International Journal of Electrical and Computer Engineering (IJECE)
Vol. 16, No. 3, June 2026, pp. 1493∼1507
ISSN: 2088-8708, DOI: 10.11591/ijece.v16i3.pp1493-1507

Hardware-aware comparative study of lightweight convolutional neural networks for Raspberry Pi-based autonomous driving

Hyung In Kim[1], Youngmin Park[2]
[1]School of Electrical and Electronic Engineering, Yonsei University, Seoul, South Korea
[2]Division of Convergence, Sookmyung Women's University, Seoul, South Korea

Article Info

Article history:

Received Jan 12, 2026
Revised Feb 2, 2026
Accepted Mar 16, 2026

Keywords:

Autonomous driving
Edge AI
Embedded deep learning
Lightweight convolutional neural network
Raspberry Pi

ABSTRACT

Deploying deep learning models for autonomous driving on resource-constrained edge devices, such as the Raspberry Pi, presents significant challenges due to strict limitations on inference latency and memory capacity. To address these constraints, this study conducts a comprehensive comparative evaluation of lightweight convolutional neural networks (CNNs) optimized for dual-output regression of steering angle and driving speed. We benchmark a task-specific end-to-end baseline (NVIDIA CNN) against representative classification-oriented architectures—including MobileNet, ShuffleNet, EfficientNet, GhostNet, and SqueezeNet—all reformulated for this regression task. Experiments were conducted on a physical Raspberry Pi-based autonomous RC car platform to assess prediction accuracy, inference speed, and real-world closed-loop driving stability using quantitative metrics such as the normalized jerk ratio. Experimental results demonstrate a clear trade-off: while GhostNetV1 0.5× achieved the highest regression accuracy with a Total R^2 score of 95.8% and MobileNetV1 recorded a competitive MAE of 1.95, they failed to provide stable control due to severe high-frequency steering jitter. Conversely, the NVIDIA CNN proved to be the most practical solution for general edge deployment, achieving the lowest inference latency of 61.1 ms (16.4 FPS) and a minimal memory footprint of 2.78 MB, ensuring stable autonomous navigation (1.50× jerk ratio). Furthermore, ShuffleNetV2 0.5× emerged as the superior architecture for trajectory precision, recording the lowest weighted MAE of 1.60. These findings underscore that theoretical accuracy does not guarantee real-world drivability on embedded systems, providing practical guidelines for hardware-aware model selection in edge-based autonomous driving.

Corresponding Author:

Young-Min Park
Division of Convergence, Sookmyung Women's University
100 Cheongpa-ro 47-gil, Yongsan-gu, Seoul-04310, South Korea
Email: ympillow@sookmyung.ac.kr

1. INTRODUCTION

Autonomous driving has evolved significantly with the advent of deep learning, particularly through end-to-end learning paradigms that map raw sensor inputs directly to control commands [1], [2]. While high-performance computing platforms have successfully demonstrated autonomous capabilities, deploying these systems on resource-constrained edge devices remains a critical challenge [3]. Embedded platforms, such as

the Raspberry Pi, are widely used for small-scale mobility and delivery robots due to their cost-effectiveness and accessibility. However, their limited computational throughput and memory capacity impose stringent constraints on executing convolutional neural networks (CNNs) in real-time. The latency induced by computationally intensive models directly degrades control stability in closed-loop driving scenarios, making hardware-aware model selection paramount [4].

To mitigate these computational constraints, various lightweight CNN architectures—such as MobileNet [5], ShuffleNet [6], and GhostNet [7]—have been proposed. Although these models demonstrate superior efficiency in image classification tasks, their applicability to regression-based autonomous driving on edge hardware has not been systematically validated. Prior studies on embedded autonomous driving have primarily relied on simulation-based validation [8] or offline accuracy metrics [9]. These approaches often fail to fully capture the critical trade-offs between inference latency, memory usage, and real-world driving stability. Consequently, there is a lack of comprehensive benchmarks that quantitatively evaluate whether classification-oriented lightweight architectures can effectively replace task-specific models (*e.g.*, NVIDIA CNN) on severely resource-constrained platforms.

In this study, we bridge this research gap by conducting a hardware-aware comparative evaluation of lightweight CNN architectures for real-time end-to-end autonomous driving. The primary contribution of this work is the establishment of a verified, accessible Edge-AI testbed using a Raspberry Pi-based platform. As highlighted in recent surveys, small-scale vehicle platforms serve as a critical bridge for validating autonomous systems prior to full-scale deployment [10]. This offers a cost-effective alternative to full-scale autonomous vehicles, enabling researchers to validate deep learning-based steering and speed prediction algorithms in a real-world physical environment. Furthermore, we present a unified regression adaptation strategy to repurpose diverse classification-oriented lightweight CNNs into dual-output regression models, facilitating a fair and consistent comparison of architectural efficiency in edge computing scenarios [3]. Finally, by systematically analyzing the trade-offs between accuracy, latency, and memory usage, we provide a comprehensive benchmark and research environment. This aligns with the emerging need for energy-efficient Edge AI frameworks in autonomous driving [11], establishing a clear reference point for future researchers to implement and quantitatively benchmark their own algorithmic ideas against state-of-the-art baselines on resource-constrained hardware. The remainder of this paper is organized as follows. Section 2 reviews related work on end-to-end autonomous driving and lightweight CNN architectures. Section 3 describes the system overview and experimental platform. Section 4 presents the unified model adaptation and implementation strategy. Section 5 details the experimental setup, while section 6 discusses the comparative evaluation results. Finally, section 7 concludes the paper and outlines directions for future work.

2. RELATED WORKS

End-to-end autonomous driving using CNNs has been widely studied since NVIDIA demonstrated that steering angles can be directly regressed from monocular camera images without explicit modular pipelines [1]. This paradigm shift significantly simplified autonomous driving systems and enabled their deployment on small-scale platforms.

With the increasing demand for real-time autonomy on embedded and edge devices, several studies have explored autonomous driving on resource-constrained hardware such as the Raspberry Pi. Projects such as DeepPicar [12] and its variants demonstrated that low-cost platforms can support CNN-based driving, albeit with strict limitations on model complexity and inference latency. More recent studies [13] further investigated Raspberry Pi-based autonomous RC cars for educational and experimental purposes, confirming the feasibility of real-time control under constrained computational budgets.

To address these constraints, lightweight CNN architectures originally developed for mobile and embedded vision have been proposed. MobileNetV1 [5] introduced depth-wise separable convolution to significantly reduce computational cost and parameter count, while MobileNetV3 [14] further optimized inference latency through neural architecture search and hardware-aware activation functions. ShuffleNetV2 [6] focused on minimizing practical memory access cost rather than theoretical FLOPs, achieving high inference speed on ARM-based CPUs. EfficientNet [15] and EfficientNetV2 [16] employed compound scaling and fused convolution strategies to improve parameter efficiency and training speed, respectively. Additional architectures such as ResNet [17], GhostNet [7], and SqueezeNet [18] explored alternative strategies including residual learning, feature redundancy reduction, and aggressive parameter compression. Although these lightweight models

have demonstrated strong performance in image classification tasks, their applicability to regression-based autonomous driving—particularly under real hardware constraints—remains insufficiently explored. Most prior studies either evaluated classification accuracy alone or relied on simulation-based benchmarks [8], [10], [19], without comprehensive real-world driving validation on edge devices.

In contrast to existing work, this study conducts a hardware-aware comparative analysis of multiple lightweight CNN architectures adapted for dual-output regression of steering angle and driving speed. All models are evaluated under identical experimental conditions using a Raspberry Pi-based autonomous RC car, enabling a fair comparison across prediction accuracy, inference latency, memory usage, and real-world driving stability. By benchmarking classification-oriented lightweight architectures against a task-specific baseline (NVIDIA CNN), this work aims to identify the most suitable deep learning architecture for real-time autonomous driving on severely resource-constrained edge platforms.

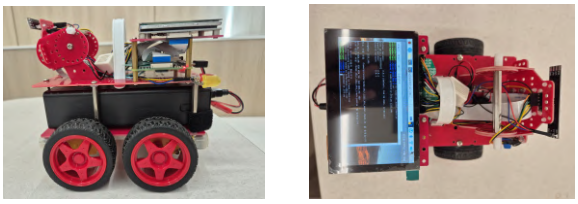
3. SYSTEM OVERVIEW

This section describes the autonomous RC car platform used for real-world evaluation of lightweight CNN architectures. The system was designed to provide a reproducible and hardware-constrained environment suitable for benchmarking real-time end-to-end autonomous driving models.

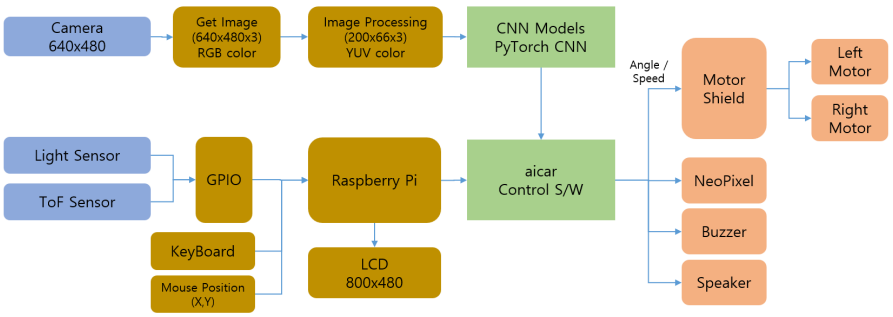
3.1. Hardware platform

The experimental platform is a small-scale autonomous RC car equipped with a Raspberry Pi 4 Model B (Broadcom BCM2711, 4 GB RAM) as the primary processing unit. Operating at 1.5 GHz, this board serves as a representative resource-constrained edge device [20]. A monocular front-facing camera is mounted on the vehicle to capture road images during operation. Lateral control (steering) is actuated via a servo motor, while longitudinal control (propulsion) is provided by a DC motor. All actuation and sensor interfacing are handled through the Raspberry Pi's general-purpose input/output (GPIO) interface. To ensure stable real-time operation, the system is powered by an external battery supply capable of sustaining continuous autonomous driving without voltage fluctuation.

Figures 1(a) and 1(b) present the side and top views of the assembled vehicle, illustrating the physical integration of the primary processing unit. The overall hardware connectivity, detailed in Figure 1(c), delineates the data flow from sensors to the Raspberry Pi and subsequently to the motor drivers. This hardware configuration was intentionally selected to reflect the computational and memory limitations commonly encountered in embedded autonomous systems.



(a) (b)



(c)

Figure 1. Overview of the Raspberry Pi-based autonomous RC car platform used for real-world evaluation, (a) side view, (b) top view, and (c) hardware configuration

3.2. Software architecture

The software system was implemented in Python using open-source libraries. Image acquisition and preprocessing were performed using OpenCV [21], while all deep learning models were implemented and executed using the PyTorch framework [22]. The operational workflow consists of three sequential stages: image data collection, model training and parameter optimization, and real-time autonomous driving execution. During data collection, camera images were captured and labeled with corresponding steering angles and driving speeds obtained through manual operation. For model training, the collected dataset was processed offline using GPU acceleration, and the optimized model parameters were subsequently deployed to the Raspberry Pi. During autonomous driving, the system operates in a closed-loop manner. At each control cycle, a camera frame is captured, processed by the deployed CNN model to predict steering and speed commands, and immediately applied to the vehicle actuators. This pipeline enables real-time evaluation of inference latency, control stability, and driving feasibility under practical hardware constraints. Figure 2(a) illustrates the overall diagram of the software workflow. As shown in Figure 2(b), images are captured using a wide-angle camera while the vehicle is manually controlled with a mouse, and the collected images undergo a preprocessing stage. Afterward, as illustrated in Figure 2(c), the images are classified, and deep learning is performed either directly on the Raspberry Pi or by compressing the entire dataset and uploading the compressed file to a Google Colab environment to generate the deep learning model. Figure 2(d) demonstrates the autonomous driving process using the trained deep learning model.

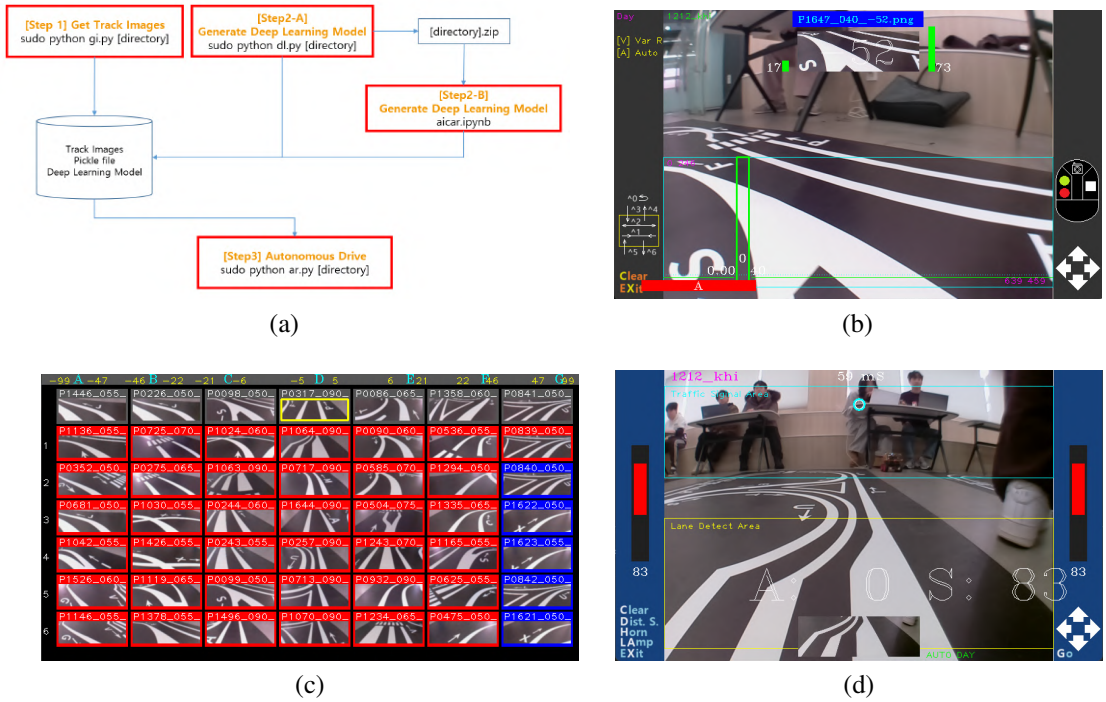


Figure 2. Software configuration of proposed autonomous RC car, (a) a block dagram of software, (b) collecting images for preprocessing, (c) image classification and generating model, and (d) autonomous driving with deep learning model

4. MODEL ARCHITECTURE AND IMPLEMENTATION

4.1. Unified regression adaptation strategy

All evaluated architectures were originally designed for image classification tasks. To repurpose them for end-to-end autonomous driving, we implemented a unified regression head adaptation scheme, as illustrated in Figure 3. Each model was systematically adapted to perform dual-output regression, simultaneously predicting steering angle and driving speed from a single visual input.

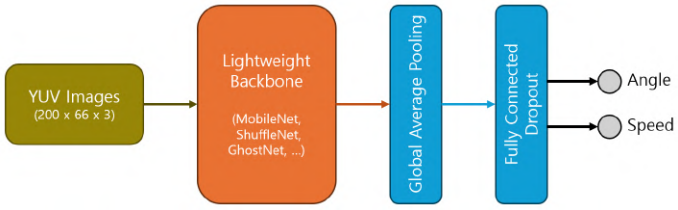


Figure 3. Unified regression head adaptation scheme applied to lightweight CNN backbones

First, the original classification heads were removed and replaced with fully connected layers, producing two continuous scalar outputs corresponding to steering and velocity commands. Second, all models were configured to process camera images in a unified input format of 66×200 pixels in the YUV color space, which is consistent with the NVIDIA CNN baseline [1] and optimized for embedded inference. This resolution provides sufficient visual information for road feature extraction while minimizing computational overhead. Third, activation functions, normalization layers [23], and regularization strategies were preserved as defined in each original architecture, except where minor modifications were required to stabilize regression performance. Dropout [24] was selectively applied to fully connected layers to mitigate overfitting across all models. This unified adaptation strategy ensures that performance differences arise from architectural characteristics rather than task-specific implementation bias.

4.2. Baseline architecture: NVIDIA CNN

The NVIDIA CNN serves as the task-specific baseline for comparison. Unlike classification-oriented architectures, this model was originally designed for end-to-end autonomous driving and directly regresses steering angles from monocular camera input [1].

The architecture, shown in Figure 4, consists of five convolutional layers followed by fully connected layers, employing strided convolutions for progressive spatial downsampling. Exponential linear unit (ELU) activation functions, defined as shown in (1), are utilized throughout the network to enhance training stability in regression tasks [25]. Due to its shallow depth and absence of complex modules, the NVIDIA CNN exhibits minimal memory usage and low inference latency, making it well-suited for deployment on embedded platforms.

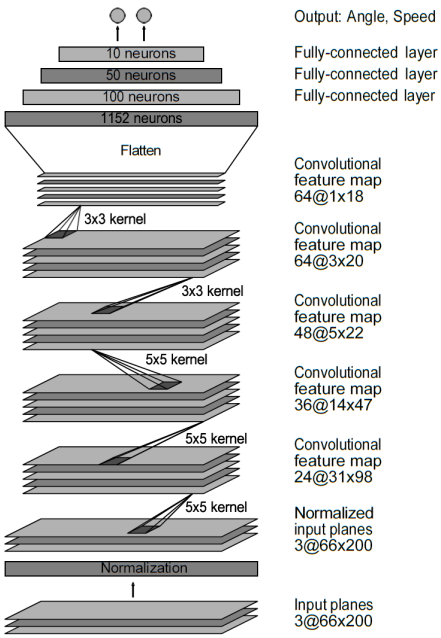


Figure 4. NVIDIA CNN architecture

$$ELU(x) = \begin{cases} e^x - 1 & , x \leq 0 \\ x & , x > 0 \end{cases} \quad (1)$$

4.3. Lightweight CNN architectures for edge deployment

To evaluate the effectiveness of classification-oriented lightweight models in autonomous driving regression tasks, several representative architectures were selected based on their distinct efficiency strategies. This selection aligns with recent findings that optimized CNNs remain the most viable option for real-time edge inference [26]. MobileNet architectures [5] employ depth-wise separable convolutions to significantly reduce computational cost. MobileNetV3 [14] further incorporates hardware-aware neural architecture search and lightweight activation functions to optimize CPU inference latency. ShuffleNetV2 [6] emphasizes practical execution speed on embedded hardware by minimizing memory access cost through channel shuffling mechanisms. EfficientNet [15] and EfficientNetV2 [16] focus on parameter efficiency through compound scaling and fused convolutional blocks, respectively, while ResNet [17] introduces residual learning to stabilize gradient propagation in deeper networks. GhostNet [7] and SqueezeNet [18] adopt alternative lightweight strategies, including feature redundancy reduction and aggressive parameter compression.

Instead of using full-scale architectures intended for server-grade GPUs, we selected their compact variants (e.g., MobileNetV3-Small [14], ShuffleNetV2 0.5× [6]) to ensure compatibility with the limited resources of the Raspberry Pi. As detailed in section 4.1, the classification heads of all selected backbones were replaced with the unified regression head shown in Figure 3. This ensures that performance variations are attributable solely to the backbone architecture.

5. EXPERIMENTAL SETUP

5.1. Dataset collection

To evaluate the performance of lightweight CNN architectures under real-world conditions, an image dataset was collected using the Raspberry Pi-based autonomous RC car on the test track shown in Figure 5. The custom test track was constructed with overall dimensions of 3.1×1.3 m. This testbed was iteratively designed over two years to encompass a comprehensive set of driving primitives required for robust autonomous navigation. In contrast to simplistic loop tracks frequently utilized in prior studies [10], our final track configuration integrates high-complexity scenarios, including multiple straight and curved segments with diverse curvature profiles, complex intersections (X-crossing) and T-junctions, lane merging scenarios (two lanes converging into one), right-turn maneuvers at forked paths, and simulated pedestrian crosswalks. This 'all-in-one' design rigorously tests the model's ability to handle diverse control tasks within a single lap, thereby providing a sufficient level of environmental complexity to validate the architectural feasibility of the proposed models. Data acquisition was performed by manually driving the vehicle along the track for five complete laps, sequentially traversing designated points from A to Z. By completing five laps on this track, sufficient training data for deep learning model construction was collected, consisting of track images along with corresponding steering angles and driving speeds.

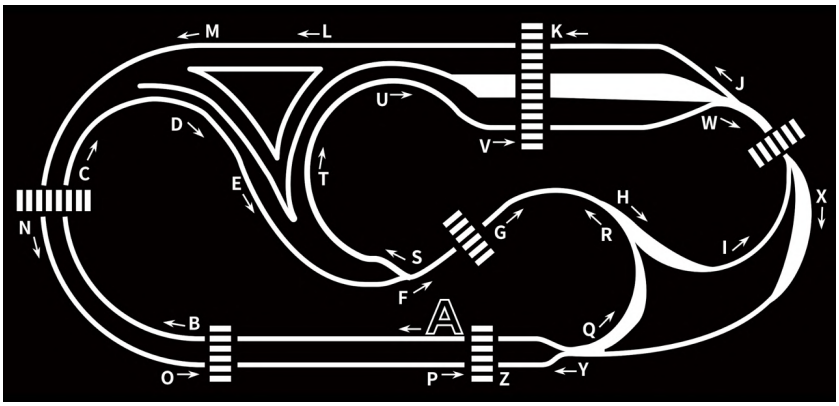


Figure 5. Autonomous RC car test driving track

A total of 1,647 images were collected from five complete laps, each paired with corresponding steering angle and driving speed values. Steering angles were recorded as integer values in the range of -99 to +99 with a resolution of 1, while driving speeds were recorded in increments of 5 within the range of 40 to 90. All captured images were resized and converted to a 66×200 YUV format, which was consistently used across all evaluated models. The dataset was divided into 1,314 training samples and 326 validation samples, representing an approximate 80:20 split. Additionally, a distinct subset of 7 samples was reserved specifically for qualitative visual verification of model predictions. This identical data split was applied to all models to ensure fair and consistent evaluation. The detailed specifications of the collected dataset are summarized in Table 1.

5.2. Training configuration

To bypass the computational constraints of the edge device, all model training was performed off-board using an NVIDIA T4 GPU in the Google Colab environment. Optimization was conducted using the Adam optimizer [27] with an initial learning rate of 1×10^{-3}, training for a maximum of 300 epochs. Batch sizes were configured at 16 for training and 64 for validation. To mitigate overfitting and conserve computational resources, an early stopping mechanism was applied, terminating training when validation loss failed to improve over a patience of 30 epochs. The model parameters corresponding to the minimum validation loss were retained for subsequent deployment. Given that lateral control (steering) stability is paramount for autonomous navigation, a weighted mean squared error loss was adopted. The loss contribution was explicitly weighted at 0.9 for steering angle and 0.1 for driving speed. This configuration reflects practical driving requirements, where minor steering errors may result in lane deviation, whereas moderate speed errors are generally tolerable. The detailed training hyperparameters are summarized in Table 2.

Table 1. Specifications of the collected dataset

Parameter	Specification
Total samples	1,647 images
Input resolution	66×200 pixels (YUV)
Steering label	[-99, +99], Integer, Resolution: 1
Speed label	[40, 90], Integer, Step: 5
Training set	1,314 samples (79.8%)
Validation set	326 samples (19.8%)
Visualization set	7 samples (0.4%)

Table 2. Training hyperparameters

Hyperparameter	Specification
Hardware	NVIDIA T4 GPU (Google Colab)
Optimizer	Adam (β_1 = 0.9, β_2 = 0.999)
Learning rate	1×10^{-3}
Batch size	16 (Train) / 64 (Validation)
Max Epochs	300
Early stopping	Patience: 30 Epochs
Loss weights	Steering: 0.9, Speed: 0.1

5.3. Deployment and evaluation metrics

The deployment workflow followed a cloud-to-edge paradigm. Data collection and preprocessing were performed on the edge device, followed by model training on a high-performance cloud GPU environment (Google Colab) to ensure convergence stability. However, for the inference phase on the autonomous vehicle, we intentionally restricted the execution to the Raspberry Pi CPU without utilizing hardware accelerators (*e.g.*, Edge TPU or GPU). This constraint was imposed to: i) Quantify the intrinsic architectural efficiency of the lightweight models without hardware-specific optimizations masking their computational overhead; and ii) Verify the feasibility of autonomous driving on ubiquitous, low-cost commodity hardware, thereby establishing a baseline for entry-level edge deployment.

To assess real-time feasibility, two temporal performance metrics were measured during closed-loop driving: i) inference time, defined as the time required to process a single image and generate steering and speed predictions; and ii) frame time, defined as the total duration of a control cycle, including image capture, inference, and preparation for the next frame.

To mitigate experimental variance, each model was evaluated over three consecutive laps on the test track, and average values were reported. In addition to these real-time latency measurements, memory usage, model size, parameter count, and prediction accuracy on the validation set were recorded to provide a comprehensive assessment of deployment feasibility on resource-constrained hardware. Furthermore, to provide a single holistic metric for model performance, we introduced the weighted total R^2 score. This score represents a weighted regression coefficient combining steering angle and speed predictions, utilizing the same 0.9/0.1 weighting scheme as the training loss function described in section 5.2. This weighted aggregation reflects the prioritization of steering stability over speed control in the autonomous driving task.

To evaluate the quality of autonomous navigation, we utilized intervention rate and smoothness (jerk) as practical robotics metrics. To maintain experimental consistency and objectivity, a strict zero-human-

intervention protocol was enforced during all test runs. Consequently, the intervention rate was recorded as 0 for all evaluated models, as any track deviation resulted in the immediate termination of the run (failure) rather than manual correction. Regarding stability analysis, we formulated a composite metric termed the weighted normalized jerk ratio. First, raw Jerk values—defined as the rate of change in control commands—were measured independently for both steering angle and driving speed across all models. Crucially, the corresponding jerk values generated by the human driver were also recorded to serve as the baseline for smoothness. To derive a relative smoothness score, the raw jerk values of each model were normalized against the human's baseline, effectively calculating a ratio of model-to-human behavior. Subsequently, these normalized steering and speed ratios were aggregated using the specific 0.9/0.1 weighting scheme established in our training configuration (section 5.2). This metric quantifies how closely each model replicates the stable driving characteristics of a human operator. Similarly, to assess the aggregate predictive error with a focus on lateral control, we utilized the weighted mean absolute error (weighted MAE), using the same weighting scheme (section 5.2).

6. RESULT AND DISCUSSION

6.1. Training results of lightweight models

All evaluated models were trained using an identical dataset and optimization protocol to ensure fair comparison. As illustrated in Figure 6, most architectures exhibited rapid loss reduction during the initial training phase, followed by gradual convergence. The minimal gap observed between training and validation loss curves indicates that severe over fitting did not occur across the evaluated models.

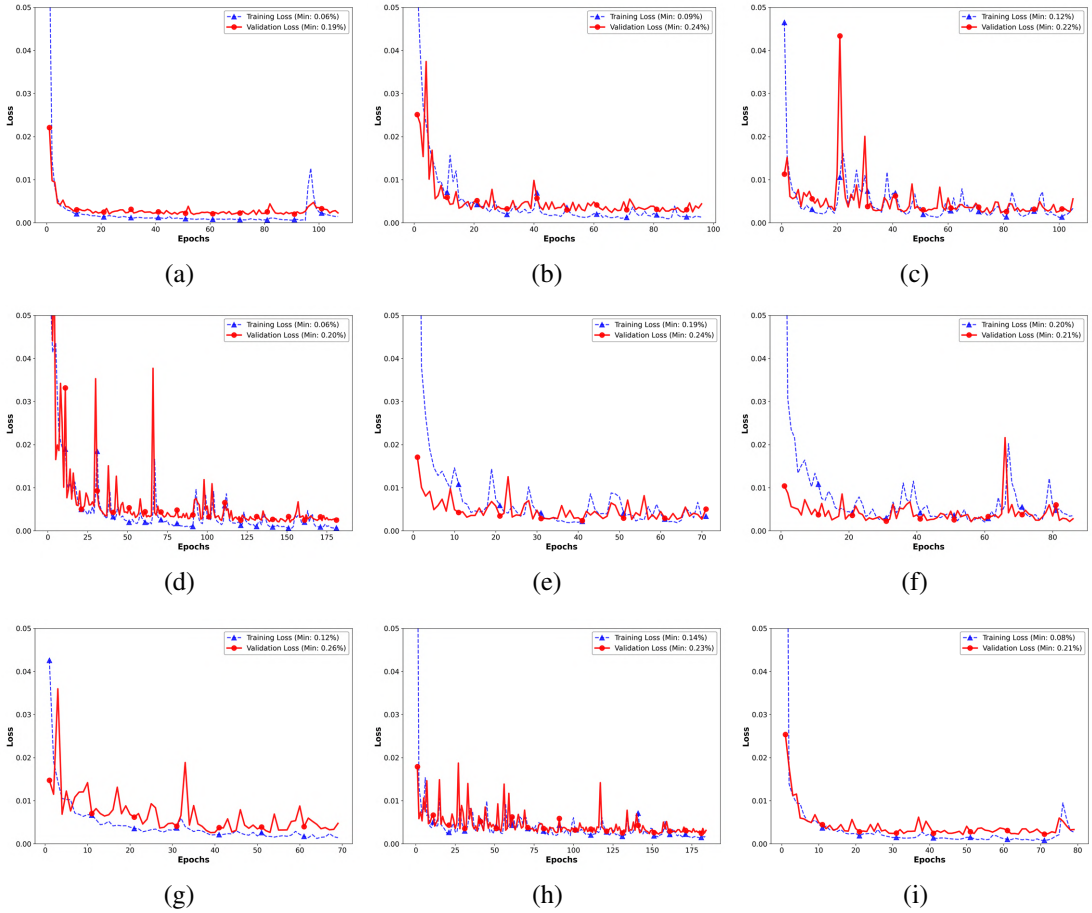


Figure 6. Training and validation loss curves of evaluated models, (a) NVIDIA CNN, (b) MobileNetV1, (c) MobileNetV3-Small, (d) ShuffleNetV2 0.5×, (e) EfficientNet-B0, (f) EfficientNetV2-Small, (g) ResNet-20 (Option B), (h) GhostNetV1 0.5×, and (i) SqueezeNet

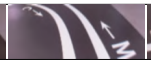
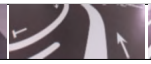
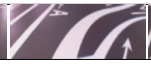
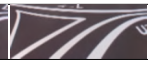
Among the candidates, the NVIDIA CNN in Figure 6(a) demonstrated the most stable convergence behavior, achieving the lowest minimum validation loss despite its relatively simple architecture. This result suggests that its structural design is well aligned with regression-based autonomous driving tasks. Similarly, MobileNetV1 in Figure 6(b) showed a rapid initial decline in loss with steady convergence, maintaining a narrow gap between training and validation sets. In contrast, MobileNetV3-Small in Figure 6(c) and ShuffleNetV2 0.5× in Figure 6(d) show noticeable oscillations in their validation loss curves during later training stages. These fluctuations suggest sensitivity to specific data samples or suboptimal optimization dynamics, which may introduce minor steering instability during real-world driving. EfficientNet-B0 in Figure 6(e) and EfficientNetV2-Small in Figure 6(f) both converged rapidly and required fewer epochs; however, their final validation loss remained slightly higher than that of the baseline model. Similarly, ResNet-20 (Option B) in Figure 6(g) and GhostNetV1 0.5× in Figure 6(h) exhibited early convergence but were prone to minor spikes in validation loss, indicating some volatility in learning dynamics. Finally, Squeezenet in Figure 6(i) exhibited smooth convergence with minimal fluctuation, indicating consistent parameter updates throughout training. Overall, the training results indicate that architectural simplicity and task-specific design play a critical role in stabilizing regression learning under limited data conditions.

6.2. Prediction performance on test dataset

Table 3 compares the predicted steering angles and driving speeds produced by each trained model against the ground truth values from the estimation dataset. All models demonstrated comparable predictive accuracy, successfully reproducing human-like driving behaviors such as decelerating during curves and accelerating on straight segments. For samples involving large steering angles, predicted values closely matched the ground truth across all architectures. In contrast, minor discrepancies were observed in near-straight driving segments, where several models slightly underestimated speed and exhibited a small leftward steering bias. This behavior suggests conservative control strategies learned from the dataset.

A notable observation was that all models predicted larger steering angles and lower speeds for a specific sample corresponding to a gentle curve. This consistent behavior indicates that the models interpreted subtle visual features as indicative of increased curvature, effectively embedding a safety-oriented driving policy. These results confirm that lightweight CNN architectures can capture essential driving dynamics when adapted to regression tasks, even under limited training data.

Table 3. Comparison of ground truth and predicted angles and speeds on selected test samples

Images / Models														
Ground Truth	Angle	Speed	Angle	Speed	Angle	Speed	Angle	Speed	Angle	Speed	Angle	Speed	Angle	Speed
	-48	**50**	**-33**	**55**	**-8**	**85**	**-1**	**90**	**+8**	**75**	**+38**	**50**	**+48**	**50**
NVIDIA CNN	-50	52	-29	56	-11	86	-8	84	+20	65	+40	49	+42	48
MobileNetV1	-51	54	-27	52	-10	81	-7	78	+18	68	+43	51	+44	50
MobileNetV3-Small	-50	49	-33	53	-11	85	-13	77	+19	67	+43	52	+42	47
ShuffleNetV2 0.5×	-54	53	-26	52	-2	82	-11	81	+15	67	+42	51	+43	49
EfficientNet-B0	-48	53	-25	54	-4	83	-5	77	+12	73	+38	49	+41	47
EfficientNetV2-Small	-50	52	-28	50	-9	78	-5	82	+19	70	+41	53	+39	50
ResNet-20 (Option B)	-51	53	-31	52	-10	78	-5	88	+19	65	+46	53	+36	50
GhostNetV1 0.5×	-51	57	-27	60	-8	86	-10	85	+11	75	+42	51	+42	51
SqueezeNet	-49	54	-25	56	-11	80	-2	79	+21	69	+40	51	+40	48

6.3. Real-time performance on Raspberry Pi

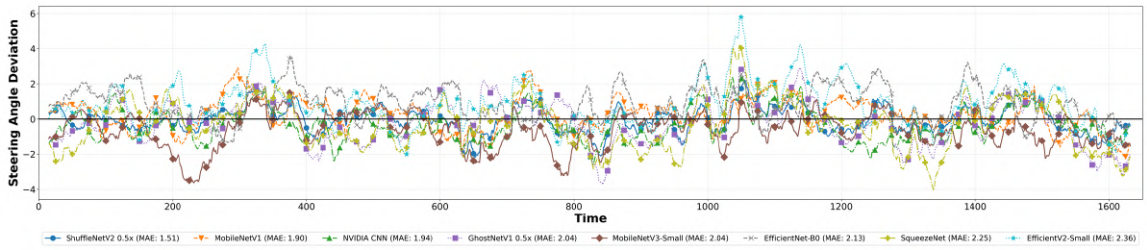
The quantitative evaluation results, encompassing both on-board performance metrics and offline predictive accuracy, are summarized in Table 4. Notably, with the exception of ResNet-20, all models achieved a 100% autonomous completion rate, successfully navigating the test track without any human intervention. Although ResNet-20 was selected as a representative shallow residual network to reflect minimal residual architectures suitable for embedded deployment, it recorded a 0% completion rate, failing to finish a single lap. This failure is primarily attributed to its critical latency and memory inefficiency. As shown in Table 2, it recorded the highest frame time of 269.5 ms (3.7 FPS), causing severe phase lag that prevented real-time steering corrections. Furthermore, despite its low parameter count, the heavy memory usage (49.33 MB) induced by residual skip connections created a significant bandwidth bottleneck on the CPU. This computational burden, combined with the lowest predictive accuracy, resulted in the vehicle's inability to maintain the track trajectory.

Table 4. Quantitative evaluation of inference latency, memory footprint, accuracy, and driving stability on Raspberry Pi

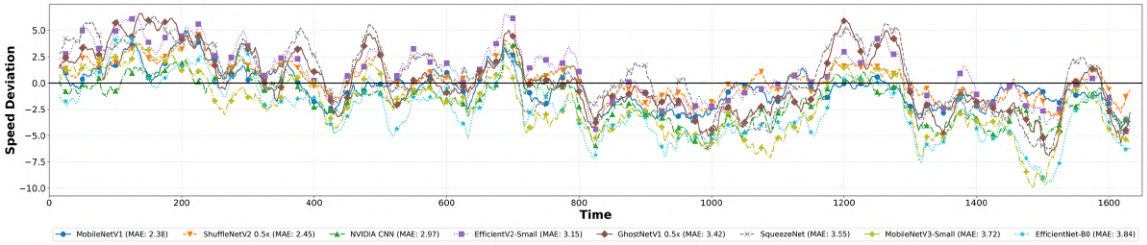
Models	Frame Time (ms) ↓	Inference Time (ms) ↓	Latency (FPS) ↑	Params (M)	Total R^2 ↑	Memory Usage (MB)	Validation Loss ↓	Weighted MAE ↓	Weighted Normalized Jerk Ratio ↓
NVIDIA CNN	**61.1**	**35.9**	**16.4**	0.25	94.3%	**2.78**	**0.19%**	2.04	1.50×
MobileNetV1	122.2	95.7	8.2	0.81	92.3%	35.34	0.24%	1.95	2.06×
MobileNetV3-Small	92.9	67.1	10.8	1.51	91.7%	16.77	0.22%	2.21	1.58×
ShuffleNetV2 0.5×	87.0	60.8	11.5	0.34	93.7%	9.12	0.20%	**1.60**	1.68×
EfficientNet-B0	195.5	170.2	5.1	4.01	94.7%	66.79	0.24%	2.30	1.75×
EfficientNetV2-Small	195.0	187.5	5.1	6.05	93.7%	75.54	0.21%	2.44	**1.37**×
ResNet-20 (Option B)	269.5	243.3	3.7	0.27	92.9%	49.33	0.26%	N/A	N/A
GhostNetV1 0.5×	123.8	98.2	8.1	1.31	**95.8%**	20.37	0.23%	2.18	1.65×
SqueezeNet	101.1	74.9	9.9	0.74	93.0%	24.36	0.21%	2.38	1.60×

Inference latency is a critical factor for real-time autonomous control. The NVIDIA CNN achieved the lowest average frame time of 61.1 ms, corresponding to approximately 16.4 FPS, outperforming all other models. ShuffleNetV2 0.5× ranked second, achieving around 11.5 FPS. In contrast, EfficientNet variants and ResNet-20 exhibited frame times approaching or exceeding 200 ms, resulting in insufficient throughput for stable control. Memory efficiency further distinguished the baseline model. The NVIDIA CNN required less than 3 MB of estimated total memory during runtime, significantly lower than all lightweight classification-oriented architectures. Even ShuffleNetV2 0.5×, the most memory-efficient among the lightweight models, required over three times more memory than the baseline. Although GhostNetV1 0.5× achieved the highest overall predictive performance (Total R^2 score), its inference latency and memory consumption were substantially higher than those of the NVIDIA CNN. These results highlight the critical trade-off between predictive accuracy and real-time feasibility on embedded hardware.

For the autonomous driving successful models, we first conducted a visual inspection of driving quality using the temporal deviation profile in Figure 7. This graph plots the real-time deviation of the models' predictions relative to the human's driving record for steering angle and speed, as shown in Figures 7(a) and 7(b). Based on these trajectory data, we derived the predictive accuracy metric, MAE, as illustrated in Figure 8. Compares the performance of all evaluated architectures in terms of steering angle MAE and speed MAE presented in Figures 8(a) and 8(b). Quantitatively, MobileNetV1 and ShuffleNetV2 0.5× demonstrated superior performance, achieving the lowest Speed MAE (2.38) and Steering MAE (1.51), respectively.

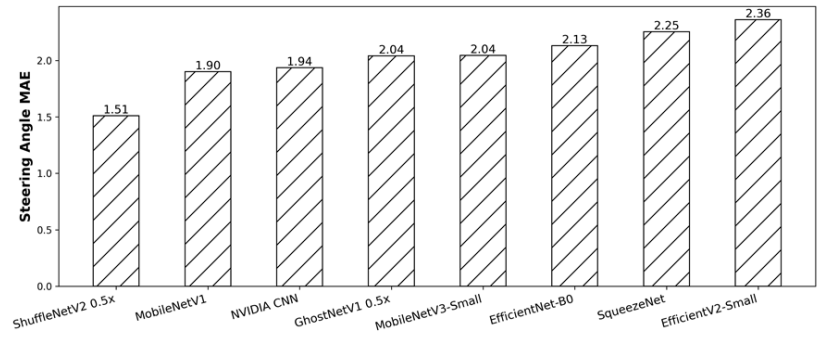


(a)

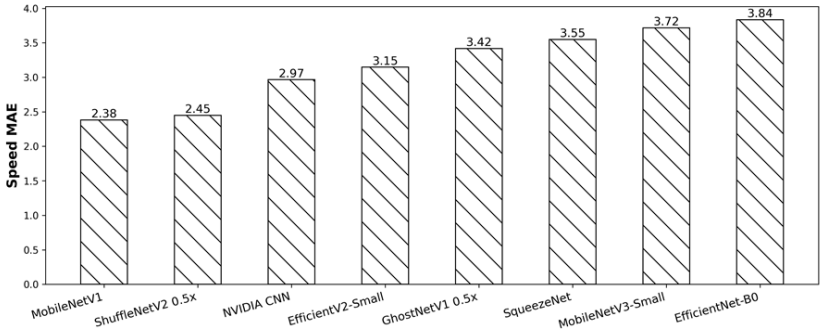


(b)

Figure 7. Real-time deviation profiles of (a) steering angle and (b) speed



(a)



(b)

Figure 8. Comparison of (a) MAE for steering and (b) speed

However, a comparative analysis between the visual traces in Figure 7 and average metrics presented in Figure 8 reveals a critical insight regarding 'Jitter'. While MobileNetV1 shows excellent MAE scores, its deviation profile in Figure 7 exhibits significant high-frequency oscillations. This indicates that although the model's predictions are accurate on average, they fluctuate rapidly around the target value, resulting in unstable driving behavior. In contrast, EfficientNetV2-Small and NVIDIA CNN display much flatter and smoother curves in Figure 7, despite having slightly higher MAE values.

To strictly quantify this instability, we evaluated the normalized jerk ratio in Figure 9. This metric confirms the visual observation: MobileNetV1 recorded the worst jerk ratio of 2.06×, proving that its low MAE masked a physically jerky control policy. Conversely, EfficientNetV2-Small achieved the best Jerk Ratio of 1.37×, followed by NVIDIA CNN (1.50×). This conclusively demonstrates that for comfortable and stable autonomous driving, minimizing the jerk ratio (smoothness) is a more critical objective than simply minimizing MAE (accuracy). Ultimately, EfficientNetV2-Small and NVIDIA CNN proved to be the most robust architectures, offering the optimal balance between trajectory tracking and control stability.

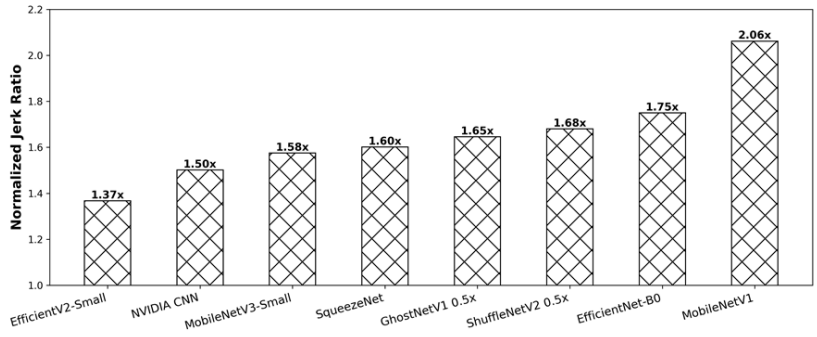


Figure 9. Comparative assessment of driving stability (weighted normalized jerk ratio)

6.4. Discussion

The experimental results demonstrate that while classification-oriented lightweight CNNs can be adapted for regression-based driving, theoretical architectural efficiency does not necessarily translate into real-time closed-loop performance. Specifically, our findings indicate that even models with high offline accuracy can suffer from physical instability due to inference latency and architectural characteristics on constrained edge devices.

To rigorously synthesize these multi-dimensional trade-offs, we present a Pareto Frontier analysis in Figure 10. This visualization plots inference latency (X-axis) against predictive precision (weighted MAE, Y-axis), with the bubble size and color representing the normalized jerk ratio (stability). Ideally, an optimal model should reside in the bottom-left corner (fast & precise) with a small, blue bubble (stable).

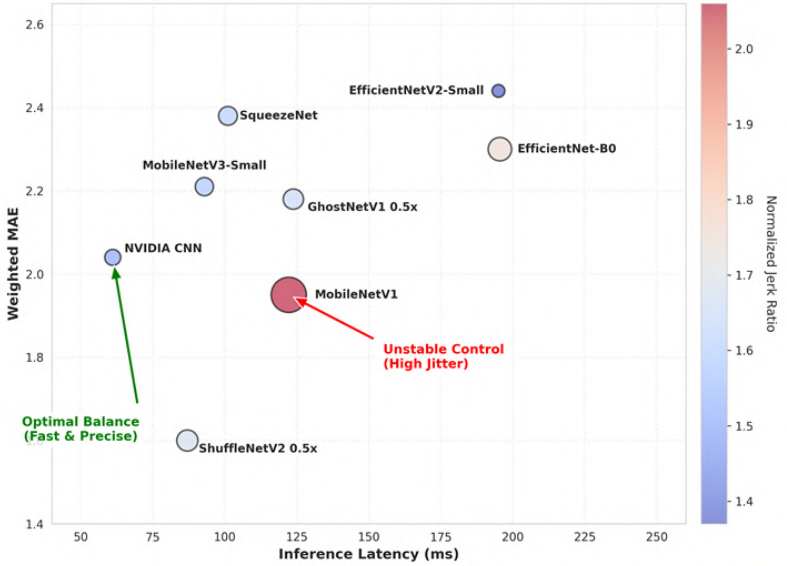


Figure 10. Pareto Frontier analysis of inference latency, predictive accuracy, and driving stability

As illustrated in Figure 10, the NVIDIA CNN is positioned closest to this optimal zone, offering the fastest inference with competitive precision and a compact bubble, confirming its robustness. ShuffleNetV2 0.5× occupies the lowest position on the Y-axis, validating its role as the precision specialist. Crucially, the chart exposes the deceptive nature of MobileNetV1; while its coordinate position suggests reasonable performance, its disproportionately large red bubble reveals severe control jitter. This visual evidence reinforces our core argument: optimizing for latency and accuracy alone is insufficient for autonomous driving; stability must be treated as a primary design constraint.

From a practical perspective, these findings suggest that simpler, task-specific architectures may outperform more sophisticated general-purpose models when deployed on severely resource-constrained platforms. This is particularly true for closed-loop control tasks where minimizing the jerk ratio and ensuring temporal consistency are more critical than maximizing absolute regression accuracy.

7. CONCLUSION

This study presented a comprehensive evaluation of lightweight deep learning models for end-to-end autonomous driving on resource-constrained edge devices. By deploying various architectures on a Raspberry Pi-based autonomous RC car platform, we analyzed the critical relationship between inference latency, predictive accuracy, and physical driving stability.

A task-specific NVIDIA CNN was employed as the baseline and compared against several representative lightweight architectures originally designed for image classification. All models were trained using an identical dataset and evaluated under the same hardware and experimental conditions to ensure fairness. With the exception of ResNet-20, all evaluated models successfully achieved autonomous navigation without lane

deviation, demonstrating that lightweight CNNs can be effectively repurposed for regression-based driving tasks on embedded platforms.

Our comprehensive experiments revealed a critical discrepancy between conventional offline metrics, such as R^2 accuracy, and real-world driving quality. A prime example is GhostNetV1 0.5×, which achieved the highest Total R^2 of 95.8%, demonstrating superior offline learning capability. However, this theoretical advantage did not fully translate into physical driving performance, as it recorded a higher MAE (2.18) compared to other optimized models. Similarly, MobileNetV1 achieved highest accuracy but failed to provide stable control, recording the worst Normalized Jerk Ratio of 2.06×. This indicates that despite low regression error, the model suffered from severe high-frequency steering jitter, proving that accuracy metrics alone cannot guarantee temporal consistency in closed-loop environments.

Conversely, NVIDIA CNN and ShuffleNetV2 0.5× demonstrated that optimized lightweight architectures can achieve human-level driving performance on the Raspberry Pi 4 CPU without hardware acceleration. Specifically, NVIDIA CNN offered the best real-time responsiveness (16.4 FPS), which allowed for rapid control updates and resulted in a highly stable drive (1.50× jerk ratio). Meanwhile, ShuffleNetV2 excelled in trajectory precision, achieving the lowest weighted MAE of 1.60, making it ideal for tasks requiring exact path following. These findings confirm that for edge-based autonomy, minimizing inference latency and jerk is just as critical as maximizing predictive accuracy.

In summary, this study establishes a practical benchmark for deploying deep learning-based autonomous driving on extreme edge platforms. The results highlight that while classification-oriented architectures can be effectively adapted for regression tasks, selecting the optimal model requires a holistic evaluation of latency, stability, and accuracy rather than relying solely on offline validation metrics. Based on our rigorous analysis, we recommend NVIDIA CNN as the optimal choice for general-purpose edge autonomy due to its superior balance of speed and stability, while ShuffleNetV2 0.5× is advised for applications demanding high trajectory precision. By providing these actionable guidelines, this work serves as a foundational reference for developers aiming to build accessible, low-cost autonomous robotic systems.

8. LIMITATIONS AND FUTURE WORK

Despite the effectiveness of the proposed comparative evaluation, this study has several limitations that warrant further investigation.

First, the dataset used in this work was collected from a single test track under fixed environmental conditions. All training, validation, and testing data were acquired using the same track layout, lighting conditions, and surface characteristics. Although this controlled setup enabled fair model comparison, it may limit the generalization capability of the trained models when exposed to unseen environments, such as different track geometries, outdoor settings, or varying illumination conditions. Consequently, the learned driving policies may be partially biased toward the specific visual features of the experimental environment.

Second, the scale of the dataset was relatively modest. However, this scale is consistent with prior Raspberry Pi-based autonomous driving studies such as DeepPicar, where real-time feasibility rather than large-scale generalization was the primary objective. While sufficient for comparative benchmarking, the limited number of samples may restrict the robustness of deeper or more complex architectures, which typically require larger datasets to fully exploit their representational capacity. This constraint may have contributed to the observed instability or underperformance of certain models during training and real-world deployment.

Third, this study focused exclusively on software-level architectural optimization and did not incorporate advanced model compression techniques. All models were evaluated using full-precision floating-point inference on the Raspberry Pi CPU. As a result, the potential benefits of hardware-aware optimization strategies were not explored.

Future work will address these limitations in several directions. First, data collection will be expanded to include diverse driving scenarios, such as varying track shapes, outdoor environments, dynamic lighting conditions, and potential obstacles. This extension is expected to improve model generalization and robustness in real-world applications.

Second, advanced optimization techniques such as quantization, pruning, and mixed-precision inference will be investigated to further reduce inference latency and memory consumption. These methods are particularly promising for enabling stable real-time autonomous driving on even more severely resource-constrained embedded platforms.

Finally, future studies will explore the deployment of the proposed models on a broader range of edge devices with varying computational capabilities. By extending the evaluation beyond the Raspberry Pi, the applicability of the proposed benchmarking framework to heterogeneous embedded systems can be systematically assessed.

ACKNOWLEDGMENTS

The authors would like to thank Yonsei University and Sookmyung Women's University for providing the data, facilities, and equipment necessary to conduct this research. The authors also express their sincere gratitude to the professors in the School of Electrical and Electronic Engineering at Yonsei University and the Division of Convergence at Sookmyung Women's University for their valuable guidance and support throughout this study.

FUNDING INFORMATION

Authors state no funding involved.

AUTHOR CONTRIBUTIONS STATEMENT

This journal uses the Contributor Roles Taxonomy (CRediT) to recognize individual author contributions, reduce authorship disputes, and facilitate collaboration.

Name of Author	C	M	So	Va	Fo	I	R	D	O	E	Vi	Su	P	Fu
Hyung In Kim	✓	✓	✓	✓	✓	✓		✓	✓		✓			
Young-Min Park	✓	✓							✓			✓	✓	

C : **C**onceptualization
M : **M**ethodology
So : **So**ftware
Va : **Va**lidation
Fo : **Fo**rmal Analysis
I : **I**nvestigation
R : **R**esources
D : **D**ata Curation
O : Writing - **O**riginal Draft
E : Writing - Review & **E**diting
Vi : **Vi**sualization
Su : **Su**pervision
P : **P**roject Administration
Fu : **Fu**nding Acquisition

CONFLICT OF INTEREST STATEMENT

Authors state no conflict of interest.

DATA AVAILABILITY

The data supporting the findings of this study are available from the corresponding author upon reasonable request.

REFERENCES

[1] M. Bojarski *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.
[2] L. Chen, P. Wu, K. Chitta, B. Jaeger, A. Geiger, and H. Li, "End-to-end autonomous driving: Challenges and frontiers," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 46, no. 12, pp. 10164–10183, 2024. doi: 10.1109/TPAMI.2024.3435937.
[3] J. Xie, X. Zhou, and L. Cheng, "Edge computing for real-time decision making in autonomous driving: Review of challenges, solutions, and future trends," *International Journal of Advanced Computer Science and Applications*, vol. 15, no. 7, pp. 1198–1210, 2024. doi: 10.14569/ijacsa.2024.0150759.
[4] D. Kim, J. Lee, and K. Park, "Latency analysis of in-vehicle network for advanced driver assistance system," *2024 IEEE 100th Vehicular Technology Conference* (VTC2024-Fall), 2024, pp. 1–5. doi: 10.1109/VTC2024-Fall63153.2024.10757452.
[5] A. G. Howard *et al.*, "MobileNets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.
[6] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "ShuffleNet V2: Practical guidelines for efficient CNN architecture design," *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 116–131, 2018.
[7] K. Han, Y. Wang, Q. Tian, J. Guo, C. Xu, and C. Xu, "GhostNet: More features from cheap operations," *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 1580–1589.
[8] A. Aich, A. Kulkarni, and E. Ohn-Bar, "Scalable offline metrics for autonomous driving," *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2025, pp. 13682-13689. doi: 10.1109/IROS60139.2025.11247722.

[9] D. Butani, R. Kaddis, and C.-J. Chung, "Evolutionary hyperparameter optimization to find lightweight CNN models for autonomous steering," *2025 IEEE International Conference on Electro Information Technology (eIT)*, 2025, pp. 344–349. doi: 10.1109/eIT64391.2025.11103679.
[10] Z. Li, X. Hu, and Y. Wang, "Autonomous driving small-scale cars: A survey of recent development," *arXiv preprint arXiv:2404.06229*, 2024.
[11] D. Katare, D. Perino, J. Nurmi, M. Warnier, M. Janssen and A. Y. Ding, "A survey on approximate edge AI for energy efficient autonomous driving services," *IEEE Communications Surveys & Tutorials*, vol. 25, no. 4, pp. 2714-2754, 2023. doi: 10.1109/COMST.2023.3302474.
[12] M. G. Bechtel, E. McEllhiney, M. Kim, and H. Yun, "DeepPicar: A low-cost deep neural network-based autonomous car," *IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pp. 11–21, 2018. doi: 10.1109/RTCSA.2018.00011.
[13] Y. Park, "Development of Raspberry Pi autonomous car using OpenCV and NVIDIA CNN model," *The journal of the convergence on culture technology*, vol. 11, no. 2, pp. 367–378, 2025.
[14] A. Howard *et al.*, "Searching for MobileNetV3," *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 1314–1324.
[15] M. Tan and Q. V. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," *Proceedings of the 36th International Conference on Machine Learning (ICML)*, 2019, pp. 6105–6114.
[16] M. Tan and Q. V. Le, "EfficientNetV2: Smaller models and faster training," *Proceedings of the 38th International Conference on Machine Learning (ICML)*, 2021, pp. 10096–10106.
[17] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.
[18] F. N. Iandola *et al.*, "SqueezeNet: AlexNet-level accuracy with 50× fewer parameters and <0.5MB model size," *arXiv preprint arXiv:1602.07360*, 2016.
[19] Y. A. Ozaibi, M. Dulva Hina and A. Ramdane-Cherif, "End-to-end autonomous driving in CARLA: A survey," *IEEE Access*, vol. 12, 2024, pp. 146866-146900. doi: 10.1109/ACCESS.2024.3473611.
[20] Raspberry Pi Foundation, "Raspberry Pi 4 model B dtasheet," *Raspberry Pi Foundation*, Cambridge, UK, Tech. Rep., 2019.
[21] G. Bradski, "The OpenCV library," *Dr. Dobb's Journal: Software Tools for the Professional Programmer*, vol. 25, no. 11, pp. 120–123, 2000.
[22] A. Paszke *et al.*, "PyTorch: An imperative style, high-performance deep learning library," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 32, pp. 8026–8037, 2019.
[23] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, 2015, pp. 448–456.
[24] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research (JMLR)*, vol. 15, no. 1, pp. 1929–1958, 2014.
[25] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (ELUs)," *International Conference on Learning Representations (ICLR)*, 2016.
[26] M. A. A. Talib, *et al.*, "Systematic review of a lightweight convolutional neural network architectures on edge devices," *International Journal of Reconfigurable and Embedded Systems (IJRES)*, vol. 14, no. 2, pp. 339-352, 2025.
[27] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations (ICLR)*, 2015.

BIOGRAPHIES OF AUTHORS

Hyung In Kim received B.S. degree in electronic and electrical engineering from Sungkyunkwan University, South Korea, in 2025. He is currently enrolled in the master's program in the School of Electrical and Electronic Engineering at Yonsei University, South Korea. He can be contacted at email: khi103@yonsei.ac.kr.

Youngmin Park received the B.Eng., M.Eng., and Ph.D. degrees from Yeungnam University, Republic of Korea, in 1993, 1995, and 2005, respectively. He is currently an assistant professor with the Division of Convergence, Sookmyung Women's University, Republic of Korea. His research interests include artificial intelligence, autonomous driving, and computer vision. He can be contacted at ympillow@sm.ac.kr.