

# Driver activity recognition using deep learning based on multi-step batch size up

Darmawan Utomo<sup>1</sup>, Natanael Indria Prambodo<sup>1</sup>, Budihardja Murtianta<sup>2</sup>

<sup>1</sup>Department of Computer Engineering, Faculty of Electronics and Computer Engineering, Satya Wacana Christian University, Salatiga, Indonesia

<sup>2</sup>Department of Electronics Engineering, Faculty of Electronics and Computer Engineering, Satya Wacana Christian University, Salatiga, Indonesia

## Article Info

### Article history:

Received Jul 28, 2024

Revised Jul 9, 2025

Accepted Sep 1, 2025

### Keywords:

Activity recognition

Batch size

Electric vehicle

Event data recorder

Long short-term memory

## ABSTRACT

The increasing popularity of electric motorbikes in Indonesia, while promoting sustainable mobility, also raises concerns regarding traffic safety. Given the high incidence of motorcycle-related accidents, there is a critical need for systems capable of monitoring and recognizing driver behavior. This study proposes a driver activity recognition system for electric motorbikes, utilizing an event data recorder (EDR) to capture seven key sensor signals: three-axis acceleration, voltage, current, power, and speed. A custom dataset was constructed using data collected from 10 subjects, each performing five driving activities including forward drive, brake, stop, left turn, and right turn for over three-minute intervals per activity. The classification model is based on a long short-term memory (LSTM) neural network. To optimize training efficiency, a multi-step batch size up (MSBU) strategy was introduced, which accelerates training time by 1.84× compared to a fixed batch size of 32. The best performance was achieved using a segment length of 75 time-steps, yielding an accuracy and macro F1-score of 0.9873. These results demonstrate the effectiveness of the proposed system for real-time driver behavior monitoring and activity recognition in electric motorbike applications.

*This is an open access article under the [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.*



## Corresponding Author:

Darmawan Utomo

Department of Computer Engineering, Faculty of Electronics and Computer Engineering

Satya Wacana Christian University

Jalan Diponegoro 52-60, Salatiga, Indonesia

Email: darmawan.utomo@uksw.edu

## 1. INTRODUCTION

The increasing adoption of battery electric vehicles (BEVs), especially motorbikes and cars, in Indonesia highlights their growing role as essential modes of transportation. In fact, motorbikes (combustion and electric engines) are the dominant choice, used by 81.78% of the population. However, this growing dependency also comes with significant risks including motorcycle-related traffic accidents remain alarmingly high, contributing to over 150,000 casualties in 2021, with a year-on-year increase of 3.62% [1]. Human factors such as reckless driving, fatigue, and lack of skill are the leading causes [2]. In Southeast Asia, motorcycles are involved in up to 70% of traffic fatalities, often due to erratic driving behaviors like sudden braking or improper lane changes [3]. While electric motorbikes contribute positively to environmental sustainability, their quiet operation and the inexperience of new users pose additional safety challenges. Therefore, continuous monitoring of driver behavior and real-time recognition of vehicle

activities (e.g., braking, turning, and idling) becomes critical to improving road safety and enabling intelligent response systems.

Understanding the state of the vehicle, whether it's accelerating, braking, or turning, is essential for ensuring both driver and system-level decision-making accuracy. This awareness supports enhanced safety features, such as collision avoidance, motion-based sensor calibration, and improved coordination in connected environments (vehicle to vehicle or vehicle to infrastructure). It also helps prevent misinterpretations, such as detecting a false collision when stopped at a traffic light.

Integrating such state awareness with deep learning-based behavior recognition enables more proactive and adaptive vehicle systems. To support these capabilities, the integration of an event data recorder (EDR) system becomes essential. Initially introduced in Formula 1 racing to analyze crashes based on driver input, engine performance, and electronic behavior [4], EDRs are now increasingly relevant for civilian use. By recording key data during vehicle operation, EDRs provide a reliable basis for reconstructing the driver activities or traffic incidents. Figure 1 illustrates the data collection flow used in EDR systems, showing how critical vehicle and driver information is gathered to determine the exact causes of accidents. EDRs play a critical role in reconstructing accidents by recording parameters related to driver behavior and vehicle dynamics, making them a key component in future intelligent transportation systems.



Figure 1. Data collection flow [3]

The application of EDRs in electric vehicles (EVs) has been actively explored in recent years. Prior implementations typically employed microcontrollers such as the Atmega2560, interfaced with sensors including GPS modules and electrical sensors (e.g., current and voltage detectors). These systems recorded operational data to SD cards at temporal intervals ranging from 100 milliseconds to one minute [5]. Building on these foundational systems, more advanced frameworks have been proposed to facilitate structured extract–transform–load (ETL) pipelines for EV trajectory data, enabling improved analytics and visualization [6].

As the automotive industry moves toward greater automation, EDRs are evolving to support autonomous driving capabilities. These advanced EDRs now enable automated data storage and facilitate access to recorded data for key stakeholders, including insurance companies, automotive manufacturers, government agencies, and researchers [7]. In parallel, deep learning has emerged as a transformative approach within the field of intelligent transportation systems. Applications include driver-in-the-loop modeling for parameter tuning [8], road condition monitoring using GPS and camera systems [9], battery degradation prediction [10], and clustering-based behavioral modeling in connected and autonomous vehicle (CAV) networks based on sensor-derived metrics such as brake pressure, throttle position, steering angle, and velocity [11].

Despite the temporal nature of vehicle and driver behavior data, many of these studies have not leveraged sequence-based deep learning models such as recurrent neural networks (RNNs), particularly long short-term memory (LSTM) networks. LSTMs are especially well-suited for time-series due to their unique architecture incorporating memory cells and gating mechanisms [12]. These capabilities have made LSTMs widely successful in domains requiring sequential analysis, including visual learning behavior identification [13], intrusion detection [14], [15], fraud detection [16], phishing recognition [17], medical diagnostics [18]–[21], autonomous navigation [22], and context-aware systems [23]–[25].

In the specific domain of human activity recognition (HAR), LSTM networks and their hybrid variants (often paired with convolutional neural networks (CNNs)) have enabled real-time classification using accelerometer data sampled at 50 ms intervals via mobile devices [26]–[29]. However, most of these studies utilize static batch sizes (e.g., batch size of 64 in [28]), overlooking recent findings that show dynamic batch sizing can significantly enhance training efficiency and model performance [30]–[32]. Furthermore, these HAR models have rarely been adapted to the unique operational characteristics of electric motorbikes, which differ notably from four-wheeled vehicles in terms of maneuverability, rider posture, and electrical system behavior.

This paper addresses a pressing challenge in the field of intelligent transportation systems: the need for a reliable, low-cost, real-time driver activity recognition system that can operate efficiently on resource-constrained platforms, particularly electric motorbikes. While existing HAR solutions have made progress using LSTM and CNN models on mobile or cloud platforms [26]–[29], they often rely on static batch sizes and high-compute resources, making them ill-suited for embedded real-time deployment. Moreover, current

HAR models are not tailored to the distinct characteristics of electric motorbikes, which differ from cars in terms of maneuverability, rider posture, and electrical system dynamics.

To address these gaps, we propose the driver activity recognition system for electric motorbikes (DARSEM), an embedded, end-to-end solution designed specifically for real-world two-wheeled EV. DARSEM leverages an ESP32 microcontroller connected to multiple sensors and applies a lightweight LSTM model trained using a novel multi-step batch size up (MSBU) strategy.

DARSEM introduces the following key innovations:

- MSBU: a dynamic batch size scheduling strategy that accelerates model convergence by incrementally increasing batch size during training, achieving a 1.84× reduction in training time without loss in accuracy.
- Sensor fusion: integration of multidimensional time-series data from a 3D accelerometer (ax, ay, and az), speed sensor, voltage, current, and power metrics to comprehensively model motorbike dynamics.
- Temporal behavior modeling: application of LSTM architecture to capture temporal dependencies across five critical driver behavior classes: driving forward, braking, idling, turning left, and turning right.
- Onboard real-time feasibility: an implementation pipeline using Arduino IDE and ESP32 hardware suitable for low-cost, embedded deployment.

These contributions are substantiated through extensive experimentation, including ablation studies and comparative evaluation with recent state-of-the-art approaches, as shown in Table 1.

Table 1. Comparative analysis of recent driver activity recognition systems and the proposed DARSEM

Feature/aspect	WISDM (2020) [28]	Driver activity recognition (2024) [33]	Adaptive eco-driving identification (2024) [34]	Proposed: DARSEM
Platform	Smartphone-based sensor logging	Real vehicle environment with onboard cameras	Simulated driving environment using unreal engine 4	ESP32 microcontroller (embedded system)
Sensor inputs	Accelerometer and gyroscope (3-axis)	Sequential image inputs	Multivariate time-series data: speed, throttle, steering, acceleration	3D accelerometer (ax, ay, az), speed, voltage, current, power (7 features total)
Model architecture	LSTM-based deep learning	TimeDistributed CNN + LSTM	LSTM-based network with adaptive modules	LSTM with MSBU technique
Recognition capabilities	6 activities: walking, jogging, sitting, standing, upstairs, downstairs	9 activities: driving, drinking, texting, smoking, and talking.	Eco-driving behavior identification	5 motorbike activities: drive, brake, stop, turn left, and turn right
Dataset	Public dataset with thousands of samples from smartphones	Data from 35 participants in real driving scenarios	30 subjects, controlled simulator setting	10 subjects, each performing each activity for 3 minutes
Performance metrics	Accuracy: ~0.96; F1-score: ~0.95	Accuracy: 88.7% (daytime), 92.4% (nighttime); F1 up to 0.92	Accuracy and F1-score not explicitly stated	Accuracy and F1-score: 0.9873 (segment size=75, MSBU technique)

## 2. METHOD

This section outlines the end-to-end methodology used to develop DARSEM. The method encompasses the design and integration of embedded hardware, sensor deployment, and data acquisition processes tailored for electric motorbike operation. It also includes the strategies employed for data labeling, preprocessing, and training a deep learning model. The main goal of this method is to systematically collect high-quality time-series data, label it accurately, and use it to train an LSTM neural network to classify different driving tasks. Each component in this pipeline as shown in Figure 2 plays a critical role in ensuring the reliability and accuracy of the driving activity recognition system.

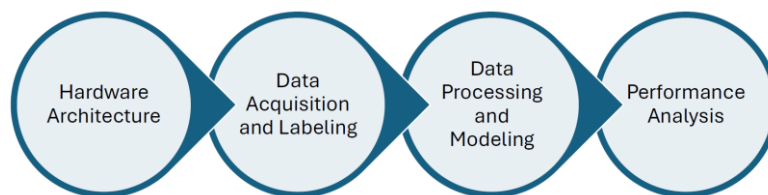


Figure 2. Overall workflow of the DARSEM methodology, comprising four main components: hardware architecture, data acquisition and labeling, data processing and modeling, and performance analysis

## 2.1. Hardware architecture

### 2.1.1. System block diagram

DARSEM is built upon an ESP32-based embedded system integrated with a suite of sensors, storage, and communication peripherals. The sensor suite includes an MPU6050 for capturing tri-axial acceleration data, a GY-NEO6V2 GPS module for vehicle speed tracking, and a PZEM017 module for measuring battery voltage, current, and power. These sensor inputs represent the core signals needed to model driver behavior in electric motorbike scenarios. To facilitate onboard data storage, an SD card module is interfaced with the ESP32, allowing all sensor readings to be saved in CSV format. The embedded system is fully programmed using the Arduino IDE. The overall configuration and schematic of the system are presented in Figure 3, where Figure 3(a) illustrates the system configuration and Figure 3(b) depicts the detailed schematic of the embedded EDR subsystem.

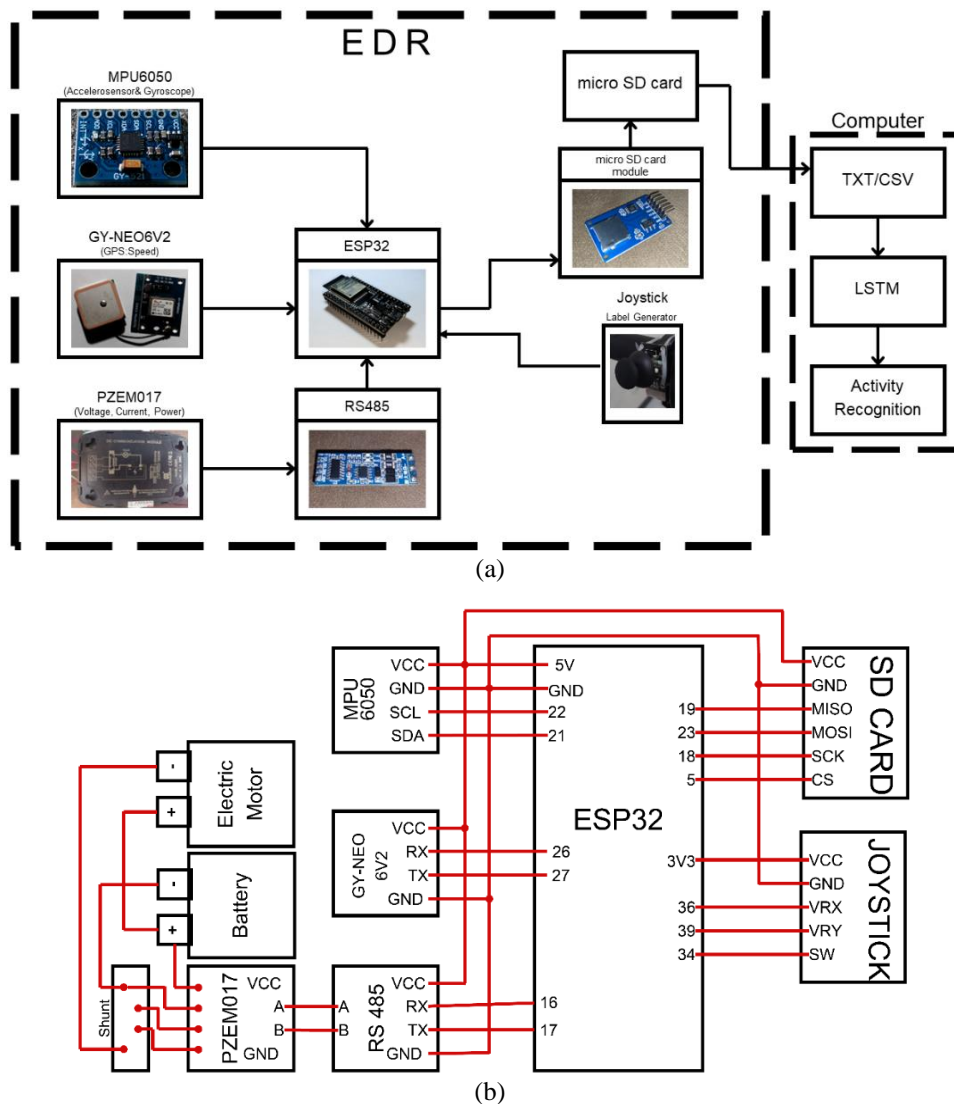


Figure 3. Block diagram of the DARSEM; (a) illustrating the embedded EDR and external computer system and (b) schematic diagram of the EDR subsystem

Sensor data acquisition follows a modular design. The MPU6050 provides accelerometric readings; the GY-NEO6V2 reports real-time velocity via GPS; and the PZEM017 transmits power metrics through an RS485 serial communication interface to the ESP32. Additionally, joystick switch inputs are employed as labels for driver activity classes, offering supervised annotations for training data. The ESP32 microcontroller orchestrates the entire data acquisition process including synchronizing sensor readings, managing the storage protocol, and logging labeled data in real time.

The computer system component of DARSEM handles the preprocessing, training, and evaluation of the deep learning models. Raw sensor data stored on the SD card is transferred and parsed in CSV format. In the preprocessing phase, the continuous sensor streams are segmented into data windows representing time-series slices of driver behavior. These data segments are partitioned into training-validation and test sets. Multiple LSTM-based neural network architectures are then designed and trained using these segmented datasets. The experiments focus particularly on evaluating the influence of two critical hyperparameters:

- Batch size, which affects the frequency of weight updates during training.
- Segment (or window) size, which determines the temporal resolution of input sequences.

Model performance is assessed using standard classification metrics: accuracy, macro-averaged precision, recall, and F1-score, as well as training time and inference latency. These metrics enable a comprehensive evaluation of both effectiveness and efficiency across architectural and parameter configurations. This modular system design, combined with robust temporal modeling using LSTM, enables DARSEM to capture and classify subtle variations in electric motorbike driver behavior and offering a foundation for intelligent mobility applications, forensic analysis, and safety-critical decision support.

### 2.1.2. Event data recorder deployment

The EDR system is physically integrated into the electric motorbike, with all components securely housed within the under-seat trunk compartment, except for the PZEM017 module, which is installed adjacent to the battery to directly monitor load voltage and current. The hardware configuration is illustrated in Figure 4. The electric motorbike operates on a 72 V battery system with a maximum current capacity of 38 A, powering a 2,000 W electric motor. The PZEM017 sensor module is selected for its capacity to handle DC voltages up to 300 V and currents up to 100 A, making it suitable for monitoring high-power EV systems.



Figure 4. EDR component placement in the electric motorbike

The EDR system is powered via a 12 V line tapped from the vehicle's ignition key switch, which is subsequently stepped down to 5 V using a voltage regulator to supply the ESP32 and peripheral sensors. To ensure continuous GPS tracking, the GY-NEO6V2 GPS module is powered independently via a portable power bank, allowing it to maintain satellite acquisition even when the motorbike is switched off. This design choice addresses the inherent delay in GPS satellite lock-on, which typically requires several minutes upon reactivation.

## 2.2. Labeling and data acquisition

### 2.2.1. Label generator

The labeling of activity data is conducted synchronously with sensor data acquisition, enabling the system to generate supervised datasets in real time. This process is facilitated through the integration of a push button and a joystick module, both of which are mounted on the motorbike's steering handle, as depicted in Figure 5. The push button (non-joystick) is used to signal subject transitions, such as when a new driver assumes control of the motorbike, allowing for clear demarcation in the recorded dataset. The joystick module itself consists of two analog potentiometers (X and Y axes) and an integrated push button, enabling multi-directional input labeling.

These inputs are used to tag specific driving activities or behaviors (e.g., acceleration, braking, and turning), thereby enriching the dataset with precise and interpretable ground truth labels. Sensor data collection is carried out simultaneously with the labeling process. This labeling process is assisted with the help of a push button component and a joystick module which is installed on the steering wheel of a motorbike as shown in Figure 5. The non-joystick push button is used for labeling subject changes just before

data collection begins. Meanwhile, the joystick module is a module consisting of a push button and two potentiometers X and Y.

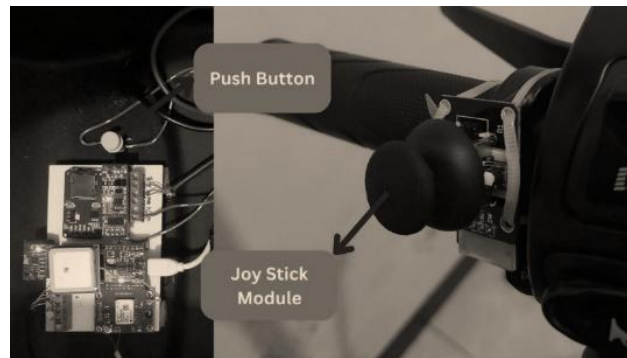


Figure 5. Installation of the joystick module and push button on the motorbike steering system

### 2.2.2. Data acquisition

For driving task data collection in intelligent vehicle systems, acquisition of high-quality sensor data is essential for training and evaluating machine learning models such as LSTM networks. In this study, data collection was carried out by ten licensed participants, each performing five predefined driving tasks: forward driving, braking, turning (left/right), and stopping. Each task was executed over a three-minute interval under various terrain conditions—including inclines, descents, and flat roundabouts with a diameter of approximately six meters—to capture diverse driving behaviors. The “forward” and “brake” labels were obtained on hilly roads, “left” and “right” through continuous circular movement at roundabouts, and the “stop” label when the vehicle remained stationary.

A custom-designed electronic circuit, integrated into the electric motor system, was employed for data acquisition. An embedded program running on a microcontroller continuously collected signals from multiple sensors, including accelerometers, GPS, voltage, current, power, and speed sensors. During each task, participants were required to press a labeling button to mark their activity in real time. All sensor readings and their corresponding activity labels were recorded and stored on an SD card for subsequent analysis. The following formal pseudocode (Algorithm 1) outlines the systematic procedure for iterating through all subjects and tasks, activating sensors, performing each driving maneuver, and logging data for downstream use in model development.

---

#### Algorithm 1. Sensor-based driving task data collection using electric vehicle

---

```

for subject = 1 to 10 do
  for each task in {Forward, Braking, Turning (Left/Right), Stopping (Idle) } do
    (a) Activate sensors:
      i. Initialize accelerometer, GPS, voltage, current, power, speed, and labeling
      button.
    (b) Perform driving task:
      i. if task = Forward:
        - If road is flat: accelerate to target speed and maintain.
        - If road is uphill: increase throttle to maintain speed.
        - If road is downhill: control descent to maintain safe speed.
        - Continue forward driving actions for 3 minutes.
      ii. if task = Braking:
        - If road is flat: accelerate then apply brakes to stop.
        - If road is uphill: drive forward then apply brakes to stop.
        - If road is downhill: engage brakes to prevent overspeed, decelerate to
        stop.
        - Maintain stationary (stopped) state for 3 minutes.
      iii. if task = Turning (Left/Right):
        - Position on flat approach at moderate speed.
        - Execute left or right turn maneuver.
        - Repeat turning actions for 3 minutes.
      iv. if task = Stopping (Idle):
        - Bring vehicle to a complete stop.
        - Idle (vehicle stationary) for 3 minutes.
      v. if task = AdditionalTask:
        - Execute the additional task-specific action.
        - Repeat action for 3 minutes.
  
```

---



```

(c) Data logging:
    i. Start recording sensor data
       (accelerometer, GPS, voltage, current, power, speed, label).
    ii. Execute the task for the full 3-minute duration while logging.
    iii. Stop recording sensor data.
(d) Save logged data to SD card.
end for each task
end for each subject

```

## 2.3. Data processing and modeling

### 2.3.1. Data preprocessing

Before training the LSTM classification model, sensor data collected from all subjects must be preprocessed. The preprocessing step involves aggregating raw data from the SD cards of all participants and normalizing the feature values to a common scale using min-max normalization. This ensures that all input features contribute proportionately during model training. The features to be normalized include accelerometer readings (Ax, Ay, Az), voltage (V), current (I), power (P), and speed (v). The mathematical formulation of this normalization process is defined in (1), which rescales the feature values to the range [0,1]:

$$X_{new} = \frac{(X - X.min)}{(X.max - X.min)} \quad (1)$$

where:  $X_{new}$  is the array of data after normalization;  $X$  is the array of raw data features;  $X.min$  is the minimum value in the data feature array; and  $X.max$  is the maximum value in the data feature array.

The complete data preprocessing pipeline, including aggregation, normalization, and output formatting, is formalized in Algorithm 2.

---

#### Algorithm 2. Data preprocessing using min-max normalization

---

Input: Raw data files from SD cards of all subjects

Output: Normalized feature matrix  $X_{new}$

---

```

1. Initialize empty list All_Data
2. for subject = 1 to 10 do
    (a) Load data from SD card into variable Subject_Data
    (b) Append Subject_Data to All_Data
3. Concatenate All_Data into a single dataset X_raw
4. Extract features: Ax, Ay, Az, V, I, P, v from X_raw
5. for each feature in {Ax, Ay, Az, V, I, P, v} do
    (a) Compute min_val = min(feature)
    (b) Compute max_val = max(feature)
    (c) Normalize feature using:
        i. feature_norm = (feature - min_val) / (max_val - min_val)
6. Combine all normalized features into X_new
7. Return X_new

```

---

### 2.3.2. Model architecture, hyperparameters, and multi-step batch size

The preprocessed time-series dataset was used to train an LSTM-based classification model implemented using the Keras–TensorFlow framework. This architecture was selected due to the sequential and temporal nature of the input data, which is well-suited for RNN, particularly LSTM networks. LSTM is capable of learning long-term dependencies in time-series data, making it an effective choice for recognizing temporal patterns in driver behavior.

The model comprises several hyperparameters, including the number of LSTM units, dropout rate, dense layer configuration, and importantly, batch size, which is explored extensively in this study. As the classification task involves five distinct classes of driver activities, the model employs categorical cross-entropy as the loss function and Adam as the optimizer. The performance metric tracked during training is accuracy.

The training pipeline incorporates a novel batch size strategy, namely MSBU, in addition to multi-step batch size down (MSBD) and fixed batch sizes, to evaluate the impact of dynamic batch size adjustments. Each training experiment is conducted for a total of 120 epochs, with 20 epochs per batch size configuration in the dynamic strategies. The data split consists of 56% for training, 14% for validation, and 30% for testing. The training process is outlined in Algorithm 3.

Batch size, a critical hyperparameter in deep learning, governs how many samples are used to compute a single gradient update. Its choice significantly influences training dynamics, convergence behavior, and computational efficiency. Common batch size paradigms include:

- Stochastic gradient descent (SGD): batch size=1
- Mini-batch: batch size between 1 and the dataset size
- Full-batch: batch size=total dataset size

**Algorithm 3. LSTM-based classification model with batch size up evaluation**


---

Input:

- X: Feature set with shape (samples, timesteps, features)
- y: One-hot encoded target labels
- batch\_sizes: List of batch sizes to evaluate = [32, 64, 128, 256, 512, 1024]

Output:

- history\_list: Training history for each batch size

1. Initialize an empty list history\_list
2. Create a Sequential model
  - a. Add LSTM layer with: (Units = 128, Input shape = (number of timesteps, number of features in X))
  - b. Add Dropout layer with rate = 0.7
  - c. Add Dense layer with: (Units = 64, Activation = ReLU)
  - d. Add Output Dense layer with: (Units = number of classes in y, Activation = Softmax)
3. Compile model with: (Loss = Categorical Crossentropy, Optimizer = Adam, Metric = Accuracy)
4. For each batch\_size in batch\_sizes, do:
  - a. Train the model using: (Epochs = 20, Validation split = 20%, Current batch\_size)
  - b. Append training history to history\_list

---

SGD offers high flexibility in early training by frequently updating weights, although its loss curve is often noisy. Conversely, larger batch sizes produce smoother loss curves but may require longer to escape local minima. Mini-batch training strikes a balance, making it a widely adopted approach. In conventional practice, static batch sizes, typically powers of two for GPU optimization, are used throughout training. However, recent findings [29] highlight that dynamic batch sizing can yield performance gains by allowing larger weight updates early in training and stabilizing convergence in later stages.

Building on this insight, our study introduces and evaluates three batch size strategies:

- MSBU: progressively increases batch size every 20 epochs across six steps: [32, 64, 128, 256, 512, 1024]
- MSBD: starts with a large batch and decreases every 20 epochs in reverse: [1024, 512, 256, 128, 64, 32]
- Static batch size (Bz): keeps batch size constant at z for the full 120 training epochs

This approach differs from prior work [29], which focused on CNNs in large-scale GPU environments. Here, we apply dynamic batch sizing within a time-series classification context using LSTM networks, and on a more constrained embedded system dataset. By maintaining a consistent epoch count (120) across all experiments, we enable direct comparisons of classification performance and training efficiency under varying batch size strategies.

## 2.4. Performance analysis

Following the training, validation, and testing phases of the LSTM-based classification model, performance evaluation is conducted to assess the model's effectiveness in recognizing multiple driving activities. The classification involves five distinct classes: forward, turn left, turn right, brake, and stop. To ensure robust evaluation across all classes, particularly in the presence of class imbalance, standard multi-class metrics are employed. These include balanced accuracy weighted, macro average precision, macro average recall, and macro F1-score, as defined in (2)–(7) [35]:

$$\text{Balanced Accuracy Weighted} = \frac{\sum_{k=1}^K \frac{TP_k \cdot w_k}{Total_{row_k}}}{K \cdot W} \quad (2)$$

$$Precision_k = \frac{TP_k}{TP_k + FP_k} \quad (3)$$

$$\text{Macro Average Precision} = \frac{\sum_{k=1}^K Precision_k}{K} \quad (4)$$

$$Recall_k = \frac{TP_k}{TP_k + FN_k} \quad (5)$$

$$\text{Macro Average Recall} = \frac{\sum_{k=1}^K Recall_k}{K} \quad (6)$$

$$\text{Macro F1 - score} = \frac{2 \cdot \text{Macro Average Precision} \cdot \text{Macro Average Recall}}{(\text{Macro Average Precision} + \text{Macro Average Recall})} \quad (7)$$



where:  $TP_k$  is true positive for class  $k$ ;  $FP_k$  is false positive for class  $k$ ;  $FN_k$  is false negative for class  $k$ ;  $w_k$  is frequency of the samples in the dataset;  $W$  is total number of samples in the dataset; and  $K$  is number of classes (forward, turn left, turn right, brake, and stop).

The complete calculation of computing performance metrics, including balanced accuracy weighted, macro precision, macro recall, and macro F1-score, is formalized in Algorithm 4.

---

**Algorithm 4. Computing performance metrics**


---

Input: Confusion matrix CM of shape  $(K, K)$ , where  $K = 5$  classes

Output: Balanced Accuracy Weighted, Macro Precision, Macro Recall, Macro F1-Score

---

```

1. Initialize: total_weight = 0, weighted_accuracy_sum = 0
2. Initialize: precision_sum = 0, recall_sum = 0
3. for k = 1 to K do
  a.  $TP_k \leftarrow CM[k][k]$ 
  b.  $FP_k \leftarrow \text{sum}(CM[:,k]) - TP_k$ 
  c.  $FN_k \leftarrow \text{sum}(CM[k,:]) - TP_k$ 
  d.  $\text{Total\_row}_k \leftarrow \text{sum}(CM[k,:])$ 
  e.  $w_k \leftarrow \text{Total\_row}_k$ 
  f.  $\text{total\_weight} \leftarrow \text{total\_weight} + w_k$ 
  g.  $\text{precision}_k \leftarrow TP_k / (TP_k + FP_k)$ 
  h.  $\text{recall}_k \leftarrow TP_k / (TP_k + FN_k)$ 
  i.  $\text{weighted\_accuracy\_sum} \leftarrow \text{weighted\_accuracy\_sum} + (TP_k * w_k / \text{Total\_row}_k)$ 
  j.  $\text{precision\_sum} \leftarrow \text{precision\_sum} + \text{precision}_k$ 
  k.  $\text{recall\_sum} \leftarrow \text{recall\_sum} + \text{recall}_k$ 
4.  $\text{Balanced Accuracy Weighted} \leftarrow \text{weighted\_accuracy\_sum} / (K * \text{total\_weight})$ 
5.  $\text{Macro Precision} \leftarrow \text{precision\_sum} / K$ 
6.  $\text{Macro Recall} \leftarrow \text{recall\_sum} / K$ 
7.  $\text{Macro F1} \leftarrow (2 * \text{Macro Precision} * \text{Macro Recall}) / (\text{Macro Precision} + \text{Macro Recall})$ 
8. Return Balanced Accuracy Weighted, Macro Precision, Macro Recall, Macro F1

```

---

### 3. RESULTS AND DISCUSSION

This section presents and analyzes the outcomes of the experimental workflow described in section 2. The discussion encompasses data acquisition, preprocessing, data segmentation for training and testing, and the effects of varying batch sizes and segment lengths. Additionally, performance evaluations are carried out using the metrics defined in (2)–(7).

#### 3.1. Data collection and pre-processing

Figure 6 illustrates the time-series sensor data recorded from Subject-7, capturing three activity classes and eight distinct measurements. Among these, the energy accumulation (E) data is included solely for visualization purposes and is not used as a model feature. The stop and forward drive (drove) activities are not depicted in this figure. In the stop activity, sensor readings, especially from the accelerometer, exhibit minimal fluctuation, indicating the vehicle is stationary. Electrical parameters such as current (I), power (P), and motor speed drop to near zero, while the voltage (V) remains constant. These characteristics serve as effective indicators for the stop condition. In contrast, the forward driving activity is inferred through changes in acceleration followed by braking sequences. An example can be seen when the motor is accelerated prior to braking. During such events, increases in current and power values signal the onset of acceleration.

The left and right turning activities produce similar dynamic patterns across the same set of sensors, making visual differentiation challenging. As shown in Figure 6, for instance, the (current, left) and (current, right) plots display nearly identical signal shapes—any perceived differences are primarily due to differences in axis scaling. On the other hand, the braking activity demonstrates clearer distinguishing features. For instance, the braking phase begins after the vehicle accelerates to approximately 60 km/h. Braking reduces the speed to around 20 km/h, followed by another acceleration phase. These transitions are clearly observable through rising current and power values, indicative of motor activity.

Figure 7 provides an overview of sample distributions per subject and per activity. Each subject performed every activity continuously for approximately three minutes. Given the embedded system's effective sampling rate of 3.33 Hz, each activity yields approximately 600 samples per subject, as shown in Figure 7(a). Aggregating data from all 10 subjects results in a dataset of approximately 6,000 samples, as illustrated in Figure 7(b).

#### 3.2. Data slicing for training and testing

The classification task involves five activity classes derived from time-series data. From the preprocessing stage, a total of 30568 data points, each consisting of seven features, were transformed into overlapping segments using sliding windows. These segments were constructed with time-steps  $n=25, 51, 75$ ,

and a stride of two time-steps. As a result, the respective segment shapes formed were (15272, 7, 25), (15259, 7, 51), and (15247, 7, 75). The dataset was subsequently split into training (56%), validation (14%), and testing (30%) sets for model evaluation.

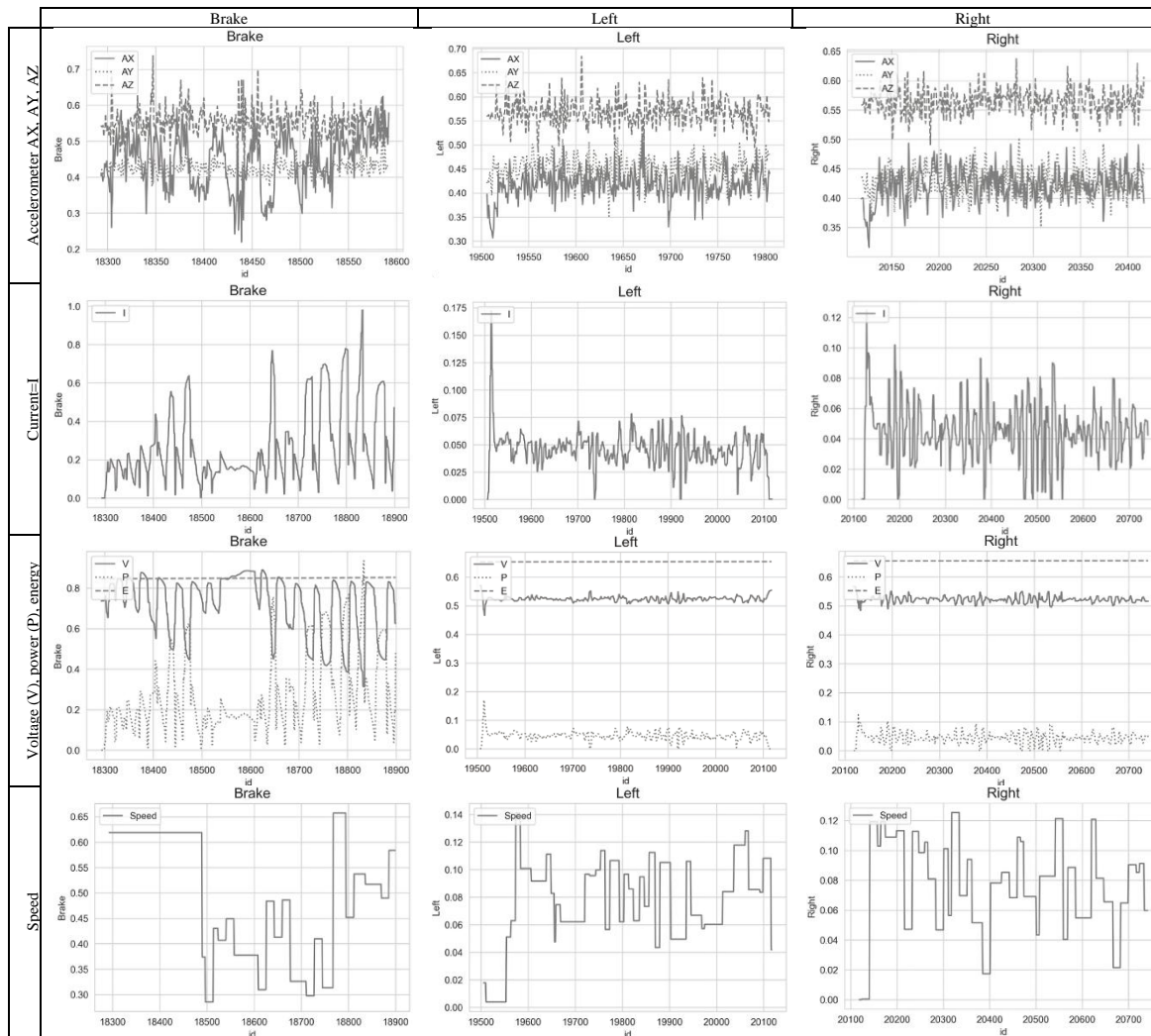


Figure 6. Sensor data for three activities from subject-7 (the x-axis (ID) represents the sampling index, while the y-axis shows normalized sensor values)

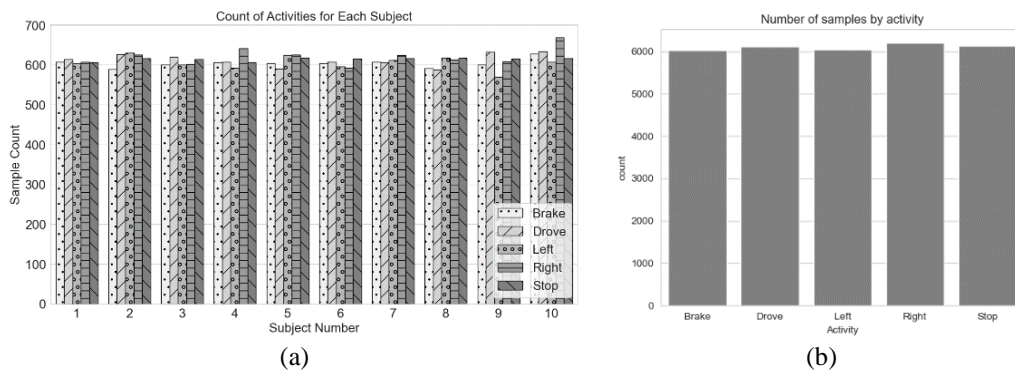


Figure 7. Number of samples and activities; (a) per subject and (b) per activity

### 3.3. Batch size

Two dynamic batch size scheduling techniques were evaluated: MSBU and MSBD. Figures 8 and 9 illustrate the training and validation loss and accuracy trends over 120 epochs for MSBU and MSBD, respectively. In the MSBU method (Figure 8), where the batch size increases progressively from 32 to 1024, the training loss rapidly decreases, and accuracy rises within the initial 20 epochs. This improvement continues steadily as the batch size increases, demonstrating effective optimization. Conversely, the MSBD method (Figure 9), which begins with a batch size of 1024 and decreases stepwise, shows limited improvement in training performance during the initial epochs. Notable gains in accuracy only emerge once the batch size reaches 512 or lower. Overall, MSBU consistently outperforms MSBD in both training and validation accuracy across the 120-epoch run.

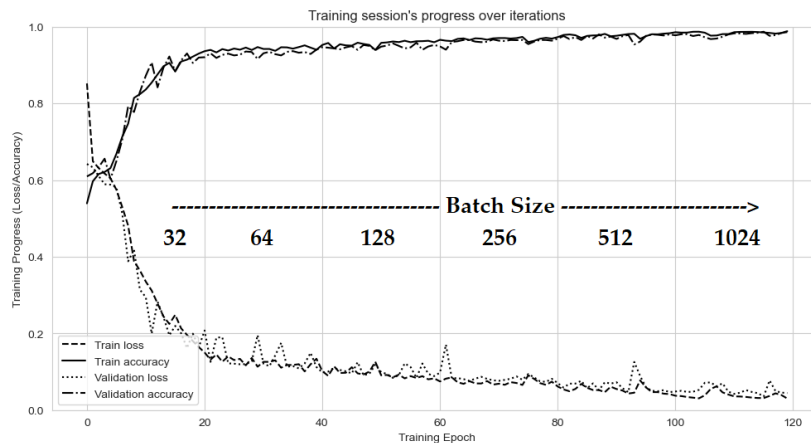


Figure 8. Training and validation performance for MSBU (batch size: 32 to 1024)

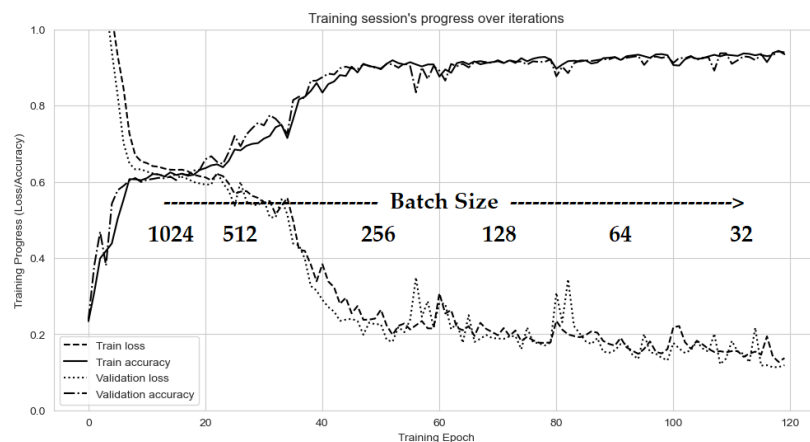


Figure 9. Training and validation performance for MSBD (batch size: 32 to 1024)

Additionally, two static batch size configurations 32 and 1024 were tested as baselines. Performance comparisons are summarized in Table 2. MSBU achieved the highest macro F1-score of 0.9873, outperforming both MSBD and static batch size experiments. Specifically, the F1-score improvements of MSBU over MSBD and static (32, 1024) are 0.0377, 0.0069, and 0.2123, respectively.

Table 2. Performance metrics for various batch size configurations (total epochs=120)

Experiment batch sizes	Accuracy	Macro average precision	Macro average recall	Macro average F1-score	Total training time (s)	Inference time (ms)
MSBU	0.9873	0.9873	0.9873	<b>0.9873</b>	<b>970</b>	19
MSBD	0.9493	0.9502	0.9492	0.9496	988	20
32 <sub>120</sub>	0.9803	0.9809	0.9805	<b>0.9804</b>	<b>1786</b>	20
1024 <sub>120</sub>	0.7786	0.8181	0.7796	0.7750	571	20

32<sub>20</sub> is the sizes of batches and epochs during the training are 32 and 20, respectively; MSBU is 32<sub>20</sub>, 64<sub>20</sub>, 128<sub>20</sub>, 256<sub>20</sub>, 512<sub>20</sub>, and 1024<sub>20</sub>; and MSBD is 1024<sub>20</sub>, 512<sub>20</sub>, 256<sub>20</sub>, 128<sub>20</sub>, 64<sub>20</sub>, and 32<sub>20</sub>.

Training durations for MSBU and MSBD are similar; however, MSBU achieves comparable accuracy to the static batch size of 32 while reducing training time by approximately 1.84 times. Notably, inference time remains stable at ~20 ms across all configurations, as it depends more on model architecture and segment size than on training batch size. Thus, MSBU is a favorable strategy for achieving both high accuracy and reduced training overhead. Moreover, the MSBU method surpassed the HAR baseline in [27] by a margin of 0.0253 in F1-score, reinforcing its effectiveness.

### 3.4. Impact of segment length (N time-steps)

The segment size, or time-step length N, functions as a critical hyperparameter in LSTM models for capturing temporal dependencies. Table 3 presents the classification performance for segment sizes N=25, 51, and 75, where increasing N leads to consistently higher accuracy and F1-scores. As expected, longer segments enhance the LSTM's memory of past events, which improves classification performance. However, this also increases training and inference times. The optimal segment length must therefore balance performance requirements and computational constraints.

Table 3. Performance metrics for varying segment lengths using MSBU

Experiment N step segment size	Accuracy	Macro average precision	Macro average recall	Macro average F1-score	Total training time(s)	Inference time (ms)
25	0.9603	0.9612	0.9596	0.9602	382	8
51	0.9760	0.9760	0.9757	0.9757	654	14
75	0.9873	0.9873	0.9873	0.9873	970	19

Confusion matrices for each segment length are depicted in Figure 10. These matrices highlight classification tendencies and reveal that the most frequent misclassifications occur between left and right turning activities. This observation aligns with the earlier analysis in Figure 6, where sensor signals for these classes appeared highly similar. Error counts for segment lengths of 25, 51, and 75 are 126, 87, and 45 samples, respectively, indicating that larger segment lengths significantly reduce class confusion. Figures 10(a) to (c) show the results of experiments with N of 25, 51, and 75 respectively.

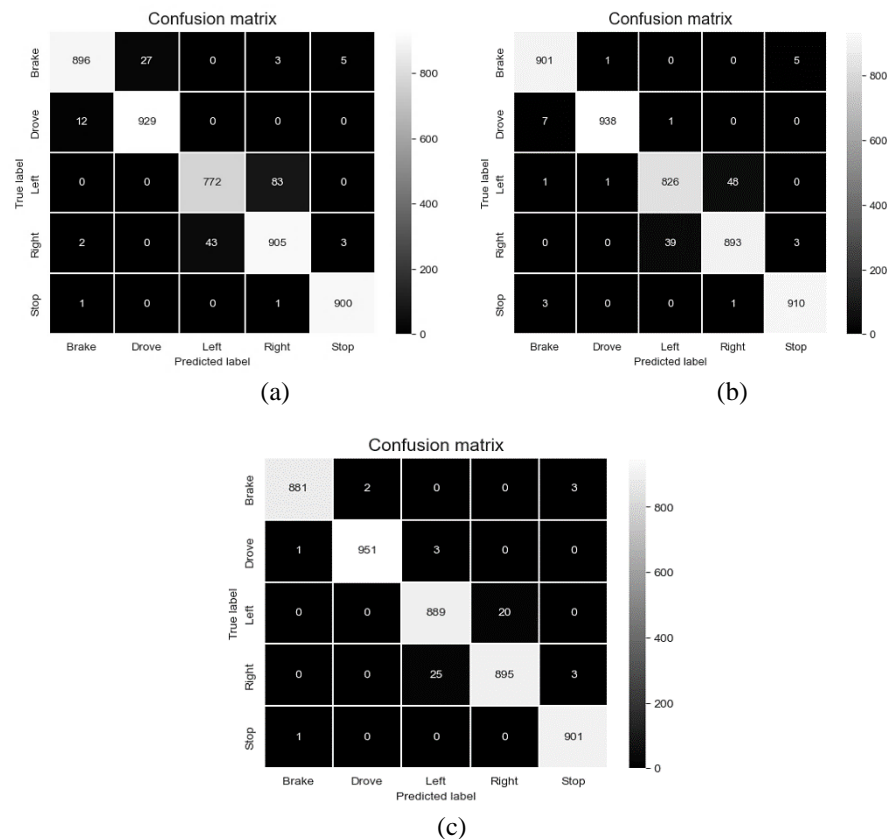


Figure 10. Confusion matrix for segment sizes; (a) 25, (b) 51, and (c) 75 with batch size from 32 to 1024

The software for data acquisition on the ESP32 microcontroller was developed using the Arduino IDE. Model training and evaluation were conducted on a machine equipped with an Intel i7-4720HQ processor and 16 GB RAM, with GPU acceleration disabled. The experimental pipeline utilized the following software libraries and frameworks: Keras, TensorFlow, Jupyter Notebook, Matplotlib, and Scikit-learn, along with their supporting APIs.

#### 4. CONCLUSION

This study successfully developed a driver activity recognition system for electric motorbikes using a self-constructed dataset comprising data from 10 subjects. The dataset includes seven sensor features: three-axis acceleration (ax, ay, and az), motor speed, voltage, current, and electric power, all recorded through an embedded EDR system. The classification task targets five distinct driver activities and is implemented using an LSTM neural network.

The impact of batch size scheduling and segment (window) length on model performance was systematically investigated. Three batch size strategies including MSBU, MSBD, and fixed batch sizes were evaluated. Results demonstrate that MSBU significantly accelerates training, achieving a 1.84× reduction in training time compared to a fixed small batch size, while maintaining high classification performance.

Furthermore, experiments with varying segment lengths show that longer sequences lead to improved recognition accuracy and F1-score. Specifically, a segment length of 75 time steps yielded the highest performance, with both accuracy and macro F1-score reaching 0.9873. These findings highlight the importance of dynamic batch sizing and appropriate temporal windowing in optimizing LSTM-based activity recognition systems for electric motorbike applications.

Despite its effectiveness in enabling real-time data acquisition and activity recognition on embedded platforms, DARSEM has several limitations. First, due to the computational constraints of the ESP32 microcontroller, model training cannot be conducted on-device, limiting the system to offline training and pre-deployed inference only. Additionally, the data labeling process depends on manual input via buttons and joystick controls, which introduces the possibility of human error and inconsistent timing during annotation. The dataset itself, while collected from real driving scenarios, is limited in size and diversity, comprising only ten participants and a specific set of terrain conditions, thus potentially affecting the generalizability of the trained model. Furthermore, the system has been developed specifically for electric motorbikes, and its applicability to other vehicle types may require significant hardware and software adaptation. These limitations suggest avenues for future work, including more automated labeling mechanisms, broader data collection efforts, and hardware-agnostic system designs to enhance scalability and performance.

#### ACKNOWLEDGMENTS

The authors extend heartfelt gratitude to the Directorate of Research and Community Service at Satya Wacana Christian University for their financial support.

#### FUNDING INFORMATION

This research received financial support from the Directorate of Research and Community Service at Satya Wacana Christian University.

#### 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
Darmawan Utomo	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
Natanael Indria		✓	✓	✓		✓	✓	✓	✓	✓	✓			
Prambodo														
Budihardja Murtianta	✓									✓		✓		✓

C : **C**onceptualization

M : **M**ethodology

So : **S**oftware

Va : **V**alidation

Fo : **F**ormal analysis

I : **I**nvestigation

R : **R**esources

D : **D**ata Curation

O : Writing - **O**riginal Draft

E : Writing - Review & **E**editing

Vi : **V**isualization

Su : **S**upervision

P : **P**roject administration

Fu : **F**unding acquisition

*Driver activity recognition using deep learning based on multi-step batch size up (Darmawan Utomo)*

## CONFLICT OF INTEREST STATEMENT

The authors declare that they have no conflict of interest.

## DATA AVAILABILITY

The data that support the findings of this study are available from the corresponding author, [initials: DU], upon reasonable request.

## REFERENCES




- [1] D. o. D. Statistics, *Land Transportation Statistics 2021*. Jakarta: BPS-Statistics Indonesia: BPS-Statistics Indonesia, 2022.
- [2] J. W. Nazemetz, F. D. Bents, J. G. Perry, C. Thor, and Y. M. Mohamedshah, "Motorcycle Crash Causation Study: Final Report," US Department of Transportation, Old Georgetown Pike, 2019.
- [3] World Health Organization, *Global status report on road safety 2023*. Geneva, Switzerland: WHO Press, 2023. [Online]. Available: <https://www.who.int/publications/b/68866>.
- [4] P. G. Wright, "The analysis of Accident Data Recorder (ADR) data in Formula 1," *SAE Technical Papers*, pp. 2520–2530, 2000, doi: 10.4271/2000-01-3551.
- [5] A. Łebkowski, "Electric Vehicle Data Recorder," *Przegląd Elektrotechniczny*, vol. 1, no. 2, pp. 286–290, 2017, doi: 10.15199/48.2017.02.62.
- [6] T. Komamizu, T. Amagasa, and H. Kitagawa, "Visual Spatial-OLAP for Vehicle Recorder Data on Micro-sized Electric Vehicles," in *Proceedings of the 20th International Database Engineering & Applications Symposium on - IDEAS '16*, New York, NY, USA: ACM Press, 2016, pp. 358–363, doi: 10.1145/2938503.2938532.
- [7] K. Böhm, T. Kubjatko, D. Paula, and H. G. Schweiger, "New developments on EDR (Event Data Recorder) for automated vehicles," *Open Engineering*, vol. 10, no. 1, pp. 140–146, Mar. 2020, doi: 10.1515/eng-2020-0007.
- [8] I. D. Buzdugan, S. Butnariu, I. A. Roşu, A. C. Pridie, and C. Antonya, "Personalized Driving Styles in Safety-Critical Scenarios for Autonomous Vehicles: An Approach Using Driver-in-the-Loop Simulations," *Vehicles*, vol. 5, no. 3, pp. 1149–1166, Sep. 2023, doi: 10.3390/vehicles5030064.
- [9] C. Ruseruka, J. Mwakalonge, G. Comert, S. Siuhi, and J. Perkins, "Road Condition Monitoring Using Vehicle Built-in Cameras and GPS Sensors: A Deep Learning Approach," *Vehicles*, vol. 5, no. 3, pp. 931–948, Aug. 2023, doi: 10.3390/vehicles5030051.
- [10] J. Zhao, H. Ling, J. Liu, J. Wang, A. F. Burke, and Y. Lian, "Machine learning for predicting battery capacity for electric vehicles," *eTransportation*, vol. 15, p. 100214, Jan. 2023, doi: 10.1016/j.etrans.2022.100214.
- [11] G. Caruso, M. K. Yousefi, and L. Mussone, "From Human to Autonomous Driving: A Method to Identify and Draw Up the Driving Behaviour of Connected Autonomous Vehicles," *Vehicles*, vol. 4, no. 4, pp. 1430–1449, Dec. 2022, doi: 10.3390/vehicles4040075.
- [12] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, doi: 10.1162/neco.1997.9.8.1735.
- [13] S. Jawed, I. Faye, and A. S. Malik, "Deep Learning-Based Assessment Model for Real-Time Identification of Visual Learners Using Raw EEG," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 32, pp. 378–390, 2024, doi: 10.1109/TNSRE.2024.3351694.
- [14] V. Hnamte, H. Nhung-Nguyen, J. Hussain, and Y. Hwa-Kim, "A Novel Two-Stage Deep Learning Model for Network Intrusion Detection: LSTM-AE," *IEEE Access*, vol. 11, pp. 37131–37148, 2023, doi: 10.1109/ACCESS.2023.3266979.
- [15] L. Liu, P. Wang, J. Lin, and L. Liu, "Intrusion Detection of Imbalanced Network Traffic Based on Machine Learning and Deep Learning," *IEEE Access*, vol. 9, pp. 7550–7563, 2021, doi: 10.1109/ACCESS.2020.3048198.
- [16] I. D. Mienye and N. Jere, "Deep Learning for Credit Card Fraud Detection: A Review of Algorithms, Challenges, and Solutions," *IEEE Access*, vol. 12, pp. 96893–96910, 2024, doi: 10.1109/ACCESS.2024.3426955.
- [17] N. Q. Do, A. Selamat, O. Krejcar, E. Herrera-Viedma, and H. Fujita, "Deep Learning for Phishing Detection: Taxonomy, Current Challenges and Future Directions," *IEEE Access*, vol. 10, pp. 36429–36463, 2022, doi: 10.1109/ACCESS.2022.3151903.
- [18] T. Rahman, M. K. A. Al-Ruweidi, M. S. I. Sumon, R. Y. Kamal, M. E. H. Chowdhury, and H. C. Yalcin, "Deep Learning Technique for Congenital Heart Disease Detection Using Stacking-Based CNN-LSTM Models from Fetal Echocardiogram: A Pilot Study," *IEEE Access*, vol. 11, pp. 110375–110390, 2023, doi: 10.1109/ACCESS.2023.3316719.
- [19] S. Iqbal *et al.*, "Prostate Cancer Detection Using Deep Learning and Traditional Techniques," *IEEE Access*, vol. 9, pp. 27085–27100, 2021, doi: 10.1109/ACCESS.2021.3057654.
- [20] T. Zengya and J. V. Fonou-Dombeu, "A Review of State of the Art Deep Learning Models for Ontology Construction," *IEEE Access*, vol. 12, pp. 82354–82383, 2024, doi: 10.1109/ACCESS.2024.3406426.
- [21] S. Abbas, G. A. Sampedro, M. Krichen, M. A. Alamro, A. Mihoub, and R. Kulhanek, "Effective Hypertension Detection Using Predictive Feature Engineering and Deep Learning," *IEEE Access*, vol. 12, pp. 89055–89068, 2024, doi: 10.1109/ACCESS.2024.3418553.
- [22] X. Jiawei, Z. Xufang, L. Zhong, and X. Qingtao, "LSTM-DPPO based deep reinforcement learning controller for path following optimization of unmanned surface vehicle," *Journal of Systems Engineering and Electronics*, vol. 34, no. 5, pp. 1343–1358, Oct. 2023, doi: 10.23919/JSEE.2023.000113.
- [23] S. Bilotta, L. A. I. Palesi, and P. Nesi, "Predicting Free Parking Slots via Deep Learning in Short-Mid Terms Explaining Temporal Impact of Features," *IEEE Access*, vol. 11, pp. 101678–101693, 2023, doi: 10.1109/ACCESS.2023.3314660.
- [24] M. Bilal, A. Khan, S. Jan, and S. Musa, "Context-Aware Deep Learning Model for Detection of Roman Urdu Hate Speech on Social Media Platform," *IEEE Access*, vol. 10, pp. 121133–121151, 2022, doi: 10.1109/ACCESS.2022.3216375.
- [25] K. L. Tan, C. P. Lee, K. M. Lim, and K. S. M. Anbananthen, "Sentiment Analysis With Ensemble Hybrid Deep Learning Model," *IEEE Access*, vol. 10, pp. 103694–103704, 2022, doi: 10.1109/ACCESS.2022.3210182.
- [26] Z. Yang, M. Qu, Y. Pan, and R. Huan, "Comparing Cross-Subject Performance on Human Activities Recognition Using Learning Models," *IEEE Access*, vol. 10, pp. 95179–95196, 2022, doi: 10.1109/ACCESS.2022.3204739.
- [27] M. Milenkoski, K. Trivodaliev, S. Kalajdziski, M. Jovanov, and B. R. Stojkoska, "Real time human activity recognition on smartphones using LSTM networks," in *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2018 - Proceedings*, Opatija, Croatia, 2018, pp. 1126–1131, doi:






- 10.23919/MIPRO.2018.8400205.
- [28] S. Mekruksavanich and A. Jitpattanakul, "Smartwatch-based Human Activity Recognition Using Hybrid LSTM Network," in *Proceedings of IEEE Sensors*, IEEE, Oct. 2020, pp. 1–4, doi: 10.1109/SENSOR47125.2020.9278630.
- [29] K. Peppas, A. C. Tsolakis, S. Krinidis, and D. Tzovaras, "Real-time physical activity recognition on smart mobile devices using convolutional neural networks," *Applied Sciences*, vol. 10, no. 23, pp. 1–25, Nov. 2020, doi: 10.3390/app10238482.
- [30] Z. Hu *et al.*, "A variable batch size strategy for large scale distributed dnn training," in *Proceedings - 2019 IEEE Intl Conf on Parallel and Distributed Processing with Applications, Big Data and Cloud Computing, Sustainable Computing and Communications, Social Computing and Networking, ISPA/BDCLOUD/SustainCom/SocialCom 2019*, Xiamen, China, 2019, pp. 476–485, doi: 10.1109/ISPA-BDCLOUD-SustainCom-SocialCom48970.2019.00074.
- [31] K. W. Lu, P. Liu, D. Y. Hong, and J. J. Wu, "Efficient Dual Batch Size Deep Learning for Distributed Parameter Server Systems," in *Proceedings - 2022 IEEE 46th Annual Computers, Software, and Applications Conference, COMPSAC 2022*, Los Alamitos, CA, USA, 2022, pp. 630–639, doi: 10.1109/COMPSAC54236.2022.00110.
- [32] C. Yu, Q. Li, F. Feng, and Q. J. Zhang, "Convolutional Neural Network With Adaptive Batch-Size Training Technique for High-Dimensional Inverse Modeling of Microwave Filters," *IEEE Microwave and Wireless Technology Letters*, vol. 33, no. 2, pp. 122–125, Feb. 2023, doi: 10.1109/LMWC.2022.3208355.
- [33] T. Kidu, Y. Song, K. W. Seo, S. Lee, and T. Park, "An Intelligent Real-Time Driver Activity Recognition System Using Spatio-Temporal Features," *Applied Sciences*, vol. 14, no. 17, pp. 1–25, Sep. 2024, doi: 10.3390/app14177985.
- [34] L. Yan, L. Jia, S. Lu, L. Peng, and Y. He, "LSTM-based deep learning framework for adaptive identifying eco-driving on intelligent vehicle multivariate time-series data," *IET Intelligent Transport Systems*, vol. 18, no. 1, pp. 186–202, Jan. 2024, doi: 10.1049/itr2.12443.
- [35] M. Grandini, E. Bagli, and G. Visani, "Metrics for Multi-Class Classification: an Overview," *arXiv*, pp. 1–17, 2020, doi: 10.48550/arXiv.2008.05756.

## BIOGRAPHIES OF AUTHORS






**Darmawan Utomo**    received his B.S. degree from Satya Wacana Christian University in electrical engineering, in 1993, the M.Eng. degree from Asian Institute of Technology, in 1999, and the Ph.D. degree from National Chung Cheng University, in 2021, both in computer science. He is an Associate Professor with the Department of Electronics and Computer Engineering, from 1994 until now. His work has been funded by the State Research and Development, local companies, and university. He has published over seven articles in deep learning and electronic applications. He can be contacted at email: darmawan.utomo@uksw.edu.



**Natanael Indria Prambodo**    received the B.S. degree in Computer Engineering from Satya Wacana Christian University, Salatiga, Indonesia, in 2024. In 2023, he was an intern as a Software Engineer. His research interests include internet of things (IoT), development, and deep learning. He can be contacted at email: natanael67176@gmail.com.



**Budihardja Murtianta**    received the B.S. degree from Satya Wacana Christian University, in 1985, the M.Eng. degree from Asian Institute of Technology Thailand, in 1986, in Telecommunication Engineering. In 1986, he joined Satya Wacana Christian University, where he is currently as lecturer with the Department of Electronic and Computer Engineering. He also directs the Electronic and Computer Lab. He can be contacted at email: budihardja.murtianta@uksw.edu.