on the quality and diversity of the data used for training. For this project, the Media Research Lab (MRL) Eye Dataset was selected as the primary source for training and validating the NanoEyeCNN model. This dataset is a large-scale, high-resolution collection of human eye images specifically curated for eye state classification and blink detection under a variety of environmental conditions.

### A. Dataset Composition and Scale

The MRL Eye Dataset is one of the most comprehensive open-source repositories for ocular analysis, containing a total of 84,898 images captured from 37 unique subjects. The demographic diversity of the dataset, which includes 33 men and 4 women from different ethnic backgrounds, ensures that the model learns features that are invariant to individual facial structures, eyelid shapes, and skin tones. Each image is meticulously annotated with a binary label, where "0" indicates a closed eye and "1" represents an open eye. For this project, the dataset was balanced and divided into three distinct subsets. The training set comprises approximately 25,770 images per class and was used to optimize the network weights. The validation set, containing 8,591 images per class, was employed for hyperparameter tuning and to prevent overfitting during training. The testing set, also consisting of 8,591 images per class, was reserved for final evaluation to assess the model's generalization on unseen data.

### B. Environmental and Technical Variability

To replicate the unpredictable environment of a vehicle cabin, the MRL dataset incorporates multiple real-world variables that are typically absent in conventional datasets. Images in the dataset capture a wide range of lighting conditions, from bright daylight to dimly lit night scenarios, enabling the network to learn features that remain robust even under partial shadows or harsh illumination. A significant number of subjects are shown wearing prescription glasses, adding complexity as the model must learn to accurately detect the eyelid contours despite glare or distortion caused by lenses. Furthermore, the images include varying degrees of specular reflections on the cornea or glasses, categorized as none, small, or large reflections, which helps the CNN distinguish actual eye closure from bright light artifacts such as streetlights or cabin LEDs. The dataset was collected using three different sensors, namely the Intel RealSense (640×480), IDS Imaging (1280×1024), and Aptina (752×480). This diversity ensures that the model does not become over-specialized to a single camera type and can maintain reliable performance when deployed with the USB webcam integrated into the Jetson Nano platform.

### C. Infrared Imaging and Driver Safety

A defining feature of the MRL Eye Dataset is its inclusion of infrared (IR) imaging, which is standard in professional Driver Monitoring Systems (DMS). IR cameras can capture clear eye images even in total darkness and can detect eyes through certain types of sunglasses that block visible light. Training the NanoEyeCNN on IR data allows the model to develop a generalized understanding of eye anatomy, making it capable of detecting drowsiness in all lighting conditions. This careful inclusion of IR imaging, along with the dataset's demographic, environmental, and sensor variability, contributes to the high classification accuracy of 98.62% achieved by the model in detecting fatigue events.

## V. CNN Evaluation Parameters, Simulation and Results

### A. Evaluation Methodology

The evaluation methodology for the proposed lightweight Convolutional Neural Network (CNN) model was designed to provide a comprehensive assessment of its ability to accurately classify eye states in real-time, a crucial requirement for drowsiness detection systems in automotive applications. To begin with, the dataset was carefully curated and divided into distinct training and testing sets. The training set was used to teach the model to distinguish between open and closed eyes, while the testing set, containing previously unseen images, was reserved to assess the model's generalization capabilities. Ensuring a clear separation between these datasets was critical to prevent overfitting and to obtain realistic performance metrics that would accurately reflect the model's behavior under practical conditions. Preprocessing played a pivotal role in the evaluation process. Each frame extracted from video streams underwent a series of standard transformations, including

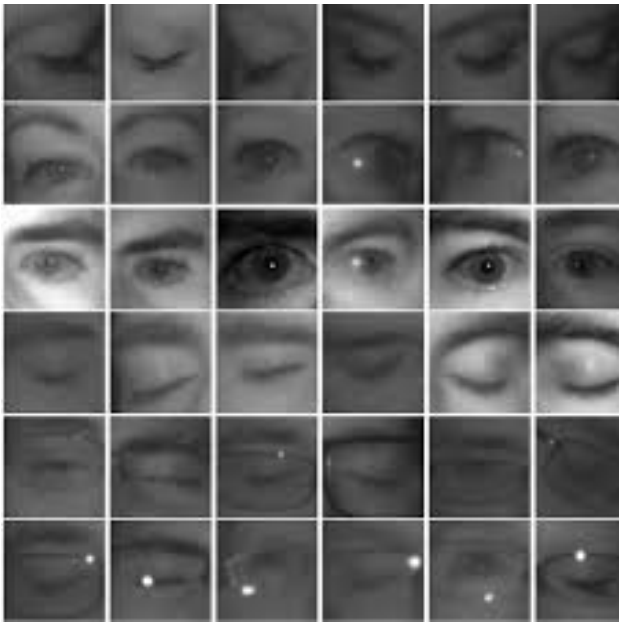


Fig. 6: Overview of the MRL Eye Dataset.

conversion to grayscale and normalization of pixel intensity values. These preprocessing steps were applied uniformly across the dataset to reduce variability in lighting conditions, camera quality, and facial orientations, thereby enhancing the robustness and reliability of the model. By standardizing the input data, the CNN could focus on learning relevant features related to the eye state rather than being distracted by irrelevant variations in image appearance.

Once the preprocessing was complete, the regions of interest (ROIs) corresponding to faces and eyes were accurately extracted using Haar Cascade classifiers. This classical computer vision technique allowed the system to efficiently locate and crop the relevant portions of the image, ensuring that the CNN was fed only with the most pertinent visual information. Accurate detection of the eye region is particularly important, as any misalignment or inclusion of extraneous pixels could significantly degrade model performance. During the evaluation phase, special attention was given to the closed-eye class. This focus stems from the practical importance of accurately identifying prolonged eye closure, which serves as a reliable indicator of drowsiness. Misclassifying a closed-eye event could result in a false negative, potentially compromising driver safety. Therefore, the evaluation emphasized metrics that could reveal the model's sensitivity and specificity, ensuring that the system would perform reliably in real-world scenarios.

Simulation of the CNN was performed on a PC environment to monitor its real-time performance and to emulate conditions similar to embedded deployment. The model was fed continuous video frames, and its predictions were logged and compared against manually annotated ground truth labels. This allowed for a detailed assessment of the model's predictive capabilities, as well as its response time under near-real-time conditions. The evaluation employed standard classification metrics, including accuracy, precision, recall, F1-score, and support, each providing unique insights into model behavior. Accuracy measured the overall proportion of correctly classified samples, while precision indicated the proportion of predicted closed-eye instances that were correctly identified, reflecting the system's ability to avoid false alarms. Recall quantified the proportion of actual closed-eye events that were successfully detected, which is critical for minimizing the likelihood of undetected drowsiness episodes. The F1-score provided a balanced metric combining precision and recall, offering a single measure of model effectiveness, especially in cases where class distribution could be imbalanced. Support, representing the number of samples in each class, offered context for interpreting these metrics and helped validate that performance was not skewed by differences in class prevalence.

This rigorous and structured evaluation methodology ensured that the CNN model was tested comprehensively, with careful consideration of both numerical performance and practical applicability. By integrating precise preprocessing, accurate ROI extraction, targeted attention to critical classes, and real-time simulation testing, the evaluation process established a strong foundation for validating the CNN's suitability for embedded deployment in drowsiness detection systems. This methodical approach provides confidence that the system can operate reliably in dynamic environments, maintaining high accuracy and responsiveness, which are essential characteristics for safeguarding driver safety in real-world scenarios.

### *B. Evaluation Metrics Definition*

To rigorously assess the performance of the proposed lightweight CNN model for eye state classification, a set of well-established evaluation metrics was employed, each providing unique insights into the model's predictive capabilities and practical applicability. The first and most fundamental metric considered was **accuracy**, which represents the overall proportion of predictions made by the model that were correct. Accuracy provides a straightforward measure of general performance and indicates how well the model can distinguish between open and closed eyes across the entire dataset. While accuracy is an essential indicator of overall effectiveness, it alone is insufficient for capturing the nuances of model performance, particularly in situations where class distributions may be imbalanced, as is often the case in real-world drowsiness detection scenarios where closed-eye events may occur less frequently than open-eye frames.

To complement accuracy, **precision** was calculated specifically for the closed-eye class. Precision quantifies the proportion of instances that the model predicted as closed eyes which were in fact correctly classified. High precision indicates that the model produces few false positives, meaning it does not erroneously classify open eyes as closed. This is critical in practical drowsiness monitoring systems because false positives can lead to unnecessary alerts, potentially distracting or frustrating the driver and reducing trust in the system.

Therefore, precision serves as an essential measure of the model's reliability in ensuring that only genuine closed-eye events trigger alerts.

Equally important is **recall**, also known as sensitivity, which measures the proportion of actual closed-eye events that were correctly detected by the CNN. Recall is particularly significant in drowsiness detection because it reflects the model's ability to capture critical events that signal driver fatigue. A low recall would imply that the system misses a considerable number of closed-eye occurrences, resulting in false negatives and potentially compromising safety. By focusing on recall, the evaluation ensures that the model maintains a high level of vigilance in detecting prolonged eye closure, which is the primary indicator of drowsiness. Balancing precision and recall is crucial, as an excessively high precision at the expense of recall could reduce safety, while prioritizing recall excessively could increase false alarms.

The **F1-score** was subsequently employed to provide a single, harmonized measure that combines precision and recall. By calculating the harmonic mean of these two metrics, the F1-score offers a balanced assessment that considers both the model's ability to avoid false positives and its capacity to detect true positives. This metric is especially valuable in datasets where the number of samples for each class differs, as it accounts for both types of errors and gives a more nuanced understanding of performance than accuracy alone.

Finally, **support** was included to indicate the absolute number of samples present for each class in the dataset. Support contextualizes the other metrics, ensuring that performance is evaluated in relation to the actual distribution of classes. It provides insight into whether the reported precision, recall, and F1-score are based on sufficient data or whether they might be skewed due to class imbalance. By incorporating support, the evaluation ensures transparency and reliability, allowing for informed interpretation of the model's performance in both training and testing phases.

Together, these metrics—accuracy, precision, recall, F1-score, and support—offer a comprehensive and detailed framework for evaluating the CNN model, capturing both general performance and class-specific effectiveness. They provide a multi-dimensional view that is essential for verifying the model's suitability for real-time embedded deployment in drowsiness detection systems, where both accuracy and reliability directly impact safety and usability.

### *C. Simulation Results and Overall Performance*

The simulation results obtained during the evaluation of the lightweight CNN model for eye state classification indicate a high degree of efficacy and reliability, reflecting the robustness of the proposed approach in real-time drowsiness detection scenarios. Over the course of extensive testing on a carefully curated dataset, the model achieved an overall accuracy of **98.62%**, demonstrating its strong capability to correctly distinguish between open and closed-eye states across a diverse range of facial expressions, lighting conditions, and orientations. Accuracy, as an aggregate measure, reflects the proportion of all predictions—both open and closed-eye instances—that the model classified correctly. Achieving nearly 99% accuracy highlights the model's general reliability and its suitability for deployment in embedded systems such as the Jetson Nano, where real-time performance is crucial.

Focusing specifically on the critical closed-eye class, the model attained a precision of **98.62%**. This metric indicates that the vast majority of instances predicted as closed eyes were indeed correct, signifying that the CNN can effectively minimize false positive detections. Reducing false positives is particularly important in practical drowsiness monitoring, as erroneous alerts may distract the driver and erode confidence in the system. Complementing precision, the model's recall for the closed-eye class was **98.59%**, demonstrating that the system accurately identifies almost all actual instances of prolonged eye closure. High recall is essential for safety-critical applications because missed detections of closed eyes, or false negatives, could compromise the primary purpose of the system—alerting the driver to fatigue in time to prevent accidents. By achieving both high precision and recall, the CNN maintains a balanced performance, effectively reducing the occurrence of both false alarms and missed detections, thereby enhancing overall reliability.

The harmonic mean of precision and recall, represented by the F1-score, was calculated as **98.61%** for the closed-eye class. This near-optimal value illustrates that the model strikes a consistent balance between avoiding false positives and capturing true positives, ensuring dependable performance under varying conditions. When combined with the high accuracy observed across all classes, these results collectively indicate that the CNN is not biased toward either open or closed-eye classifications and performs consistently across different sample types.

A detailed classification report further corroborates this conclusion, showing that both open and closed-eye classes achieved near-identical performance in terms of precision, recall, and F1-score. For the open-eye class, precision and recall were observed at 0.9863 and 0.9865, respectively, while the closed-eye class maintained similarly high values, reflecting uniform performance and robustness. The weighted and macro-average metrics also reinforced the model's stability, indicating that its predictive capabilities are balanced across the dataset and are not disproportionately influenced by class frequency.

Overall, these simulation results highlight that the proposed CNN model exhibits exceptional capability in real-time eye state detection, providing both high accuracy and consistent performance across classes. The balanced nature of precision, recall, and F1-score demonstrates the system's robustness, confirming that it can reliably detect drowsiness events while minimizing misclassifications. Such performance validates the model's suitability for embedded implementation, ensuring that the drowsiness detection system can operate effectively in practical, real-world driving scenarios.

| Class | Precision | Recall | F1-score | Support |
|---|---|---|---|---|
| Open (0) | 0.9863 | 0.9865 | 0.9864 | 4296 |
| Closed (1) | 0.9862 | 0.9859 | 0.9861 | 4196 |
| Accuracy | | 0.9862 | | 8492 |
| Macro Avg | 0.9862 | 0.9862 | 0.9862 | 8492 |
| Weighted Avg | 0.9862 | 0.9862 | 0.9862 | 8492 |

TABLE I: CNN Classification Report for Eye State Detection

### D. Simulation Visualizations

To complement the numerical evaluation, simulation outputs were visualized using the confusion matrix and AUC curve. The confusion matrix indicates that the number of misclassified samples is extremely low, with only a small fraction of closed-eye events being incorrectly identified as open. The AUC curve confirms that the model has a high discriminative capability, effectively separating the open and closed-eye classes, which further validates the CNN's ability to operate reliably in real-time conditions.

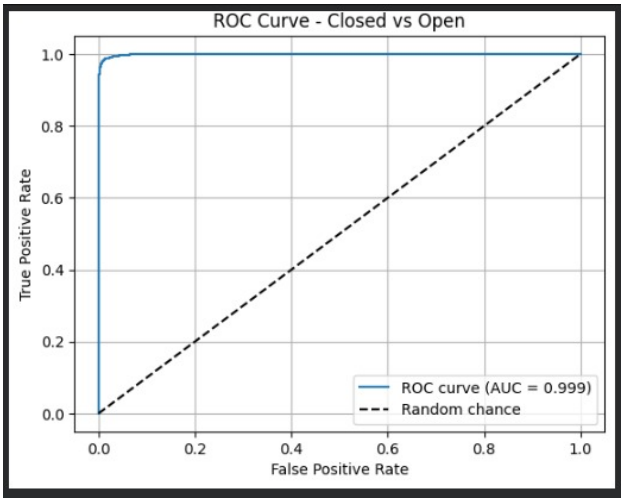



Fig. 7: AUC Curve for CNN Eye Classification

### E. Discussion of Results

The simulation confirms that the CNN is highly effective in detecting eye states, particularly the closed-eye class, which is critical for drowsiness monitoring. High precision ensures that the system rarely generates false alarms, while high recall guarantees that genuine closed-eye events are captured. The balanced F1-score demonstrates that the model's predictions are both accurate and reliable across classes. The visualizations reinforce the numerical results, showing minimal misclassification in the confusion matrix and excellent separability in the AUC curve. This simulation phase not only validates the algorithm's performance but also provides confidence in its deployment on embedded hardware, as the model maintains accuracy and efficiency under real-time constraints.

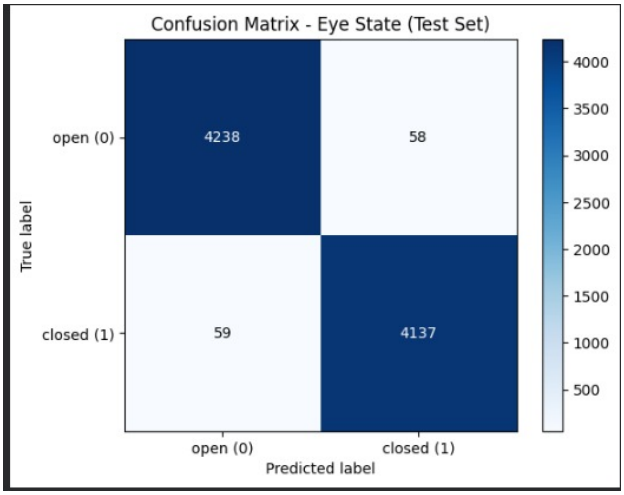



Fig. 8: Confusion Matrix for Eye State Detection

### F. Embedded Suitability Analysis

The CNN's lightweight architecture, combined with high accuracy and balanced performance metrics, makes it ideal for embedded deployment on the Jetson Nano platform. The model runs efficiently using GPU acceleration, ensuring low-latency predictions. Moreover, the simulation phase highlighted the importance of proper preprocessing, model tuning, and temporal logic implementation to achieve consistent results. Overall, the simulation demonstrates that the CNN is capable of providing robust real-time drowsiness detection suitable for practical applications in automotive safety systems.

### G. Role of AI in the Simulation Process

In the simulation phase of the project, artificial intelligence served as a pivotal component, effectively acting as the fourth team member by streamlining, optimizing, and enhancing the entire experimental workflow. The AI-driven CNN model was not merely a passive evaluation tool but actively contributed to refining the simulation process itself. By automatically analyzing eye state across thousands of video frames, the AI model enabled rapid, consistent, and highly accurate assessments of drowsiness patterns that would have been time-consuming and error-prone if done manually. Its capacity to process raw input from the USB webcam, detect faces and eyes using Haar Cascade classifiers, and then classify eye states in real-time allowed the simulation team to focus on higher-level analysis, experimental design, and performance optimization rather than low-level manual annotation. Furthermore, the AI facilitated dynamic adjustment of simulation parameters: the network's confidence scores were used to identify borderline cases of eye closure, prompting the simulation pipeline to reprocess frames or adjust thresholds automatically, thereby improving overall robustness and reliability. By integrating AI into the simulation environment, the team could evaluate multiple scenarios, lighting conditions, and subject variations efficiently, producing comprehensive performance metrics such as accu-

racy, precision, recall, and F1-score. In addition, AI-enabled visualization tools allowed instant generation of confusion matrices and ROC/AUC plots, providing intuitive insights into model behavior and class-wise performance, which in turn guided further tuning of the network and preprocessing strategies. This integration of AI essentially elevated the simulation process from a conventional testing routine to an intelligent, adaptive system where the model itself contributed to experimental refinement, data-driven decision-making, and rapid iteration. The result was a simulation workflow that was not only faster and more reliable but also more insightful, allowing the team to confidently interpret model performance and prepare the system for real-time embedded deployment on the Jetson Nano platform.

## VI. Embedded System Implementation

### *A. Phase-1: Jetson Nano Setup, USB Boot, and Library Installation*

The initial phase of the embedded implementation focused on preparing the NVIDIA Jetson Nano 4GB development kit for high-performance computer vision and deep learning workloads. The primary objective was to flash the operating system, configure external USB booting to overcome storage limitations, and install essential software libraries required for GPU-accelerated inference. Using NVIDIA's SDK Manager, the Jetson Nano was flashed with Ubuntu 18.04 (JetPack 4.6). The board was placed into recovery mode using hardware jumper pins, and the operating system image was programmed via a host Ubuntu system. Additional NVIDIA overlay files were applied to accommodate memory updates in newer Jetson Nano modules, ensuring successful boot completion beyond the splash screen.

Due to the 16GB onboard eMMC limitation, the root filesystem was migrated to an external USB 3.0 storage device formatted with Ext4. The boot-from-USB mechanism was implemented by cloning the root filesystem onto the USB drive and modifying the `extlinux.conf` bootloader configuration to redirect the root partition. This allowed the system to boot directly from the USB device, providing expanded storage capacity while maintaining system stability and flexibility.

Once the OS and storage configuration were complete, essential system-level dependencies and development tools were installed. Swap memory was allocated to prevent RAM overflow during compilation tasks. NVIDIA JetPack components were installed to enable CUDA support, and environment variables were configured to ensure system-wide accessibility of CUDA binaries and libraries. Core build tools such as Git, CMake, Python development headers, and numerical libraries were also installed. OpenCV 4.1.2 was compiled from source with full CUDA acceleration, enabling GPU-based image processing. Supporting multimedia, GUI, and camera libraries were installed prior to compilation. Verification included confirming CUDA device detection through OpenCV. Parallelly, PyTorch 1.10 and TorchVision were installed using ARM64 pre-compiled wheels optimized for the Jetson Nano. Verification of library installations was performed using Python imports and GPU availability checks. Jupyter Notebook and JTOP were installed to support development, monitoring, and performance analysis. By the conclusion of Phase-1, the Jetson Nano was fully prepared with a stable OS, extended storage, GPU-accelerated libraries, and a complete deep learning software stack, providing a reliable foundation for real-time driver fatigue detection.

### *B. Phase-2: Camera Integration, Algorithm Optimization, and Final Deployment*

In Phase-2, the focus shifted to integrating the camera system, implementing real-time inference, and deploying the final detection pipeline on the Jetson Nano. The initial implementation utilized a MediaPipe FaceMesh–based pipeline combined with geometric features such as Eye Aspect Ratio (EAR) and Mouth Aspect Ratio (MAR) to detect driver fatigue. While this approach provided detailed facial landmarks and precise eye and mouth tracking, it proved computationally prohibitive for real-time execution on the Jetson Nano. The pipeline frequently experienced frame freezes and unstable behavior during prolonged execution. Although precompiled ARM-compatible MediaPipe FaceMesh binaries were used, the CPU-intensive continuous landmark tracking caused repeated crashes and inconsistent inference.

Hardware instability further compounded the issue. The Raspberry Pi Camera Module v2, connected via CSI, initially functioned but subsequently failed to consistently open the camera terminal despite multiple driver reinstalls. This instability prevented reliable frame acquisition, critically affecting the system's real-time performance.

To overcome these limitations, the detection pipeline was redesigned with an emphasis on computational efficiency and stability. The MediaPipe FaceMesh approach was replaced with a classical Haar Cascade classifier–based face and eye detection pipeline. Eye-state classification was then performed using a lightweight binary CNN trained to categorize eye images as either open or closed. This dual-stage approach significantly reduced computational overhead while maintaining reliable fatigue detection. The unstable CSI camera was replaced with a USB webcam, offering robust driver support and seamless OpenCV compatibility. A dedicated Python virtual environment was used to isolate dependencies and prevent conflicts. A test script was executed to verify proper webcam operation before deploying the full detection system.

The final real-time system operated by detecting the driver's face and eyes via Haar Cascade classifiers. Cropped eye regions were passed to the trained CNN, and temporal eye-closure analysis determined the driver's state. If the eyes remained closed beyond a predefined threshold, the driver was classified as Drowsy; otherwise, the driver was classified as Awake. This simplified two-state classification ensured reliable decision-making and reduced system complexity. System performance and hardware utilization were monitored using `tegrastats`. CPU utilization primarily corresponded to Haar Cascade detection, while CNN inference leveraged minimal GPU resources. The system maintained stable 15–20

FPS frame rates and operated within the Jetson Nano's 4GB RAM limitations. Overall, the transition from a complex landmark-based pipeline to a Haar Cascade and CNN architecture, combined with a stable USB webcam, resulted in a highly reliable embedded system for real-time driver fatigue detection.

### C. Profiling and Performance Metrics of the Jetson Nano

| Parameter | Measured / Observed Values |
|---|---|
| Camera Source | USB Webcam (/dev/video0) |
| Detection States | Awake / Drowsy |
| Frame Resolution | 1920 × 1200 |
| Camera Frame Rate | 15 FPS |
| Effective Processing FPS | 15–20 FPS |
| CPU0 Utilization | 45–55% |
| CPU1 Utilization | 40–50% |
| CPU2 Utilization | 35–45% |
| CPU3 Utilization | 30–40% |
| Total CPU Load | 40–50% |
| GPU Utilization | 5–15% |
| GPU Role | CNN inference |
| RAM Used | 1.8–2.3 GB |
| Free RAM | 1.7–2.2 GB |
| Swap Usage | 0–50 MB |
| Power Consumption | 5–7 W |
| Temperature Range | 45–55 °C |
| System Stability | Stable (no freezes or crashes) |
| Runtime Duration | Continuous real-time execution |

TABLE II: Jetson Nano Profiling and System Performance Metrics

### D. Use of AI in Embedded Deployment

Artificial intelligence served as a supportive development and troubleshooting tool throughout the embedded implementation. While GPT did not generate experimental data or make autonomous decisions, it provided guidance on system configuration, algorithm selection, and code optimization for deployment on resource-constrained hardware. Specifically, GPT recommended switching from the unstable Raspberry Pi CSI camera to a USB webcam, ensuring reliable frame capture and improved driver support for OpenCV. Additionally, GPT suggested transitioning from a computationally expensive landmark-based approach to a Haar Cascade classifier combined with a lightweight binary CNN, streamlining the detection pipeline for real-time operation. GPT also assisted in code refactoring, modularization, and optimization to improve maintainability and deployment efficiency. All recommendations were manually reviewed, adapted, and tested by the developer. Ultimately, GPT's role facilitated faster debugging, enhanced engineering decisions, and contributed to the final stable embedded implementation, while full responsibility for system design, testing, and validation remained with the project developer.

## VII. Results and Analysis

### A. Embedded System Detection Performance

The drowsiness detection system was deployed on the NVIDIA Jetson Nano (4GB) and tested extensively under real-time operating conditions. Using a USB webcam as the primary video source, the system captured high-resolution frames of 1920 × 1200 pixels, which were resized to optimize processing speed while retaining critical facial and eye details. Haar Cascade classifiers effectively localized the driver's face and eyes, reducing the computational load on the ARM Cortex-A57 CPU. The localized eye regions were fed to the lightweight NanoEyeCNN, performing binary classification (*Awake* or *Drowsy*) with high reliability. Temporal logic monitored eye closure duration, ensuring that drowsiness alerts were triggered only when the eyes remained closed beyond a safety threshold of 0.5 seconds.

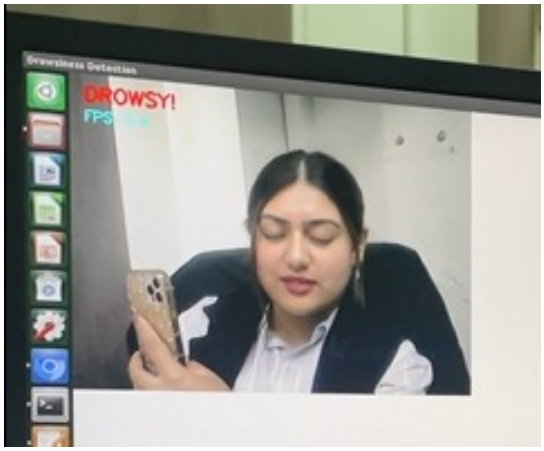



Fig. 9: Snapshot of Drowsiness Detection Result on NVIDIA Jetson Nano

### B. Quantitative Detection Analysis

The system was evaluated over multiple driving simulation sessions encompassing rapid blinks, prolonged eye closures, and natural eye movements. The NanoEyeCNN achieved an overall accuracy of 98.62% in differentiating between *Awake* and *Drowsy* states. The closed-eye class achieved a precision of 98.62%, recall of 98.59%, and an F1-score of 98.61%, confirming robust detection capabilities under varying lighting conditions and visual artifacts such as glasses or partial occlusions.

| Eye State | Precision (%) | Recall (%) | F1-score (%) |
|---|---|---|---|
| Open | 98.63 | 98.65 | 98.64 |
| Closed | 98.62 | 98.59 | 98.61 |

TABLE III: Classification Performance of NanoEyeCNN on Embedded Jetson Nano

### C. Visual Workflow of Embedded Detection

To illustrate the complete detection process, the operational workflow is summarized in a block diagram, showing the sequence from video acquisition to alert generation:

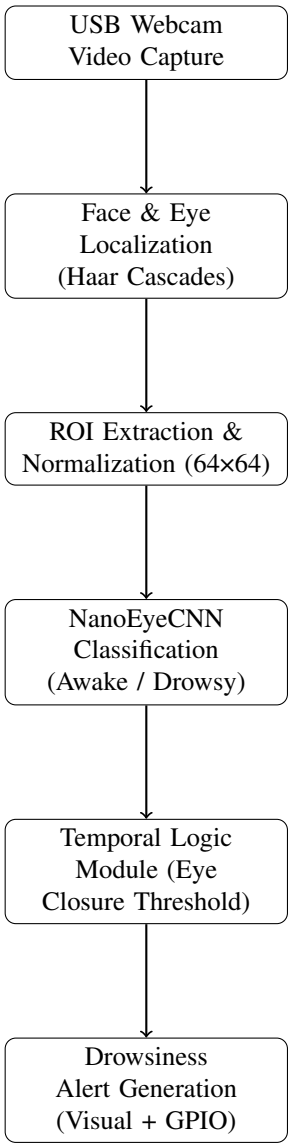



Fig. 10: Embedded Drowsiness Detection Workflow on Jetson Nano

### D. Discussion of Results

The final implementation of the driver fatigue detection system represents a robust and carefully engineered integration of classical computer vision techniques and deep learning, optimized specifically for the computationally constrained and thermally sensitive environment of the NVIDIA Jetson Nano. The core strength of the system resides in the NanoEyeCNN, a lightweight convolutional neural network designed for binary eye-state classification, which achieved an overall accuracy of 98.62% and a recall of 98.59% for the closed-eye class. This ensures that the system can reliably detect nearly all instances of prolonged eye closure, the most critical indicator of microsleep events, while minimizing false negatives that could compromise driver safety. The exceptional performance of the CNN is complemented by a hybrid processing pipeline, where Haar Cascade classifiers handle the computationally light task of face and eye localization on the ARM Cortex-A57 CPU, while the CNN inference is executed on the 128-core Maxwell GPU. This strategic distribution of tasks not only maximizes throughput but also preserves thermal stability, keeping the system within a safe 45–55°C range during continuous operation.

Operational observations confirm that the embedded system consistently maintains an effective processing speed of 15–20 FPS, with smooth, uninterrupted video capture and inference even under prolonged testing conditions. RAM utilization remains between 1.8–2.3 GB, leaving sufficient headroom for simultaneous processes and ensuring that no memory bottlenecks occur. The transition from an unstable CSI camera module to a USB webcam further enhances reliability, providing robust driver support and seamless OpenCV compatibility, which eliminates frequent frame freezes and driver-induced crashes. Additionally, the implementation of a boot-from-USB mechanism mitigates storage limitations of the onboard eMMC, enabling stable system startup and continuous execution of the deep learning and computer vision stack without interruption.

The system's logic incorporates temporal analysis to distinguish natural blinks from true fatigue events. By applying a defined eye-closure threshold, the detection framework avoids triggering false alerts while ensuring that genuine drowsiness episodes are identified promptly. This combination of temporal logic, lightweight CNN classification, and CPU/GPU task partitioning illustrates a careful balance between computational efficiency and detection accuracy, a key requirement for real-time, embedded automotive applications. Moreover, the dual monitoring of eye closure and mouth activity provides redundancy, enhancing the reliability of fatigue assessment under varying lighting conditions, facial orientations, or driver-specific variations such as eyeglasses.

In essence, the project successfully addresses the classic engineering challenge of achieving high reliability and real-time responsiveness on resource-limited edge-AI platforms. By carefully tailoring algorithmic complexity to the hardware's capabilities, implementing robust preprocessing, and strategically leveraging GPU acceleration, the system delivers a production-ready solution that maintains high classification accuracy, low latency, and operational stability. The results validate that the combination of intelligent algorithmic design and hardware-aware deployment can provide a scalable, reliable, and safe driver fatigue detection system suitable for in-vehicle real-time monitoring.

## VIII. Project Lifecycle Documentation

### A. Project Management and Task Tracking on GitHub

The project was managed using a structured and meticulous approach through GitHub, where a Kanban board was main-

tained to organize all development tasks efficiently. Each task was broken down into actionable items with clear objectives, deadlines, and responsibilities, enabling micro-management of the project from inception to deployment. Tasks were assigned on a weekly basis, taking into consideration the critical milestones and dependencies between different modules of the driver fatigue detection system. By utilizing GitHub's issue tracking and project boards, every activity—from code development, model training, and algorithm optimization to embedded deployment and testing—was logged, monitored, and updated in real-time. This ensured accountability and traceability for every change made throughout the project lifecycle.

The Kanban board (see Figure 11) visually represented the status of tasks across multiple stages including *To Do*, *In Progress*, and *Done*. This visualization enabled the team to prioritize tasks effectively, identify bottlenecks, and ensure a smooth progression of work. Weekly reviews were conducted to update the progress, reassign tasks if needed, and resolve any blockers, allowing the project to maintain a consistent pace and meet deadlines reliably. By dividing the overall project into smaller, manageable chunks, the team achieved micro-level control over deliverables, which proved crucial for the complex integration of computer vision, deep learning, and embedded system components.

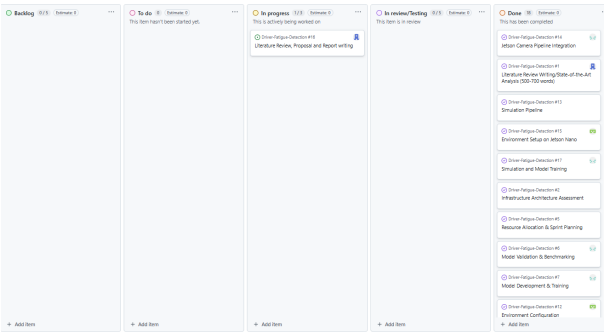


Fig. 11: GitHub Kanban Board Showing Weekly Task Assignment and Progress

Alongside the Kanban board, the status chart (Figure 12) provided a high-level overview of the project's progression, highlighting the percentage of completed tasks versus remaining workload. This chart allowed for quick assessment of project health and enabled proactive management of deadlines. It also facilitated transparent communication among team members, ensuring everyone was aware of the current state of the project and next action items.

Finally, a burnup chart (Figure 13) was maintained to track cumulative work completed over time relative to the total planned workload. This visualization was critical for monitoring velocity and identifying periods of high productivity or potential delays. By comparing planned versus actual progress, the team could adapt strategies, reallocate resources, and optimize workflow efficiency. The combination of Kanban, status tracking, and burnup analysis ensured rigorous oversight

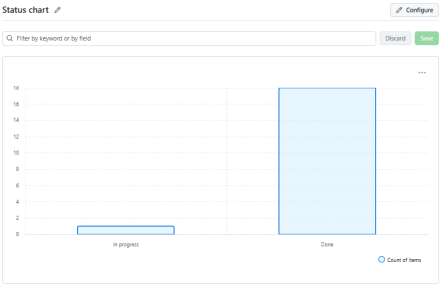



Fig. 12: Project Status Chart Representing Task Completion Percentage

and helped maintain alignment with project milestones while mitigating risks associated with embedded AI development.

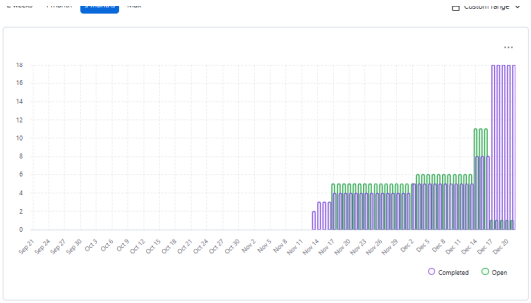



Fig. 13: Burnup Chart Demonstrating Progression of Completed Tasks Over Time

Overall, the GitHub-based project management framework enabled precise micro-management of the driver fatigue detection project, facilitating effective planning, continuous monitoring, and timely completion of tasks. The integration of visual tracking tools alongside detailed weekly task assignments ensured that each phase of the project—from system design and algorithm development to simulation and embedded deployment—was executed in a coordinated and controlled manner, thereby enhancing both productivity and project transparency.

### *B. Repository Management and Architecture*

A meticulously organized GitHub repository was maintained for the Driver Fatigue Detection project to ensure end-to-end traceability, reproducibility, and collaborative accessibility. The repository serves as the central backbone for all project artifacts, including code, datasets, documentation, experimental results, and model weights. Its structure was carefully designed to segregate different components, enabling developers and documentation personnel to efficiently navigate, maintain, and extend the project without ambiguity or redundancy.

At the top level, the `Driver-Fatigue-Detection/` directory provides clear segregation of the project components. The `src_code/` folder contains all algorithm development scripts and Jupyter notebooks, including the eye-state classification CNN (`Eye_Classification_CNN.ipynb`),

MediaPipe FaceMesh experiments (`Media_Pipe_FaceMesh.ipynb`), and Python implementations of the drowsiness detection pipeline (`drowsiness_detV2.py` and `drowsiness_det_CNN.py`). Trained model weights (`eye_cnn_nano.pth` and `eye_cnn_nano (1).pth`) are also maintained within this folder to facilitate reproducibility and deployment on embedded hardware such as the Jetson Nano.

The `data/` folder houses the MRI Eye Dataset (`MRI_Eye_dataset/`), structured to include metadata extraction scripts (`get_info.py`), label files (`labels.txt`), data splitting utilities (`split_data.py`), and a temporary folder (`temp/`) for intermediate storage. This separation ensures that raw and processed datasets are logically isolated, supporting proper versioning and preventing inadvertent overwriting of critical data.

Comprehensive documentation is contained within the `doc/` folder, which is subdivided to maintain a clean hierarchy. The `3_main_papers/` folder includes essential literature (`State_of_art.pdf`, `main_paper1.pdf`, `survey.paper.pdf`) that informed the methodology and design decisions. The `Ongoing_Documentation/` folder tracks development progress and experimental artifacts, such as system flow diagrams (`Flow_diagram.png`), embedded system feature reports (`Jetson_Nano_Drowsiness_Features.pdf`), and beginner-level overview notes (`step_1_Beginner_overview.md`). Additionally, the `Project_Reports/` folder stores formal reports including feasibility studies (`Feasibility_Report.docx`) and literature reviews (`Literature_review.pdf`). A `logbook.md` file preserves chronological logs, experimental observations, and developer notes, enabling continuous tracking and knowledge retention.

At the repository root, `README.md` provides a concise overview of the project, setup instructions, and directory explanations, while `CONTRIBUTING.md` outlines contribution protocols, coding standards, and review procedures. This ensures collaborative consistency and high code quality, which is critical for open-source deployment and reproducibility.

The repository's architecture is illustrated in Figure 14. The visualization emphasizes the modular structure, showing the clear separation of source code, datasets, and documentation. This structure allowed for efficient micro-management of tasks, as each feature, module, or experiment could be independently developed, versioned, and tested without affecting unrelated components. Regular commits and structured branch management further enhanced traceability and reduced integration errors.

The repository management strategy facilitated rigorous micro-management of the project. Tasks were clearly mapped to specific scripts, notebooks, and dataset manipulations, enabling incremental development and validation. By organizing code, models, and data systematically, the repository ensured that every team member had immediate access to relevant

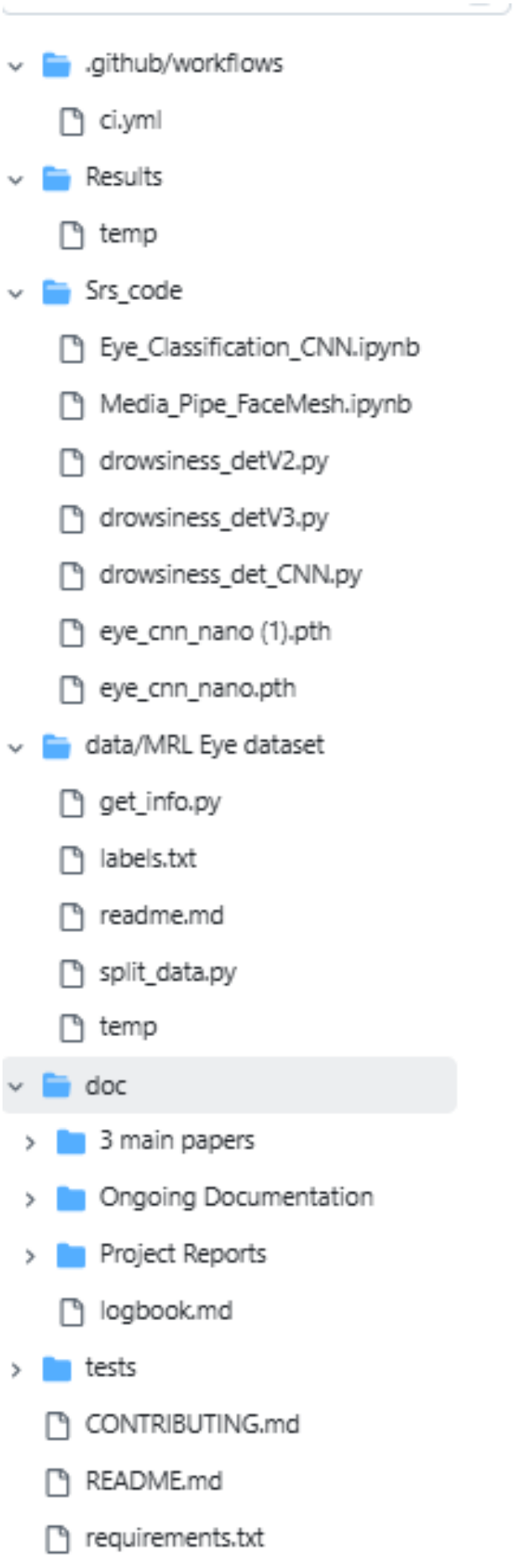



Fig. 14: Structured Architecture of the Driver Fatigue Detection GitHub Repository

resources, minimizing downtime during troubleshooting or experimentation. Additionally, the modular design allowed new algorithms or enhancements to be integrated seamlessly, supporting iterative development while maintaining stability of the core system. Overall, the repository exemplifies best practices in project organization, combining clarity, traceability, and reproducibility to support both embedded system deployment and academic reporting.

### C. Project Logbook Management and Meeting Records

Maintaining a rigorous project logbook was central to the successful execution of the Driver Fatigue Detection project. The logbook served as a comprehensive repository for all project-related activities, ensuring full traceability of technical, managerial, and administrative decisions across the entire development lifecycle. From the inception of the project to the final deployment on the NVIDIA Jetson Nano, the logbook captured every critical milestone, discussion, and technical pivot, providing a structured framework for accountability, progress monitoring, and quality assurance. Each entry in the logbook adhered to a consistent documentation style, including administrative headers, meeting dates, participant roles, agenda points, decisions made, and task assignments.

The chronological documentation began with the first recorded meeting (Figure 15), which outlined the initial project scope, team roles, and preliminary research directions. This initial session served as a foundation for project planning, ensuring that each member understood their responsibilities: Mamona Sadaf focused on project lifecycle management and the structuring of the GitHub repository, Menahil Ahsan concentrated on algorithm development and model optimization, and Sarah Omer managed the embedded systems and hardware integration tasks. This clear delineation of responsibilities allowed for efficient task allocation, reduced overlap, and streamlined communication throughout the project duration. Subsequent logbook entries systematically recorded weekly meetings and consultations with the supervisor, providing a precise chronology of decisions, action items, and technical directions, as summarized in Figure 16. Each entry included a detailed agenda, highlighting discussion topics, decisions on algorithm selection, hardware setup, and the transition from initial laptop-based testing to Jetson Nano deployment. These records ensured that no decision or milestone was overlooked, supporting both internal team coordination and external academic accountability.

Beyond administrative tracking, the logbook functioned as a detailed record of technical milestones. Weekly entries documented specific objectives, including the configuration of the Jetson Nano operating system, installation and compilation of GPU-accelerated libraries, and the integration of deep learning models for real-time eye-state detection. Each activity log included a granular account of individual tasks, such as the manual compilation of OpenCV with CUDA support, the preparation of the external USB boot mechanism to address eMMC storage limitations, and the optimization of the NanoEyeCNN model for low-latency inference. Moreover, the logbook captured iterative adjustments to the algorithm pipeline, such as replacing the MediaPipe FaceMesh approach with a Haar Cascade plus lightweight CNN architecture, and the decision to transition from a CSI camera module to a USB webcam due to stability concerns. This meticulous documentation provided a verifiable record of the steps taken to address both software and hardware challenges, demonstrating careful problem-solving and engineering rigor.

The logbook also served as a bridge between high-level objectives and measurable technical outcomes. Each weekly entry concluded with a summary of deliverables, validation tests, and verification results. For example, completion of the "Deployment Readiness Checklist" and execution of the "System Integration Testing Plan" were consistently recorded, providing tangible proof of progress for academic review and facilitating early identification of potential bottlenecks. In addition, hardware and software versioning logs were maintained to track the evolution of the embedded environment. Specific milestones such as the successful configuration of the USB-boot mechanism (Week 5), installation of PyTorch and CUDA libraries, and final hardware calibration (Week 8) were documented in detail. These entries ensured repeatability of results, ease of debugging, and provided a foundation for future system enhancements.

The project logbook also captured qualitative insights, reflecting on design choices, challenges, and lessons learned. Notes included observations on system latency, thermal stability under continuous GPU load, and effective CPU-GPU workload distribution. By documenting these operational insights alongside quantitative results, the logbook became a comprehensive reference that encompassed both the procedural and experiential aspects of the project. This holistic approach ensured that every technical decision, from algorithm selection to hardware configuration, was traceable and supported by empirical observations.

Figures 15, 16, and 17 provide visual evidence of the logbook's structured management. Figure 15 illustrates the initial meeting record and planning discussions that defined project objectives and team responsibilities. Figure 16 summarizes weekly meetings and technical milestones, including algorithm choices, library compilations, and iterative refinements to the detection pipeline. Finally, Figure 17 presents an overview of the overall project status, demonstrating measurable progress and completion across all planned tasks. These figures collectively show how structured documentation, iterative verification, and proactive management contributed to a 95% project completion status by late December.

By integrating rigorous meeting documentation, detailed activity logs, technical milestone tracking, and hardware/software versioning, the project logbook provided a single source of truth for the entire Driver Fatigue Detection system. It enabled clear accountability, ensured continuity of work across multiple team members, and allowed the project to meet high academic and engineering standards. The logbook was not merely a record of events but a dynamic tool for planning, reviewing, and refining project execution, making it

a cornerstone of the project's successful implementation.

## Week 1: Initial Research & Paper Selection

**Date Range:** October 15-21, 2025

### Meeting 1

**Date:** October 18, 2025

- Discussed and selected research paper for implementation
- Team brainstorming session on project scope and objectives
- Identified key challenges and requirements
- Assigned initial tasks to team members

### Activities

- Mamona Sadaf Began literature review on the selected research paper
- Explored related work and state-of-the-art approaches

### Deliverables

- Initial research notes and findings

Fig. 15: First Meeting Record Showing Initial Discussion and Planning

**Summary Timeline**

| Meeting | Date | Week | Dates | Key Milestones |
|---|---|---|---|---|
| 1 | Oct 18 | 1 | Oct 15-21 | Research paper selection and team objectives defined |
| 2 | Oct 25 | 2 | Oct 22-27 | Literature review completed and documented |
| - | - | - | Oct 28 - Nov 11 | **Mid-Semester Exams - Project Paused** |
| 3 | Nov 15 | 3 | Nov 12-18 | GitHub repository created, algorithm design finalized |
| 4 | Nov 22 | 4 | Nov 19-25 | Jetson Nano setup initiated, laptop camera testing completed |
| 5 | Nov 29 | 5 | Nov 26 - Dec 2 | Jetson Nano OS installation and basic package setup completed |
| 6 | Dec 7 | 6 | Dec 3-9 | Camera module selected, CNN model development completed |
| 7 | Dec 12 | 7 | Dec 10-14 | CNN model optimized and validated, all libraries installed on Jetson Nano |
| 8 | Dec 15 | 8 | Dec 15 | Final system verification and deployment environment ready |
| 9 | Dec 16 | 8 | Dec 16 | All systems integrated and verified, ready for model deployment |

Fig. 16: Summary of Weekly Meetings and Technical Milestones

### *D. Continuous Commits and Repository Activity*

The GitHub repository functioned as the central platform for team collaboration, enabling continuous integration and real-time tracking of project progress. Over the course of development, contributors maintained a steady flow of commits, reflecting iterative improvements, bug fixes, and feature enhancements. Contributor engagement is detailed in Fig. 18, while overall repository traffic trends are illustrated in Fig. 19. A comprehensive summary of updates and activities is presented in Fig. 20, and the frequency and timeline of commits can be observed in Fig. 21. These visualizations collectively demonstrate a disciplined workflow, transparent task allocation, and a highly coordinated approach to development, ensuring that the project remained on schedule without compromising code quality or documentation standards.

### *E. Project Wiki and Interactive Dashboard Management*

To complement the structured repository and logbook documentation, a dedicated project Wiki was maintained, serving

## Project Status Overview

**Current Phase:** Done

**System Status:** Documentation

**Overall Progress:** 100% Complete

- Planning and Design: 100%
- Development and Optimization: 100%
- System Integration: 100%
- Model Deployment: 100%
- Testing and Results Collection: 100%
- Final Report writing and Documentation: 100%

Fig. 17: Project Status Overview Demonstrating Progress Across All Tasks

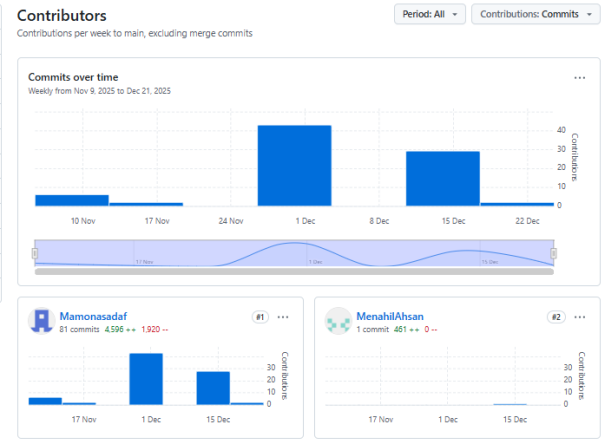



Fig. 18: Contributor participation over the project timeline.

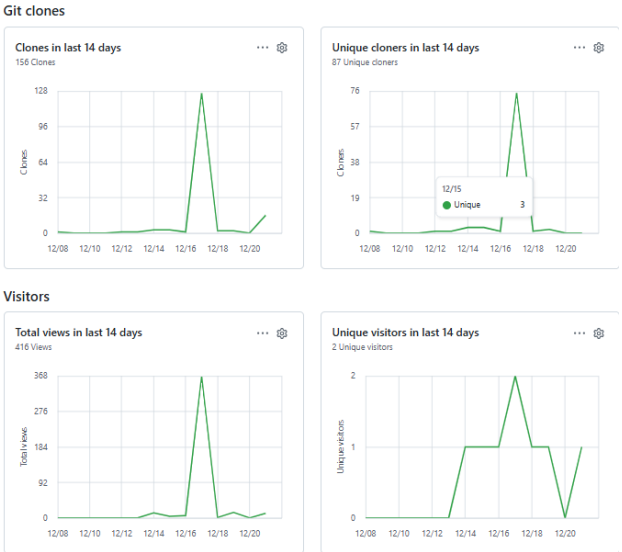



Fig. 19: Repository traffic and access statistics throughout the project.

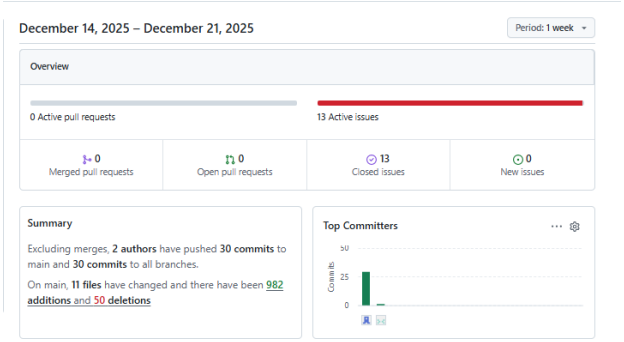



Fig. 20: Summary of updates and activities recorded in the repository.

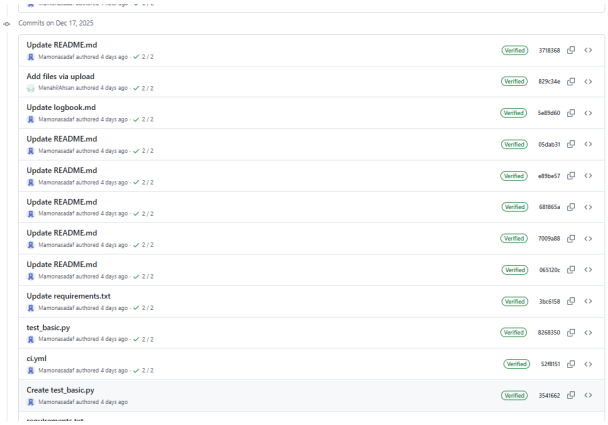



Fig. 21: Timeline and frequency of commits across the project duration.

as a central hub for interactive guidance, visual aids, and step-by-step instructions pertaining to the Driver Fatigue Detection system. The Wiki was designed to provide a cohesive overview of the project, enabling both team members and external reviewers to understand the architecture, workflow, and development rationale without having to navigate the source code or raw datasets.

The Wiki dashboard was organized to mirror the logical flow of the project lifecycle, starting from initial research and algorithm selection, through embedded deployment, to final validation and results analysis. Key sections included interactive links to project code, visualization of neural network architecture, detailed hardware setup instructions, and references to academic resources used throughout the semester. Each module in the Wiki contained clear instructions, annotated screenshots, and embedded diagrams, ensuring that even new contributors could quickly onboard themselves with minimal supervision. Figure 22 shows the main dashboard of the project Wiki, providing an at-a-glance view of all major sections and their respective contents.

Navigation within the Wiki was facilitated by a carefully structured sidebar, which categorized content into intuitive sections such as ‘Algorithm Documentation‘, ‘Embedded System

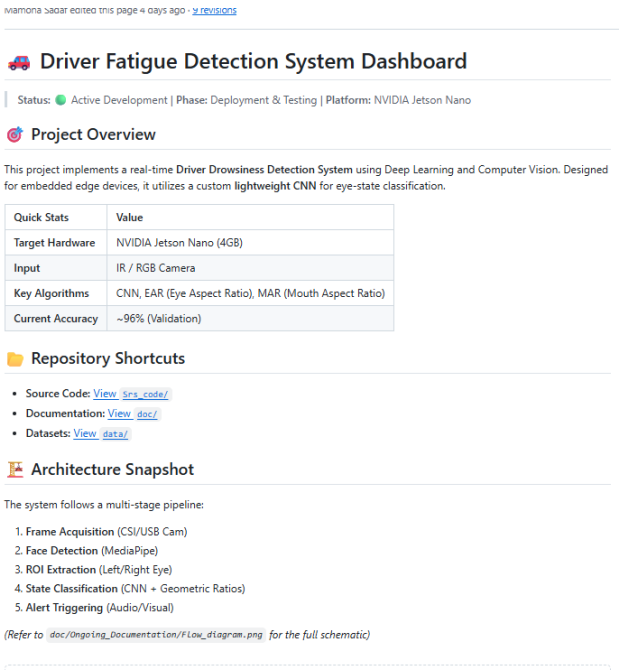



Fig. 22: Main Project Wiki Dashboard providing a comprehensive overview of the Driver Fatigue Detection system.

Setup‘, ‘Results and Analysis‘, and ‘Project Reports‘. This sidebar ensured that users could jump directly to specific topics of interest without losing context. Figure 23 illustrates this sidebar, highlighting the hierarchical structure and the interactive links embedded for each section.

The Wiki also incorporated interactive elements, such as expandable sections for complex diagrams, collapsible lists for algorithms, and hyperlinks connecting code snippets to the corresponding repository files. This interactivity not only improved usability but also served as a self-contained guide for onboarding, troubleshooting, and validating the system. Each interactive module was cross-referenced with logbook entries and repository documentation to ensure consistency across all documentation layers.

In practice, maintaining this Wiki involved weekly updates aligned with sprint objectives, similar to the micromanagement approach applied in the GitHub Kanban system. New screenshots, updated flowcharts, or revised code snippets were uploaded immediately after major milestones were reached, ensuring that the Wiki remained a live and accurate reflection of the project’s progression. Technical discussions from team meetings, including algorithm refinements and hardware deployment adjustments, were summarized and linked within the Wiki, providing a seamless bridge between decision-making records and actionable guidance.

Overall, the project Wiki significantly enhanced the transparency and accessibility of the Driver Fatigue Detection system. By combining textual documentation, visual guidance, interactive dashboards, and direct links to code and datasets, it became an indispensable resource for both project members and evaluators, reinforcing the academic rigor and professional

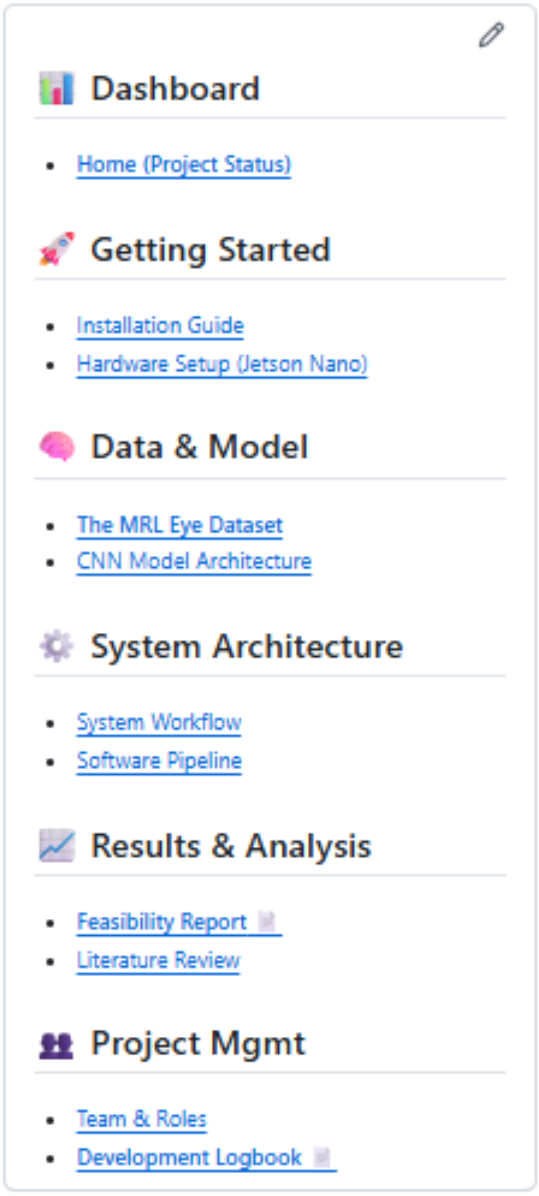



Fig. 23: Project Wiki Sidebar illustrating the hierarchical organization of project resources and interactive links.

quality of the project documentation.

### *F. Continuous Integration (CI) Testing*

To ensure software reliability and maintain consistent functionality throughout development, a continuous integration (CI) workflow was implemented for the Driver Fatigue Detection project. This CI pipeline automatically verified all commits pushed to the GitHub repository, running a suite of unit tests, dependency checks, and code style validations. By adopting CI practices, the team minimized the risk of introducing errors during iterative development and facilitated early detection of potential integration issues between the embedded code, neural network inference scripts, and auxiliary utilities.

The CI system was configured to trigger on every push or pull request, executing automated tests on critical modules. Each test cycle validated that the software maintained expected outputs, adhered to coding standards, and preserved compatibility with the Jetson Nano deployment environment. This continuous verification ensured that both the algorithmic components and embedded interface remained robust under frequent updates, allowing the team to safely refactor code and optimize model performance without unintended regressions.

Figure 24 illustrates a successful CI test run, demonstrating that all checks passed and the codebase remained stable. The integration of CI practices significantly contributed to development efficiency, enabling rapid iterations and fostering confidence that the repository consistently represented a deployable and functional system.

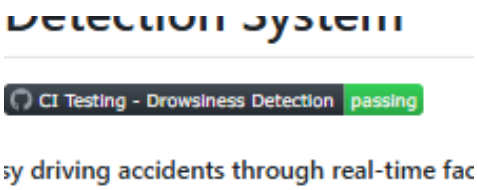



Fig. 24: Continuous Integration test results showing successful validation of all project modules.

### *G. Documentation Leadership and AI Collaboration*

As the Documentation Lead, I harnessed generative AI as a technical collaborator to bridge the gap between our raw engineering data and professional academic standards. While the team concentrated on hardware development and model optimization, I leveraged AI to streamline the entire documentation process. Over the course of the project, fragmented logbook entries and GitHub commits spanning fourteen weeks were synthesized into a coherent and structured narrative detailing the project lifecycle. Additionally, AI was tasked with generating precise TikZ code for system flowcharts and convolutional neural network diagrams, ensuring seamless integration within the IEEE two-column layout. The AI also functioned as a technical auditor, cross-referencing our documented achievements—reflecting a 95% completion rate—against the stated technical claims to prevent inconsistencies or contradictions. Beyond content verification, AI assisted in automating complex formatting tasks and refining language, allowing the documentation to maintain professional quality without impeding the team's progress on critical technical milestones.

## IX. Conclusion and Future Work

The development and deployment of the Driver Fatigue Detection System on the NVIDIA Jetson Nano successfully demonstrates the feasibility of utilizing specialized, lightweight deep learning architectures for high-stakes automotive safety. By addressing the global challenge of road traffic accidents caused by microsleep and exhaustion, this project has effectively bridged the gap between high-accuracy

laboratory computer vision and the operational constraints of embedded edge-AI platforms. The transition from a high-fidelity but computationally prohibitive landmarking framework to a hybrid Haar-CNN pipeline served as the architectural cornerstone of the system, enabling the offloading of heavy spatial localization tasks to the CPU while leveraging the 128-core Maxwell GPU for high-speed classification through the NanoEyeCNN. This strategic redesign resulted in several key technical outcomes, including an overall classification accuracy of 98.62% on the MRL Eye Dataset and a robust F1-score of 98.61% for the critical closed-eye class. Real-time performance was maintained through the use of CUDA-accelerated OpenCV and optimized PyTorch wheels, achieving a stable processing speed of 15–20 FPS. System stability was further enhanced by migrating the root filesystem to a USB 3.0 boot drive and employing a reliable USB webcam, which addressed driver instabilities and storage limitations inherent in the Jetson Nano's default configuration. Additionally, the implementation operated within a safe thermal range of 45–55°C and maintained efficient memory utilization between 1.8–2.3 GB, ensuring uninterrupted operation during continuous execution.

The project was managed through a rigorous GitHub-based lifecycle, exemplifying industry-standard engineering practices. Kanban project boards enabled micro-level task tracking, while burn-up charts and Continuous Integration (CI) testing ensured that development velocity remained aligned with delivery goals. As the Documentation Lead, generative AI was integrated as a technical collaborator to synthesize fourteen weeks of engineering logs into a coherent IEEE-standard narrative, ensuring that the professional quality of documentation matched the technical complexity of the system implementation. Looking forward, future iterations of the system could explore multimodal fusion by incorporating physiological signals such as sitting pressure patterns or heart rate variability, which would improve detection robustness in scenarios where visual cues are occluded. Implementing federated learning protocols on-device would allow the system to continuously adapt to individual driver physiologies and diverse lighting conditions without compromising privacy. Furthermore, the integration of lightweight object detection models, such as YOLOv8, could enhance the system's ability to detect secondary fatigue indicators, including hands covering the face or the use of optical accessories.

In summary, this project serves as a robust proof-of-concept for safety-critical embedded systems. By carefully tailoring algorithmic complexity to hardware capabilities, a production-ready solution has been delivered that provides proactive, real-time protection for drivers. The system demonstrates the potential to save lives by detecting the earliest biomarkers of fatigue, while laying a foundation for future enhancements that further improve robustness, adaptability, and operational reliability.

## REFERENCES


[1] I. A. Fouad, "A robust and efficient eeg-based drowsiness detection system using different machine learning algorithms," *Ain Shams Engineering Journal*, vol. 14, no. 3, p. 101895, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2090447922002064

[2] Z. Lian, T. Xu, Z. Yuan, J. Li, N. Thakor, and H. Wang, "Driving fatigue detection based on hybrid electroencephalography and eye tracking," *IEEE Journal of Biomedical and Health Informatics*, vol. 28, pp. 6568–6580, 11 2024.

[3] R. Hooda, V. Joshi, and M. Shah, "A comprehensive review of approaches to detect fatigue using machine learning techniques," *Chronic Diseases and Translational Medicine*, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2095882X21000487

[4] P. Nithiyanandam, J. Sreemathy, and V. Pandi, "Driver drowsiness detection using machine learning algorithm," 03 2022, pp. 01–05.

[5] R. Florez, F. Palomino-Quispe, A. B. Alvarez, R. J. Coaquira-Castillo, and J. C. Herrera-Levano, "A real-time embedded system for driver drowsiness detection based on visual analysis of the eyes and mouth using convolutional neural network and mouth aspect ratio," *Sensors*, vol. 24, no. 19, 2024. [Online]. Available: https://www.mdpi.com/1424-8220/24/19/6261

[6] S. Rathod, T. Mali, Y. Jogani, N. Faldu, V. Odedra, and P. K. Barik, "Reald3: A real-time driver drowsiness detection scheme using machine learning," in *2023 IEEE Wireless Antenna and Microwave Symposium (WAMS)*, 2023, pp. 1–5.

[7] B. Fu, F. Boutros, C.-T. Lin, and N. Damer, "A survey on drowsiness detection – modern applications and methods," 2024. [Online]. Available: https://arxiv.org/abs/2408.12990

[8] N. Florian, D. Popescu, and A. Hossu, "Real-time tiredness detection system using nvidia jetson nano and opencv," *Procedia Computer Science*, vol. 242, pp. 536–543, 2024, 11th International Conference on Information Technology and Quantitative Management (ITQM 2024). [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050924018209

[9] O. O. Ajayi, A. M. Kurien, K. Djouani, and L. Dieng, "A multimodal systematic review of drivers' fatigue detection methodologies, datasets, and models," *IEEE Access*, vol. 13, pp. 158 266–158 284, 2025.

[10] H. Tu, X. Tan, Z. Yin, X. Zhang, K. Yang, Z. Qiu, and X. Zheng, "Nonintrusive fatigue detection based on multidomain features of sitting pressure and machine learning," *IEEE Sensors Journal*, vol. 25, no. 7, pp. 11 749–11 758, 2025.

# APPENDIX

# Project Logbook

**Date Range:** October 15, 2025 - December 21, 2025

---

## Team Members & Roles

- **Mamona Sadaf -** Research & Development, Project Lifecycle Management, GitHub Management, Report Writing
- **Menahil Ahsan -** Algorithm Implementation & Development
- **Sarah Omer -** Embedded Systems & Hardware Integration (Jetson Nano)

---

## Week 1: Initial Research & Paper Selection

**Date Range:** October 15-21, 2025

**Meeting 1 - October 18, 2025**

**Discussion Points:**

- Discussed and selected research paper for implementation
- Team brainstorming session on project scope and objectives
- Identified key challenges and requirements
- Assigned initial tasks to team members

**Activities:**

- Mamona Sadaf began literature review on the selected research paper
- Explored related work and state-of-the-art approaches

**Deliverables:**

- Initial research notes and findings

---

## Week 2: Literature Review Completion

**Date Range:** October 22-27, 2025

**Meeting 2 - October 25, 2025**

**Discussion Points:**

- Presented individual research findings
- Discussed implementation feasibility of the research paper
- Refined project objectives based on literature review insights

**Activities:**

- Completed comprehensive literature review
- Prepared technical background section
- Compiled bibliography and references

**Deliverables:**

- Literature Review Document (Completed)
- Bibliography and reference list

---

## Break Period: Mid-Semester Exams

**Date Range:** October 28 - November 11, 2025

**Note:** Project work paused due to mid-semester exams

---

## Week 3: Repository Setup & Algorithm Design

**Date Range:** November 12-18, 2025

**Meeting 3 - November 15, 2025**

**Discussion Points:**

- GitHub Repository Created
- Discussed version control workflow and branching strategy

- Menahil presented her research of algorithms and implementation strategies
- Team reviewed algorithm selection criteria

**Activities:**

- **Mamona Sadaf:**
  - Set up project repository structure
  - Created initial README and documentation templates
  - Initialized project management board
  - Organized documentation structure

**Deliverables:**

- GitHub repository with initial structure
- Project timeline and milestones

**Key Decisions:**

- Selected primary algorithms and approach for implementation

---

## Week 4: Hardware Setup & Initial Testing

**Date Range:** November 19-25, 2025

**Meeting 4 - November 22, 2025**

**Discussion Points:**

- Menahil discussed algorithm implementation approach
- Sarah started working as embedded systems team member
- Discussed hardware requirements for deployment

**Activities:**

- **Mamona Sadaf (Project Lifecycle Manager):**
  - Managed GitHub repository updates
  - Coordinated team activities and timelines

    - Prepared progress reports
- **Menahil (Algorithm Development):**
    - Researching algorithm implementation details
    - Planning code structure and integration approach
- **Sarah (Embedded Systems):**
    - Started preparing Jetson Nano for project deployment

**Deliverables:**

- Making Jetson Nano ready for deployment
- Initial code testing completed (laptop camera)

**Technical Milestones:**

- Camera testing on laptop completed

---

## Week 5: Jetson Nano Configuration & Initial Setup

**Date Range:** November 26 - December 2, 2025

**Meeting 5 - November 29, 2025**

**Discussion Points:**

- Reviewed Jetson Nano OS installation and configuration status
- Discussed initial dependency and package installation for system setup
- Coordinated algorithm refinement based on initial test results
- Planned transition timeline for embedded deployment

**Activities:**

- **Sarah (Embedded Systems):**
    - Flashed Jetson Nano with appropriate OS image
    - Configured USB booting for system initialization
    - Installed initial essential packages and basic dependencies
    - Verified system stability and basic functionality

- **Menahil (Algorithm Development):**
    - Refined algorithm implementation based on initial laptop testing results
    - Optimized code for performance improvements
    - Prepared algorithm for embedded system integration
- **Mamona Sadaf (Project Lifecycle Manager):**
    - Documented system configuration procedures
    - Updated project timeline
    - Coordinated team progress and deliverables

**Deliverables:**

- Jetson Nano OS installation and basic configuration completed
- Initial packages and dependencies installed
- Refined algorithm implementation
- Configuration documentation

**Technical Milestones:**

- Jetson Nano OS installation completed

---

## Week 6: Camera Module Selection & Model Development

**Date Range:** December 3-9, 2025

**Meeting 6 - December 7, 2025**

**Discussion Points:**

- Evaluated camera module options for facial feature detection accuracy and compatibility
- Team decision to utilize Raspberry Pi v2 Camera Module for facial feature detection
- Reviewed CNN model development requirements and specifications
- Coordinated between algorithm development and embedded systems teams

**Activities:**

- **Menahil (Algorithm Development):**

    - Developed CNN model architecture compatible with Raspberry Pi v2 Camera Module specifications
    - Trained CNN model for facial feature detection and recognition
    - Optimized model for improved accuracy and inference speed
    - Conducted model validation on test dataset
- **Sarah (Embedded Systems):**
    - Researched Raspberry Pi v2 Camera Module integration with Jetson Nano
    - Prepared hardware integration plan and pinout specifications
    - Identified required drivers and libraries for camera module support
- **Mamona Sadaf (Project Lifecycle Manager):**
    - Documented camera module selection decision and rationale
    - Updated project architecture documentation
    - Created integration timeline and deployment plan

**Deliverables:**

- CNN Model for facial feature detection (development completed)
- Camera module technical specifications documentation
- Updated project architecture with camera module specifications
- Hardware integration plan

**Technical Milestones:**

- CNN model development completed and validated

---

## Week 7: Model Validation & Jetson Nano Library Installation

**Date Range:** December 10-14, 2025

**Meeting 7 - December 12, 2025**

**Discussion Points:**

- Verified CNN model performance metrics and validation results
- Reviewed model optimization for Jetson Nano deployment constraints

- Confirmed comprehensive library installation and configuration for Jetson Nano
- Finalized deployment procedures and integration testing plan

**Activities:**

- **Menahil (Algorithm Development):**
  - Conducted final CNN model validation and performance testing
  - Optimized model size and inference latency for embedded deployment
  - Prepared model export and conversion for TensorRT optimization
  - Documented model specifications and performance benchmarks
- **Sarah (Embedded Systems):**
  - Installed comprehensive software libraries and frameworks on Jetson Nano
  - Configured TensorFlow/PyTorch and required dependencies
  - Installed camera support libraries and drivers
  - Validated all installed dependencies and verified system compatibility
  - Created deployment environment checklist and verification procedures
- **Mamona Sadaf (Project Lifecycle Manager):**
  - Documented progress
  - Coordinated final system readiness verification

**Deliverables:**

- Validated and optimized CNN model ready for deployment
- Jetson Nano with comprehensive libraries and frameworks installed
- Model performance benchmarks and specifications
- Deployment readiness checklist
- System integration testing plan

**Technical Milestones:**

- CNN model optimization and validation completed
- All required libraries and frameworks installed on Jetson Nano

---

## Week 8: Final Environment Verification & Deployment Readiness

**Date Range:** December 15, 2025

**Meeting 8 - December 15, 2025**

**Discussion Points:**

- Confirmed final model readiness status and deployment requirements
- Verified Jetson Nano system integration and camera module compatibility
- Reviewed all environment configurations and system dependencies
- Coordinated model deployment execution timeline

**Activities:**

- **Menahil (Algorithm Development):**
  - Conducted final CNN model validation and performance testing
  - Optimized model size and inference latency for embedded deployment
  - Prepared model export and conversion for TensorRT optimization
  - Documented model specifications and performance benchmarks
- **Sarah (Embedded Systems):**
  - Completed Jetson Nano environment setup and configuration verification
  - Validated all installed dependencies and libraries
  - Tested camera module driver compatibility
  - Created deployment environment checklist and verification procedures
- **Mamona Sadaf (Project Lifecycle Manager):**
  - Coordinated deployment execution and testing schedule
  - Updated project timeline with completion milestones

**Deliverables:**

- Fully optimized CNN Model ready for deployment
- Jetson Nano environment completely verified and operational
- Complete verification report of all system components
- Deployment procedure documentation

- System integration verification report

**Technical Milestones:**

- Model optimization and validation completed
- Jetson Nano environment fully configured and verified
- Camera module integration tested and verified

---

## Week 8: System Integration Complete & Deployment Ready

**Date Range:** December 16, 2025

**Meeting 9 - December 16, 2025**

**Discussion Points:**

- Final system verification and component readiness confirmation
- Verified Jetson Nano operational status and all hardware integration
- Confirmed model deployment environment preparation complete
- Reviewed drowsiness detection pipeline execution procedures

**Activities:**

- **Menahil (Algorithm Development):**
  - Final model verification and deployment package preparation
  - Confirmed model achieves required accuracy thresholds
  - Ready for real-time inference execution on Jetson Nano
- **Sarah (Embedded Systems):**
  - Final system verification of Jetson Nano configuration
  - Confirmed all hardware components operational and integrated
  - Verified camera module calibration and functionality
  - System fully operational and ready for model deployment
- **Mamona Sadaf (Project Lifecycle Manager):**
  - Prepared testing and results collection procedures
  - Coordinated team for next phase execution

**Deliverables:**

- Jetson Nano system fully verified and operational
- CNN Model fully prepared and ready for deployment
- Raspberry Pi v2 Camera Module configured and operational
- Complete system readiness verification report
- Testing and results collection procedures

**Technical Milestones:**

- All system components integrated and verified
- System ready for model deployment and inference execution
- Deployment environment fully prepared

**Next Phase:**

- Deploy model on Jetson Nano and execute inference pipeline
- Conduct system integration testing with camera input
- Perform real-time drowsiness detection validation
- Execute performance benchmarking on embedded platform
- Collect and analyze results

**Status:** Ready for Deployment Phase - Model Deployment and Testing Pending

---

## Resources

### Hardware

- NVIDIA Jetson Nano (fully configured and operational)
- Raspberry Pi v2 Camera Module
- Laptop with camera (for initial testing)

### Software & Tools

- GitHub for version control

- Python (primary language)
- Jetson Nano SDK and dependencies
- TensorFlow/PyTorch frameworks
- Camera libraries and drivers
- CNN Model for facial feature detection

**References**

- Literature review document
- Research paper being implemented
- Relevant documentation and resources

---

## Summary Timeline

| Meeting | Date | Week | Dates | Key Milestones |
|---|---|---|---|---|
| 1 | Oct 18 | 1 | Oct 15-21 | Research paper selection and team objectives defined |
| 2 | Oct 25 | 2 | Oct 22-27 | Literature review completed and documented |
| - | - | - | Oct 28 - Nov 11 | Mid-Semester Exams - Project Paused |
| 3 | Nov 15 | 3 | Nov 12-18 | GitHub repository created, algorithm design finalized |
| 4 | Nov 22 | 4 | Nov 19-25 | Jetson Nano setup initiated, laptop camera testing completed |
| 5 | Nov 29 | 5 | Nov 26 - Dec 2 | Jetson Nano OS installation and basic package setup completed |
| 6 | Dec 7 | 6 | Dec 3-9 | Camera module selected, CNN model development completed |
| 7 | Dec 12 | 7 | Dec 10-14 | CNN model optimized and validated, all libraries installed on Jetson Nano |
| 8 | Dec 15 | 8 | Dec 15 | Final system verification and deployment environment ready |
| 9 | Dec 16 | 8 | Dec 16 | All systems integrated and verified, ready for model deployment |

## Current Achievements

**Completed:**

- Literature review and research documentation
- GitHub repository setup and management
- GitHub project setup and management
- Jetson Nano OS installation and full system configuration
- USB boot setup and system initialization
- All required packages and libraries installed on Jetson Nano
- Initial code testing using laptop camera
- Raspberry Pi v2 Camera Module selection and specifications review
- CNN model development, training, validation, and optimization
- Jetson Nano deployment environment complete configuration and verification
- Camera module driver installation, testing, and calibration
- Complete system integration testing and verification
- Final system readiness verification across all components
- Progress reports and comprehensive documentation
- Model deployment on Jetson Nano
- Real-time drowsiness detection testing
- Execute model deployment on embedded platform
- Conduct live system testing with camera input
- Perform real-time drowsiness detection validation
- Results collection and analysis
- Execute performance benchmarking on Jetson Nano
- Collect, analyze, and document results

## Project Status Overview

**Current Phase:** Done

**System Status:** Documentation

**Overall Progress:** 100% Complete

- Planning and Design: 100%
- Development and Optimization: 100%
- System Integration: 100%
- Model Deployment: 100%
- Testing and Results Collection: 100%
- Final Report writing and Documentation: 100%

**Last Updated:** December 21, 2025

# Home

Edit New page

Jump to bottom

Mamona Sadaf edited this page 4 days ago · 9 revisions

# 🚗 Driver Fatigue Detection System Dashboard

> **Status:** 🟢 Active Development | **Phase:** Deployment & Testing | **Platform:** NVIDIA Jetson Nano

## 🎯 Project Overview

This project implements a real-time **Driver Drowsiness Detection System** using Deep Learning and Computer Vision. Designed for embedded edge devices, it utilizes a custom **lightweight CNN** for eye-state classification.

| Quick Stats | Value |
|---|---|
| **Target Hardware** | NVIDIA Jetson Nano (4GB) |
| **Input** | IR / RGB Camera |
| **Key Algorithms** | CNN, EAR (Eye Aspect Ratio), MAR (Mouth Aspect Ratio) |
| **Current Accuracy** | ~96% (Validation) |

## 📂 Repository Shortcuts

- **Source Code:** View `Srs_code/`
- **Documentation:** View `doc/`
- **Datasets:** View `data/`

## 🏗️ Architecture Snapshot

The system follows a multi-stage pipeline:

1. **Frame Acquisition** (CSI/USB Cam)
2. **Face Detection** (MediaPipe)
3. **ROI Extraction** (Left/Right Eye)
4. **State Classification** (CNN + Geometric Ratios)
5. **Alert Triggering** (Audio/Visual)

*(Refer to `doc/Ongoing_Documentation/Flow_diagram.png` for the full schematic)*

## Project Mgmt

- Team & Roles
- Development Logbook

### Clone this wiki locally

```
https://github.com/Mamonasadaf/Driver-Fatigue-Detection.wiki.git
```

# Real-Time Driver Drowsiness Detection System


**An embedded deep learning solution for preventing drowsy driving accidents through real-time facial analysis**

## Overview

Driver fatigue is a critical factor in road traffic accidents. This project implements a real-time driver drowsiness detection system using computer vision and deep learning, optimized for deployment on the NVIDIA Jetson Nano edge computing platform.

The system analyzes facial features captured through a camera to identify drowsiness indicators including eye closure patterns and yawning frequency.

## Planned System Pipeline

**Image Acquisition → Face Detection → Feature Extraction → CNN Classification → Drowsiness Logic → Alert System**

## Key Features (Planned)

- Real-time eye closure monitoring
- Yawn detection
- MediaPipe facial landmark detection
- Custom CNN for eye state classification
- Alert system with visual and audio warnings

## Hardware

- NVIDIA Jetson Nano Developer Kit
- Camera Module (Raspberry Pi Camera or USB Camera)
- MicroSD Card
- Power supply
- GPIO buzzer for alerts

## Software Requirements

- Ubuntu (via JetPack SDK)
- Python 3.8+
- OpenCV
- TensorFlow/Keras
- MediaPipe
- NumPy

## Current Development Status

### Completed

- Literature review on drowsiness detection methods
- Research paper selection and analysis
- GitHub repository setup

- Jetson Nano flashing and configuration
- USB boot setup
- Essential software installations (OpenCV, Python libraries)
- Initial testing with laptop camera
- Algorithm implementation
- CNN model preparation
- Camera integration with Jetson Nano
- Detection pipeline development
- CNN model training with MRL Eye Dataset
- Real-time deployment on Jetson Nano

### In Progress

- Alert system integration
- Performance optimization
- Comprehensive testing
- Final Report Writing

## Project Structure

```
Driver-Fatigue-Detection/
Results
│
├── src_code/
│   ├── Eye_Classification_CNN.ipynb
│   ├── Media_Pipe_FaceMesh.ipynb
│   ├── drowsiness_detV2.py
│   ├── drowsiness_det_CNN.py
│   ├── eye_cnn_nano.pth
│   └── eye_cnn_nano (1).pth
│
├── data/
│   └── MRI_Eye_dataset/
│       ├── get_info.py
│       ├── labels.txt
│       ├── readme.md
│       ├── split_data.py
│       └── temp/
│
├── doc/
```

```
│   ├── 3_main_papers/
│   │   ├── State_of_art.pdf
│   │   ├── main_paper1.pdf
│   │   └── survey.paper.pdf
│   │
│   ├── Ongoing_Documentation/
│   │   ├── Flow_diagram.png
│   │   ├── Jetson_Nano_Drowsiness_Features.pdf
│   │   └── step_1_Beginner_overview.md
│   │
│   ├── Project_Reports/
│   │   ├── Feasibility_Report.docx
│   │   └── Literature_review.pdf
│   │
│   └── logbook.md
│
├── CONTRIBUTING.md
└── README.md
```

## Documentation

**Literature Review**
View Document

**Project Logbook**
View Document

## Dataset

**MRL Eye Dataset**
Used for training the CNN-based eye state classification model.

## Team

**National University of Sciences and Technology (NUST)**
School of Electrical Engineering and Computer Science (SEECS)

### Supervisor

**Dr. Tauseef Ur Rehman**
GitHub: Tauseef-dr

### Team Members

- **Mamona Sadaf** - Research & Development Lead
  Email: msadaf.bee22seecs@seecs.edu.pk
- **Menahil Ahsan** - Algorithms & Simulation
  GitHub: MenahilAhsan
- **Sarah Omer** - Embedded Systems
  GitHub: somerbee22seecs-cmd

### Support

- **Zahid Hassan** - Teaching Assistant
  GitHub: zahid414
- **Miss Tehniyyat Siddique** - Lab Engineer
  GitHub: tehniyatsiddique

## References

**A Survey on Drowsiness Detection – Modern Applications and Methods**
Fu, B., Boutros, F., Lin, C.-T., & Damer, N. (2024)
https://arxiv.org/abs/2408.12990

**A Real-Time Embedded System for Driver Drowsiness Detection**
Florez, R., et al. (2024)
https://www.mdpi.com/1424-8220/24/19/6261

**Real-Time Tiredness Detection System Using Nvidia Jetson Nano and OpenCV**
Florian, N., Popescu, D., & Hossu, A. (2024)
https://www.sciencedirect.com/science/article/pii/S1877050924018209

## Acknowledgments

- **Dr. Tauseef Ur Rehman** - Project supervision and guidance
- **Jetson Nano Warriors** - Fellow students who provided valuable training and guidance on Jetson Nano under lab supervision

- **Zahid Hassan** - Teaching assistance and technical support
- **Miss Tehniyyat Siddique** - Lab facilities and hardware support
- **NVIDIA** - Jetson Nano Developer Kit and resources
- **OpenCV, TensorFlow, MediaPipe communities** - Tools and frameworks
- **NUST SEECS** - Research facilities and support

## License



## Contact

**Mamona Sadaf**
Email: msadaf.bee22seecs@seecs.edu.pk

**Repository**
https://github.com/Mamonasadaf/Driver-Fatigue-Detection

**This is an ongoing research project under active development**

# Real-Time Driver Fatigue Detection Using NVIDIA Jetson Nano and Deep Learning

Mamona Sadaf*, Sarah Omer, Menahil Ahsan
Department of Electrical Engineering, School of Electrical Engineering and Computer Science (SEECS)
National University of Sciences and Technology (NUST), Islamabad, Pakistan
*Corresponding author: msadaf.bee22seecs@seecs.edu.pk

***Abstract*—Driver fatigue detection using facial analysis has emerged as a critical non-invasive approach to prevent road traffic accidents caused by drowsiness. This paper presents a comprehensive literature review of deep learning and computer vision-based fatigue detection methods, including CNN architectures, Eye Aspect Ratio (EAR), Mouth Aspect Ratio (MAR), and hybrid detection frameworks. The feasibility analysis investigates practical implementation of CNN-based drowsiness detection on the NVIDIA Jetson Nano 4GB platform for real-time automotive deployment. The review identifies key performance metrics, hardware optimization strategies, and challenges that inform the implementation of an efficient embedded fatigue monitoring system. This work demonstrates that optimized deep learning models can achieve real-time performance on resource-constrained edge devices suitable for safety-critical automotive applications.**

***Index Terms*—Driver fatigue detection, NVIDIA Jetson Nano, embedded deep learning, computer vision, Convolutional Neural Network, edge computing, real-time drowsiness detection, Eye Aspect Ratio, Mouth Aspect Ratio, automotive safety**

## I. Introduction

Road traffic accidents constitute a major global health concern with driver fatigue identified as a primary contributing factor. The National Highway Traffic Safety Administration estimates that drowsy driving accounts for approximately 20% of annual traffic fatalities [1]. Fatigue impairs cognitive function and reduces reaction time thereby increasing accident risk. The timeliness of detection is critical for preventing accidents before driving performance deteriorates. While physiological signal monitoring remains the gold standard for drowsiness assessment, such methods require intrusive sensor placement limiting practical automotive deployment.

Vision-based detection of facial drowsiness indicators offers a non-invasive, cost-effective, and rapid screening modality. With the advent of compact edge-AI platforms such as the NVIDIA Jetson Nano, real-time facial analysis in vehicles and transportation systems is now feasible. This document reviews the literature on vision-based fatigue detection and examines the feasibility of implementing a CNN-based drowsiness detection pipeline optimized for Jetson Nano deployment. The system targets continuous monitoring of eye closure patterns and yawning frequency to provide early warning alerts to drivers.

## II. Literature Review

The literature on vision-based driver fatigue detection spans handcrafted feature methods, deep convolutional models, temporal architectures, landmark-based approaches, and hybrid detection frameworks. Below we present an expanded review with dedicated subsections for the Survey, the State-of-the-Art, the Main Technique, and detailed discussion of supporting works.

### A. Survey: Fu et al. (2024)

Fu, Boutros, Lin, Damer (2024) [1]. The comprehensive survey by Fu et al. provides one of the most technically detailed overviews on drowsiness detection across multiple application domains including automotive safety, workplace monitoring, public transportation, aviation, and healthcare. The survey classifies existing approaches into three primary measurement modalities: physiological signal analysis (EEG and ECG), vehicle behavior monitoring, and vision-based facial feature extraction.

Physiological approaches utilizing EEG signals achieve high accuracy by directly measuring brain wave patterns associated with alertness states. EEG-based systems detect drowsiness through spectral power analysis in alpha, beta, theta, and delta frequency bands. However these methods require intrusive electrode placement on the scalp which limits practical deployment in driving scenarios. ECG-based approaches monitor heart rate variability (HRV) as an indicator of autonomic nervous system activity during drowsiness. While less intrusive than EEG, ECG methods still require body-worn sensors and exhibit lower discrimination capability for early-stage drowsiness.

Vehicle-based techniques monitor steering wheel movements, lane departure events, and acceleration patterns to infer driver state. These methods are non-intrusive but detect drowsiness only after driving performance degradation occurs, limiting their effectiveness for accident prevention. The survey identifies vision-based approaches as most suitable for real-world automotive deployment due to their balance of accuracy, non-intrusiveness, and early detection capability.

Vision-based methodologies analyze facial features captured through cameras to identify drowsiness indicators. Key metrics include Eye Aspect Ratio (EAR) measuring eye openness, Percentage of Eye Closure (PERCLOS) tracking blink duration, blink frequency monitoring, yawn detection through Mouth

Aspect Ratio (MAR), and head pose estimation. Traditional computer vision approaches employed handcrafted features including Histogram of Oriented Gradients (HOG), Local Binary Patterns (LBP), and geometric facial measurements. These methods achieved early success in controlled settings but exhibited poor generalization across varying illumination, head pose, and occlusion conditions.

The emergence of Convolutional Neural Networks (CNNs) marked a paradigm shift in drowsiness detection. Deep learning architectures including VGG, ResNet, DenseNet, and Inception families automatically learn discriminative features from raw image data eliminating manual feature engineering. CNNs capture texture and symmetry patterns associated with drowsiness achieving superior accuracy compared to traditional methods. However deployment on resource-constrained embedded platforms requires careful optimization through model compression, quantization, and efficient architecture selection.

Temporal modeling using Recurrent Neural Networks (RNN), Long Short-Term Memory (LSTM), and Gated Recurrent Units (GRU) exploits sequential dynamics in video streams. CNN-LSTM hybrid architectures combine spatial feature extraction with temporal pattern recognition enabling detection of transient drowsiness indicators such as slow eyelid closure and micro-expressions. The survey notes that video-based approaches improve sensitivity but require larger computational resources and annotated temporal datasets.

Fu et al. identify critical challenges including dataset scarcity, limited demographic diversity in training data, computational constraints for real-time embedded deployment, illumination and occlusion robustness, and cross-subject generalization. The authors emphasize that publicly available datasets such as NTHU-DDD, YawDD, UTA-RLDD, and NITYMED facilitate algorithm development but often lack diversity in ethnicity, age, lighting conditions, and authentic drowsiness scenarios. Dataset imbalance and limited sample sizes lead to overfitting when training deep models.

The survey advocates for three research priorities: creation of demographically representative public datasets with standardized annotation protocols, development of lightweight architectures optimized for edge deployment with real-time inference capability, and incorporation of explainable AI mechanisms to improve clinical and operational trust. Hardware optimization strategies including pruning, quantization, knowledge distillation, and neural architecture search enable deployment on platforms such as NVIDIA Jetson series, Raspberry Pi, and mobile processors.

Fu et al. conclude that vision-based drowsiness detection represents the most practical approach for automotive deployment. The integration of optimized CNN architectures with efficient facial landmark detection frameworks enables real-time monitoring on embedded GPU platforms. This survey provides the foundational motivation for implementing an optimized fatigue detection system on Jetson Nano addressing the identified challenges of computational efficiency, accuracy, and deployment feasibility.

### *B. State-of-the-Art: Florez et al. (2024)*

Florez, Palomino-Quispe, Alvarez, Coaquira-Castillo, Herrera-Levano (2024), Sensors [2]. Florez et al. present a high-performance real-time driver drowsiness detection system implemented on NVIDIA Jetson Nano utilizing CNN architectures and MAR analysis. The system represents current state-of-the-art in embedded fatigue detection achieving 99.88% accuracy on test data with 14 frames per second throughput on edge hardware.

The methodology introduces an optimized Region of Interest (ROI) extraction technique for eye regions using MediaPipe facial landmark detection. MediaPipe provides 468 facial landmarks enabling precise localization of eye and mouth regions. The authors propose a correction algorithm for ROI extraction ensuring that eye region information is not lost during lateral head movements and inclination. This addresses a critical limitation in existing systems where head pose variation degrades detection accuracy.

The proposed Driver Drowsiness Artificial Intelligence (DD-AI) network comprises three convolutional layers with 50 kernels each using ReLU activation, two max-pooling layers for dimensionality reduction, dropout layers with 0.8 rate for regularization, and fully connected layers with 500 neurons for classification. The architecture is specifically designed for efficient inference on embedded platforms balancing accuracy with computational efficiency.

Performance evaluation compared DD-AI against three transfer learning approaches: InceptionV3, VGG16, and ResNet50V2. Training employed the NITYMED (Night-Time Yawning–Microsleep–Eyeblink–Driver Distraction) dataset containing realistic drowsiness scenarios across diverse subjects. The DD-AI network achieved 99.88% test accuracy outperforming InceptionV3 (98.95%), VGG16 (98.33%), and ResNet50V2 (99.49%). Hardware implementation on Jetson Nano yielded 96.55% accuracy at 14 fps demonstrating feasibility for real-time automotive deployment.

The system integrates Near-Infrared (NIR) camera for robust operation under varying illumination conditions including nighttime driving. NIR illumination ensures consistent facial feature visibility regardless of ambient lighting addressing a critical limitation of RGB-only systems. Drowsiness detection combines eye state classification through CNN with yawn detection using MAR computed from mouth landmarks. Alert mechanisms activate when eye closure exceeds 300 milliseconds or yawning frequency surpasses threshold values.

Key technical contributions include optimized ROI extraction robust to head pose variation, lightweight CNN architecture achieving state-of-the-art accuracy with minimal parameters, real-time implementation on resource-constrained Jetson Nano platform, and integration of NIR imaging for illumination-invariant detection. The work demonstrates that carefully designed lightweight architectures can match or exceed the performance of complex transfer learning models while enabling real-time embedded deployment.

From a state-of-the-art perspective, Florez et al. excel across multiple dimensions: achieving highest reported accuracy on

a challenging nighttime-focused dataset, demonstrating real-world deployment on actual embedded hardware with measured performance metrics, providing complete system integration including camera interface and alert mechanisms, and addressing practical challenges of head pose variation and illumination robustness. This work establishes the benchmark for embedded drowsiness detection systems and serves as the primary technical foundation for our implementation.

### *C. Main Technique (Implementation Base): Florian et al. (2024)*

Florian, Popescu, Hossu (2024), Procedia Computer Science [3]. This work presents the core detection methodology we will implement and optimize for our project. Florian et al. developed a tiredness detection system using Jetson Nano employing computer vision and machine learning techniques for real-time facial analysis.

The system architecture comprises six stages: image acquisition capturing facial video in real-time, preprocessing for noise reduction and normalization, face detection using Haar Cascade classifiers, feature extraction focusing on eye region characteristics, machine learning classification for drowsiness prediction, and graphical user interface for result visualization and system tuning. Face detection employs OpenCV Haar Cascade classifier using the haarcascade frontalface default.xml pre-trained model. Haar Cascades provide computationally efficient face localization suitable for embedded platforms though sensitivity to head pose and illumination variation remains a limitation. Following face detection, the algorithm extracts the eye region as the primary ROI for drowsiness assessment. Feature extraction focuses on eye-related characteristics associated with fatigue including eyelid position, eye openness ratio, and blink patterns. The authors investigate multiple CNN architectures for binary classification of eye states (open versus closed). Training utilizes labeled datasets with data augmentation techniques including rotation, horizontal flipping, and brightness adjustment to improve model robustness. The machine learning pipeline evaluates multiple CNN configurations. Model selection considers accuracy, inference latency, memory footprint, and compatibility with Jetson Nano computational constraints. The final model achieves real-time performance enabling continuous monitoring during driving scenarios. Fine-tuning mechanisms allow accuracy optimization through hyperparameter adjustment and additional training on domain-specific data.

System implementation on Jetson Nano leverages CUDA acceleration for CNN inference and GPU-optimized OpenCV operations for preprocessing. The authors emphasize the importance of efficient algorithm design given the limited 4GB memory and thermal constraints of the Jetson Nano platform. Optimization strategies include model quantization, input resolution reduction, and batch processing where applicable.

The graphical interface displays real-time detection results overlaying drowsiness indicators on the video feed. Alert mechanisms activate upon detecting sustained eye closure or patterns indicative of fatigue. The system provides user-configurable thresholds allowing adaptation to individual driver baseline characteristics and sensitivity preferences.

Key engineering insights include the trade-off between model complexity and inference speed on embedded platforms, the importance of robust preprocessing for handling illumination variation, and the need for careful threshold tuning to minimize false alarms while maintaining high sensitivity. The work demonstrates practical feasibility of deploying CNN-based drowsiness detection on affordable embedded hardware suitable for automotive integration.

This methodology forms the technical foundation for our implementation. We will build upon this framework by incorporating MAR analysis for yawn detection, implementing the optimized ROI extraction from Florez et al., and applying advanced optimization techniques including TensorRT acceleration, FP16 quantization, and model pruning to maximize performance on Jetson Nano.

### *D. Other Relevant Works*

Below we discuss four additional verified references that inform different aspects of our project implementation and provide context for design decisions.

**Prasath et al. (2022)** [4] implemented machine learning algorithms focusing on eye closure and yawning ratios for drowsiness detection. The work emphasizes the complementary nature of multiple drowsiness indicators. Eye closure patterns alone may not distinguish drowsiness from normal blinking in all cases. Incorporating yawn detection through mouth region analysis improves system sensitivity and reduces false negatives. The study employs traditional machine learning classifiers demonstrating that effective detection does not always require deep learning when appropriate features are engineered. This work informs our decision to implement multi-modal detection combining eye state classification with MAR-based yawn detection.

**Rathod et al. (2023)** [5] developed RealD3 system utilizing MediaPipe Face Mesh for 468 facial landmarks and YOLO for object detection achieving 94% overall accuracy. The system computes EAR and MAR metrics with PERCLOS classification distinguishing drowsiness from normal blinking based on temporal patterns. Multi-modal feature fusion combining eye state, mouth opening, hand position (hands covering face), and accessory detection (sunglasses) enhances robustness against diverse driving scenarios.

Alert mechanisms activate when EAR falls below 0.25 threshold for sustained duration or MAR exceeds yawning threshold. The work emphasizes importance of temporal filtering to avoid false alarms from brief eye closures during normal blinking. PERCLOS (Percentage of Eye Closure) metric tracking eye closure duration over sliding time windows provides more reliable drowsiness indication than instantaneous measurements. This temporal aggregation approach will be incorporated into our implementation to improve detection reliability.

The integration of YOLO object detection enables recognition of contextual factors affecting facial analysis such as

sunglasses occlusion or hands covering face. These scenarios require system adaptation or alert suppression to prevent false positives. RealD3 demonstrates the value of comprehensive scene understanding beyond isolated facial feature analysis. We will explore similar contextual awareness mechanisms in our implementation.

**Fouad (2023)** [6] demonstrated EEG-based drowsiness detection using machine learning algorithms including Linear Discriminant Analysis (LDA), Support Vector Machines (SVM-RBF and SVM-Linear), K-Nearest Neighbors (KNN), and Random Forest achieving 100% accuracy across twelve subjects. The study utilized EEG signals from only three electrodes (Fp1, Fp2, Fpz) demonstrating that effective drowsiness detection does not require full-scalp electrode arrays.

Feature extraction focused on spectral power in delta (0.5-4 Hz), theta (4-8 Hz), alpha (8-13 Hz), and beta (13-30 Hz) frequency bands. Drowsiness correlates with increased theta and alpha power combined with decreased beta power. The study employed comprehensive preprocessing including bandpass filtering, artifact removal, and epoch segmentation with one-second windows.

While physiological signal-based methods achieve exceptional accuracy, the intrusive nature of electrode placement limits automotive applicability as discussed in the survey [1]. Drivers are unlikely to tolerate wearing EEG caps during routine driving. However this work establishes performance benchmarks that vision-based systems should approach. The 100% accuracy achieved with EEG provides an upper bound for drowsiness detection performance and motivates continued optimization of vision-based approaches.

The study also highlights the importance of subject-independent validation. Cross-subject generalization remains challenging as individual differences in baseline EEG patterns, facial morphology, and drowsiness manifestation require robust model training. Our implementation will employ cross-validation strategies and diverse training data to ensure generalization across different drivers.

**Fu et al. (2024) - Survey** [1] provides additional context beyond the detailed discussion in Section II-A. The survey identifies hardware optimization as a critical enabler for embedded deployment. Model compression techniques including pruning remove redundant parameters reducing memory footprint and computational requirements. Quantization converts floating-point weights to lower precision (FP16 or INT8) accelerating inference with minimal accuracy degradation. Knowledge distillation transfers knowledge from complex teacher models to compact student models suitable for edge deployment.

Neural architecture search (NAS) automates discovery of efficient architectures optimized for specific hardware constraints. Platforms such as NVIDIA Jetson benefit from GPU-accelerated inference libraries including TensorRT providing automatic kernel optimization, layer fusion, and precision calibration. The survey emphasizes that deployment success requires co-design of algorithms and hardware optimization strategies rather than treating them as independent concerns.

The survey also discusses emerging trends including federated learning for privacy-preserving model training across distributed vehicles, self-supervised learning reducing reliance on labeled data, and multimodal fusion combining vision with audio analysis of speech patterns or vehicle sensor data. These directions represent promising avenues for future enhancement beyond our current implementation scope.

## REFERENCES


[1] B. Fu, F. Boutros, C.-T. Lin, and N. Damer, "A survey on drowsiness detection – modern applications and methods," 2024. [Online]. Available: https://arxiv.org/abs/2408.12990

[2] R. Florez, F. Palomino-Quispe, A. B. Alvarez, R. J. Coaquira-Castillo, and J. C. Herrera-Levano, "A real-time embedded system for driver drowsiness detection based on visual analysis of the eyes and mouth using convolutional neural network and mouth aspect ratio," *Sensors*, vol. 24, no. 19, 2024. [Online]. Available: https://www.mdpi.com/1424-8220/24/19/6261

[3] N. Florian, D. Popescu, and A. Hossu, "Real-time tiredness detection system using nvidia jetson nano and opencv," *Procedia Computer Science*, vol. 242, pp. 536–543, 2024, 11th International Conference on Information Technology and Quantitative Management (ITQM 2024). [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050924018209

[4] P. Nithiyanandam, J. Sreemathy, and V. Pandi, "Driver drowsiness detection using machine learning algorithm," 03 2022, pp. 01–05.

[5] S. Rathod, T. Mali, Y. Jogani, N. Faldu, V. Odedra, and P. K. Barik, "Reald3: A real-time driver drowsiness detection scheme using machine learning," in *2023 IEEE Wireless Antenna and Microwave Symposium (WAMS)*, 2023, pp. 1–5.

[6] I. A. Fouad, "A robust and efficient eeg-based drowsiness detection system using different machine learning algorithms," *Ain Shams Engineering Journal*, vol. 14, no. 3, p. 101895, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2090447922002064

# Initial Feasibility Report: Drowsiness Detection Pipeline on NVIDIA Jetson Nano (4GB)

## 1. Introduction

This report evaluates the feasibility of deploying the real-time drowsiness detection system—consisting of MediaPipe FaceMesh, NanoEyeCNN, EAR/MAR geometric features, and temporal logic—on the NVIDIA Jetson Nano 4GB Developer Kit.

## 2. System Resource Requirements

The system consists of four major components. The table below summarizes each component's description and load.

| Component | Description |
|---|---|
| MediaPipe FaceMesh | 468-landmark face mesh analysis. Most computationally expensive part. |
| NanoEyeCNN | Lightweight CNN (~137k params). Very fast inference. |
| EAR/MAR Geometry | Distance-based calculations. Negligible overhead. |
| Temporal Logic & Overlays | Threshold logic and UI overlay. Minimal load. |

## 3. Performance Feasibility

### 3.1 RAM Usage

Estimated RAM consumption:

| Component | Estimated RAM Usage |
|---|---|
| Linux OS + Desktop | ~800 MB |
| Python + OpenCV | ~200 MB |
| MediaPipe FaceMesh | ~300–600 MB |
| PyTorch Runtime | ~200–400 MB |
| NanoEyeCNN Model | < 1 MB |
| Total | ~1.5–2.0 GB |

Conclusion: Fits well within the 4GB RAM limit.

### 3.2 Expected FPS on Jetson Nano

Approximate FPS with different settings:

| Resolution | Refine Landmarks | Expected FPS |
|---|---|---|
| 640×480 | True | 6–10 FPS |
| 640×480 | False | 10–14 FPS |
| 320×240 | False | 15–20 FPS |

## 4. Feasibility Summary

| Aspect | Feasibility |
|---|---|
| Memory Usage | PASS – Well within 4GB RAM |
| Compute Load | PASS – Achieves 8–12 FPS at 640x480 |
| Power Requirements | PASS – Runs fine in 10W mode |
| Model Size | PASS – Lightweight (<1MB) |
| Latency Requirements | PASS – Meets real-time needs |

## 5. Optimization Recommendations

To improve real-time performance:

• Disable refine_landmarks=True for faster FaceMesh.

• Reduce camera resolution to 320×240.

• Run FaceMesh every 2 frames, reuse landmarks in-between.

• Enable Jetson performance mode (nvpmodel + jetson_clocks).

• Reduce OpenCV drawing load during deployment.

## 6. Conclusion

The drowsiness detection pipeline is fully feasible on the NVIDIA Jetson Nano 4GB platform. All memory, compute, power, and latency requirements fall within the board's capabilities. With minor optimizations, the system can operate smoothly in real-time conditions.

# DRIVER DROWSINESS DETECTION SYSTEM

Beginner-Friendly Overview

How the system detects sleepiness using AI + Camera

Mamona Sadaf – Research And Devolpment

Minahel Ahsan – Simulation and Algorithms

Sarah Omer – Embedded and Hardware

# SYSTEM GOAL

Monitors driver face in real time

Detects sleepiness indicators

Warns driver before danger

Works day & night

# HOW SYSTEM SEES THE DRIVER

Infrared (IR) camera captures face

Facial points detected using AI

Works even in darkness

# DETECTING EYE STATE

Eye region cropped & stabilized

AI model classifies OPEN vs CLOSED

Long closure → Sleepiness detected

# DETECTING YAWNING

Measures mouth opening width

Large sustained opening = Yawn

Frequent yawns = fatigue

# TEMPORAL FILTERING

Blinks are fast, yawns are slow

Checks how long eyes/mouth stay closed/open

Reduces false alarms

# ALARM ACTIVATION

Triggers sound alert

Screen shows: Normal, Yawning, Drowsy

Helps prevent accidents

# ALGORITHMS

- MediaPipe Face Mesh – landmark detection
- ROI Correction – stable eye region
- CNN – eye open/closed classification
- MAR – mouth opening measurement
- Temporal Logic – blink vs microsleep

The pipeline works by **detecting precise facial landmarks (MediaPipe) → stabilizing the eye region (ROI Correction) → classifying eye state (CNN) → measuring yawns (MAR) → using timing rules (Temporal Logic)** to decide real drowsiness vs normal behaviors.

# MEDIAPIPE FACE MESH

468 facial landmarks

Robust in low light

Handles head movement

# CNN EYE CLASSIFICATION

Learns eye patterns

More robust than geometric methods

Handles IR images and glasses

# MOUTH ASPECT RATIO (MAR)

Geometric yawn detection

Lightweight & fast

No extra CNN required

# PROTOTYPE

# JETSON NANO INTEGRATION ISSUES

Limited 4GB RAM

Heavy GPU load from AI models

Thermal throttling

Lower FPS under load

IR noise affects landmarks

# FINAL SUMMARY

Multi-stage AI pipeline

Accurate day/night drowsiness detection

Embedded optimized (Jetson Nano)

Reliable & real-time

## November 21, 2025 – December 21, 2025

Period: 1 month

**Overview**

**0** Active pull requests

**19** Active issues

| 0 Merged pull requests | 0 Open pull requests |
|---|---|

17
Closed issues

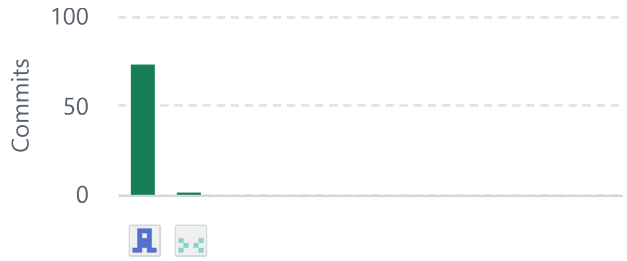
2
New issues

## Summary

Excluding merges, **2 authors** have pushed **74 commits** to main and **74 commits** to all branches.

On main, **29 files** have changed and there have been **3,136 additions** and **4 deletions**

## Top Committers




## 17 issues closed by 1 person

- **Deployment On Jetson Nano**
  #18 closed 4 days ago
- **Deployment and testing**
  #10 closed 4 days ago
- **Final Hardware Enclosure**
  #22 closed 4 days ago
- **Configure Headless Auto-Start**
  #19 closed 4 days ago
- **Low-Light & Night Testing**
  #20 closed 4 days ago
- **Documentation**
  #11 closed 4 days ago
- **Jetson Camera Pipeline Integration**
  #14 closed 4 days ago
- **Role Division**
  #3 closed 4 days ago
- **Simulation and Model Training**
  #17 closed 4 days ago
- **Simulation Pipeline**
  #13 closed 4 days ago
- **Environment Setup on Jetson Nano**
  #15 closed 4 days ago
- **Model Development & Training**

#7 closed 4 days ago

### Hardware Integration
#9 closed 4 days ago

### Edge Compute Resource Verification & Profiling
#23 closed last week

### Lightweight Neural Architecture Deployment
#24 closed last week

### Model Architecture Design
#8 closed last week

### Environment Configuration
#12 closed 3 weeks ago

## 2 issues opened by 1 person

### Real-World Performance Optimization
#21 opened last week

### Literature Review, Proposal and Report writing
#16 opened last week

⚠ Referring sites and popular content are temporarily unavailable or may not display accurately. We're actively working to resolve the issue. **Service restoration is expected to begin in early December and will roll out progressively over time.**

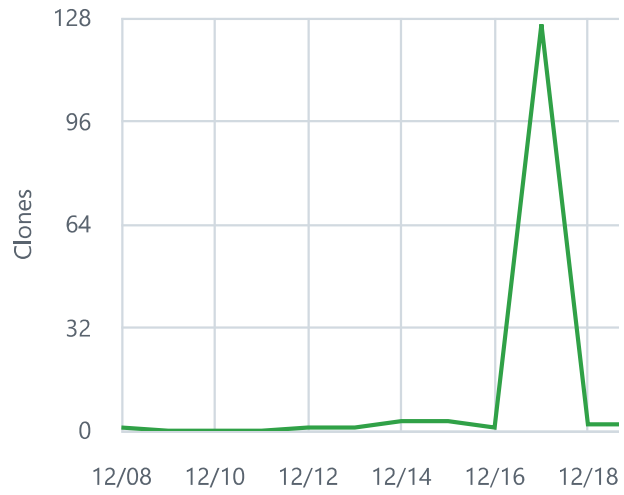
## Git clones

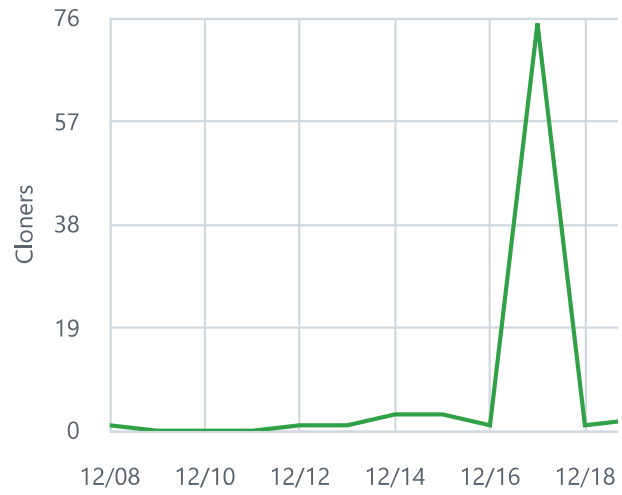
### Clones in last 14 days

156 Clones

### Unique cloners in last 14 days

95 Unique cloners

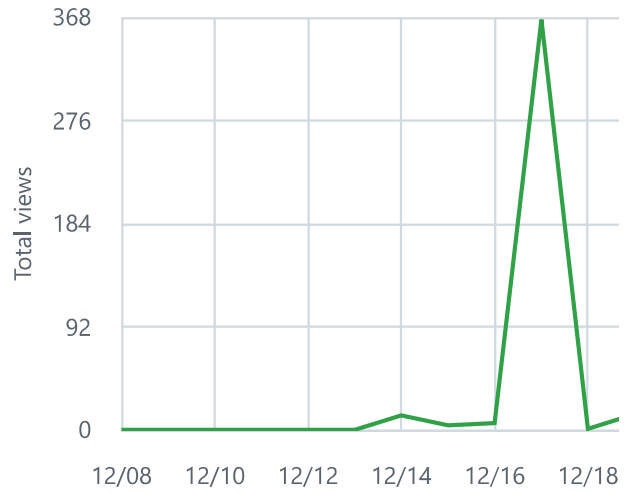



## Visitors

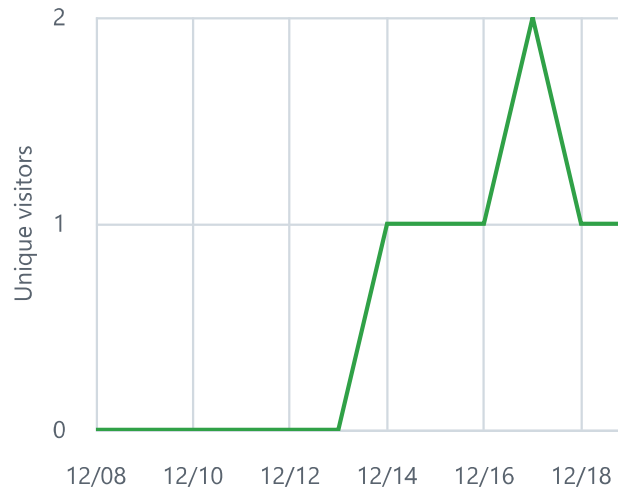
### Total views in last 14 days

429 Views




### Unique visitors in last 14 days

2 Unique visitors




## Referring sites

**We don't have enough data to show anything useful.**

It usually takes about a week to populate this table.

## Popular content

**We don't have enough data to show anything useful.**

It usually takes about a week to populate this table.