Learning Temporal Graph Representations for Intelligent Control in 3D Endless Runner Games

Oddy Virgantara Putra[1*], Daffa Sesa Rabbani[2], Alvin Fredericco[3], Arsyapradana Fadlanabil Bahri[4]

[1,2,3,4]Department of Informatics, Faculty of Science and Technology, Universitas Darussalam Gontor, Ponorogo, East Java, Indonesia
E-mail: [1*]oddy@unida.gontor.ac.id, [2]daffasesarabbani33@student.cs.unida.gontor.ac.id, [3]alvinfredericco85@student.cs.unida.gontor.ac.id, [4]arsyapradanafadlanabilbahri85@student.cs.unida.gontor.ac.id

(Received: 11 Nov 2025, revised: 24 Dec 2025, accepted: 8 Jan 2026)

Abstract

Recent advancements in deep learning have significantly enhanced body gesture recognition, enabling real-time interaction between humans and machines through the modeling of spatial–temporal features. However, many existing approaches primarily rely on frame-based or visual feature representations and are often evaluated in offline settings, which limits their stability and responsiveness when applied to real-time 3D game environments that require continuous and dynamic player movement. In this paper, we develop a gesture-controlled endless runner game using a skeleton-based Graph Neural Network–Long Short-Term Memory (GNN–LSTM) model. The proposed system enables real-time interaction without the need for conventional input devices and is directly integrated into a Unity-based game environment. A dataset of 1,000 gesture videos across five classes (Jump In Place, Jump Left, Jump Right, Looking Down, and Still Pose) is processed using MediaPipe Pose to extract 33 body keypoints per frame, which are then normalized and represented as graph structures to capture spatial and temporal motion patterns. Experimental results show that the GNN–LSTM model achieves a validation accuracy of up to 97.5% and a test accuracy of 96%. Although CNN–LSTM attains slightly higher test accuracy, the GNN–LSTM model demonstrates more stable validation performance and robustness by leveraging skeleton-based representations, making it more suitable for real-time gesture control in interactive gameplay. Integrated with Unity, the proposed system allows intuitive and responsive control of character movements during gameplay. These findings highlight the effectiveness of temporal graph-based representations for stable and natural gesture-based human–computer interaction in real-time 3D games.

Keywords: Gesture Recognition, Graph Neural Network, Long Short-Term Memory, MediaPipe, Endless Runner.

I. INTRODUCTION

The rapid development of deep learning has significantly advanced body gesture recognition by enabling automatic extraction and interpretation of human movement patterns [1]. Gesture recognition plays an important role in various applications, including human-computer interaction, motion-based game control, physical rehabilitation, and virtual reality (VR) system [2]. A key component of gesture recognition is human body pose estimation, which serves as the basis for motion feature extraction. Recent studies have shown that spatial–temporal models such as Graph Neural Networks (GNN) and Long Short-Term Memory (LSTM) are effective in capturing complex body movement dynamics [3].

Despite these advances, several limitations remain when gesture recognition systems are applied to interactive and real-time environments. Rule-based approaches often lack flexibility and fail to generalize across different users and movement variations. CNN-based methods mainly rely on visual appearance features without explicitly modeling skeletal structure, making them sensitive to background clutter, lighting variations, and camera viewpoints. Moreover, many existing studies focus on offline evaluation and do not sufficiently address latency and prediction stability, which are critical for real-time gameplay requiring continuous and responsive control. These challenges indicate the need for a skeleton-based spatial–temporal modeling approach that can provide stable and efficient gesture recognition in real-time 3D game environments [4].

To address these issues, this study proposes a skeleton-based gesture recognition system that combines Graph Neural Networks and Long Short-Term Memory (GNN-LSTM) and integrates it into a 3D endless runner game. Body pose data are extracted using MediaPipe Pose, represented as skeleton graphs, and processed to learn spatial-temporal motion patterns for real-time character control without conventional

DOI: 10.34148/teknika.v15i1.1416

input devices. The proposed system is evaluated on five gesture classes such as jump_in_place, jump_left, jump_right, looking_down, and still_pose, and integrated with the Unity game engine through a real-time communication mechanism. The contributions of this work include the development of a real-time skeleton-based gesture control framework, its seamless integration with a Unity-based game environment, and a comprehensive evaluation demonstrating its effectiveness and stability for interactive gameplay.

II. RELATED WORKS

2.1 GNN-LSTM for Gesture Classification

In the context of recognizing human body movements, graphs are used to represent the body frame or skeleton, where nodes represent joints, and edges represent logical or anatomical connections between joints [5]. This graph representation enables the processing of spatial and temporal relationships between body parts using Graph Neural Networks (GNN), which can effectively propagate information between nodes and capture complex body movement patterns.

Research by Abbate in 2024 proposed a self-supervised learning approach to predict human interaction intentions with robots based on body movements using body frame data and Graph Neural Networks (GNN) [6]. Another approach uses Graph Neural Networks (GNN) to leverage the spatial structure of skeleton data. Yan et al. introduced ST-GCN (Spatial-Temporal Graph Convolutional Network), [7] which effectively captures spatial-temporal skeleton patterns and achieves superior results on the NTU-RGBD dataset.

While ST-GCN has demonstrated strong performance in skeleton-based action recognition by modeling spatial–temporal joint dependencies [8], its deep graph convolutional architecture often introduces higher computational complexity and inference latency, which can limit its suitability for real-time interactive applications on consumer-grade hardware [9]. In this study, several alternative temporal models, including CNN–LSTM, LSTM, Bi-LSTM, and TCN, were evaluated. Although CNN–LSTM and Bi-LSTM achieved high accuracy, CNN–LSTM relies heavily on visual features, while TCN showed lower accuracy and less stable class separation. Therefore, GNN–LSTM was selected as a balanced architecture that effectively models skeletal relationships while maintaining lower latency and stable performance, making it more suitable for real-time gesture-based game control [10].

2.2 Endless Runner Game

Research conducted by D. Diffran Nur Cahyo, Sur Liyan, and Muhammad Zainuri (2023) used the System Usability Scale (SUS) method in designing the Endless Runner Mathematics Game User Interface. This method can measure the level of usability and user satisfaction with the game interface. [11] The results showed an average score of 88.92, which falls into the “very good” category, indicating that the interface was very well designed in terms of navigation, readability, and user interaction efficiency.

In another study, Pertiwi et. al. [12] examined the aspects of playability and user experience (UX) in local endless runner games. The study of the game “Joko Run” produced a factor analysis of playability on UX in Android endless runner games and found that this type of game is very influential in terms of control, feedback, and challenges that have a positive impact on the player's experience.

2.3 Gesture-Based Game Development

In addition to research on gesture classification and endless runner games, several studies have also focused on the application of gesture recognition in game development. Bakar et al. developed a gesture-based endless runner mobile game using Unity and the ManoMotion SDK, which received positive responses from users despite still facing control responsiveness issues [13].

Another study by Schmidt, who implemented a gesture-controlled Jump and Run game at an educational event, proved that the game could attract attention and increase user engagement [14].

III. PROPOSED WORK

3.1 System Architecture

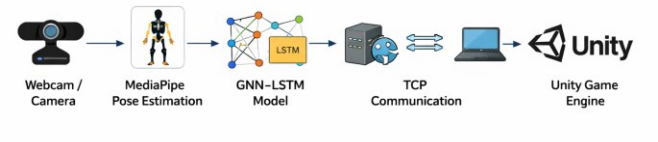

Figure 1. System Architecture of The Proposed Gesture-based Game Control system

The system architecture in this study is designed to integrate deep learning-based body gesture recognition with an endless runner game developed using Unity. The overall workflow of the proposed system is illustrated in Figure 1. System Architecture of The Proposed Gesture-based Game Control system. In general, the system workflow consists of three main stages, namely: (1) acquisition of body gesture data through a camera, (2) processing and classification of gestures using the Graph Neural Network–Long Short-Term Memory (GNN-LSTM) model, and (3) implementation of classification results as actions in the Unity game.

In the first stage, the camera records the player's body movements in real-time. The data obtained is in the form of digital images which are then processed into body skeleton coordinates using MediaPipe. The extraction results are in the form of keypoints which each represent the position of joints in the human body. This keypoint data is then represented in the form of a graph which is used as input for the classification process.

The second stage is the gesture classification process using the GNN-LSTM model. This model focuses on understanding the spatial relationships between body joints (nodes) through

DOI: 10.34148/teknika.v15i1.1416

Graph Neural Network (GNN), as well as the temporal patterns of body movements between frames through Long Short-Term Memory (LSTM). The model is trained to recognize five main gesture classes, namely Jump In Place, Jump Left, Jump Right, Looking Down, and Still Pose.

The third stage is the integration of classification results with the Unity game. The model output in the form of gesture labels is translated into character actions in the game, for example, Jump In Place as a command to jump on the same path, or Jump Left as a command to move to the left path while jumping. Thus, the game can be fully controlled through the player's body gestures without the need for conventional control devices such as a keyboard or gamepad.

3.2 Notation and Problem Formulation

This research focuses on recognizing human body gestures represented in the form of skeleton data sequences. Each gesture video consists of a series of frames containing the positions of body joints in the form of three-dimensional coordinates (x, y, z). These joint positions are represented as graphs to model the spatial relationships between parts of the human body.

Each frame in the video sequence can be represented as an undirected graph, as can be seen in Equation (1 about the representation of the skeleton graph in frame t.

$$G_t = (V, E, X_t) \quad (1)$$

$V = \{v_1, v_2, \dots, v_N\}$ is a set of nodes representing body joints, $E \subseteq V \times V$ is a set of edges describing anatomical connections between joints [15], and $X_t \in R_N \times F$ is a feature matrix at time t containing the spatial coordinates of each node. The number of nodes N=33 follows the number of keypoints extracted by MediaPipe Pose, while the number of features per node F=3 represents the coordinates (x, y, z).

The frame sequence throughout the video is denoted as $\{G_1, G_2, \dots, G_T\}$ with T=50 after the padding process so that the sequence length is uniform. The main objective of the learning process is to learn the classification function. The mapping of function GNN-LSTM can be modelled as follows:

$$f_\theta : \{G_1, G_2, \dots, G_T\} \rightarrow y \quad (2)$$

In the model mapping function shown in Equation (2, the parameter θ is trained so that it is able to map the skeleton graph sequence to gesture class labels $y \in \{1,2,3,4,5\}$.

The five gesture classes used in this study include Jump In Place, Jump Left, Jump Right, Looking Down, and Still Pose. The GNN model is used to capture the spatial relationships between nodes in each frame, while the LSTM is tasked with learning the temporal relationships between frames in a movement sequence.

3.3 Dataset

The dataset used in this study is a collection of human body movement video data recorded directly using a camera. The dataset consists of five main gesture classes, namely Jump In Place, Jump Left, Jump Right, Looking Down, and Still Pose. Each class contains 200 video samples, bringing the total data to 1,000 samples with a balanced distribution in each class.

Each video was recorded under varying conditions to increase data diversity, including individual differences (male and female), lighting (day and night), setting (indoor and outdoor), and camera distance from the subject (3, 4, and 8 meters). This variation aims to enable the model to adapt to changes in the environment and different body characteristics. For model training and testing purposes, the dataset was divided into two parts with a ratio of 80:20, namely 798 samples for the training set and 200 samples for the testing set. The separation was carried out evenly per class so that the distribution remained proportional. This dataset became the basis for the GNN-LSTM model in learning spatial and temporal patterns for each type of body gesture. The class distribution in the dataset can be seen in Table 1.

Table 1. Dataset

No	Class Name	Amount of Data
1	jump_in_place	200
2	jump_left	200
3	jump_right	200
4	looking_down	200
5	still_pose	200
Total		1000

3.4 Preprocessing and Feature Extraction

This preprocessing stage consists of four main steps: keypoint extraction using MediaPipe, data normalization, frame sequence padding, and skeleton graph structure formation.

In the keypoint extraction stage, each video frame is processed using MediaPipe Pose to detect 33 human body joints. Each joint is represented by three spatial coordinates (x, y, z), resulting in a 33×3matrix per frame. Frames in which MediaPipe fails to reliably detect body keypoints are discarded. This decision is made to preserve spatial consistency and avoid introducing noisy or incomplete skeletal representations that could negatively affect the learning process. Since such detection failures occur sporadically and do not form long consecutive gaps, their removal does not significantly disrupt the temporal continuity of the gesture sequence.

For data normalization, all extracted coordinate values are normalized using the StandardScaler method to ensure that each feature has a distribution with a mean of zero and a standard deviation of one. The normalization process follows the equation shown in Equation 3.

$$x' = \frac{x - \mu}{\sigma} \quad (3)$$

In Equation 3, the value of x is the original value, μ is the mean value, and σ is the standard deviation of all coordinate points in the dataset. This normalization is done to speed up convergence during model training and avoid the dominance of features with large value scales.

To ensure uniform input length across all samples, frame sequence padding is applied so that each video contains a fixed length of 50 frames. If a video sequence contains fewer than 50 valid frames, zero-padding is applied at the end of the sequence. Conversely, sequences longer than 50 frames are truncated to the first 50 frames. This strategy maintains temporal alignment between samples while preserving the early and most informative portion of the gesture motion. Zero-padding at the tail of the sequence minimizes its impact on temporal modeling, as the LSTM primarily learns from meaningful frame transitions occurring earlier in the sequence.

This padding strategy also contributes to temporal stability by allowing the model to process consistent sequence lengths while implicitly learning to ignore padded frames during temporal aggregation. As a result, all videos are represented as tensors of size $50 \times 33 \times 3$.

Finally, each frame is converted into a skeleton graph as defined in Equation (1) in Subsection 3.2. In this graph representation, nodes correspond to body joints, while edges follow the anatomical connections defined by the MediaPipe Pose topology. The node feature matrix X_t contains the normalized spatial coordinates for each joint at time step t. The output of this preprocessing stage is a temporally ordered sequence of 50 skeleton graphs per video, which serves as the primary input to the proposed GNN-LSTM model.

3.5 GNN-LSTM Model

The main model used in this study is a hybrid architecture that combines a Graph Neural Network (GNN) and a Long Short-Term Memory (LSTM) network to model both spatial and temporal characteristics of human body movements.

The GNN component consists of two GraphConv layers, each with a hidden dimension of 64 units, and uses the ReLU activation function to extract spatial features from the skeleton graph in each frame. This design allows the model to capture spatial dependencies among body joints based on their anatomical connections.

The output of the GNN for each frame is aggregated using global mean pooling, producing a fixed-length embedding vector. These embeddings are then arranged as a temporal sequence and fed into an LSTM network with 128 hidden units to model motion dynamics across frames.

Each graph convolution layer updates node features based on the basic GraphConv equation, where h_i^l is the feature vector of node i at layer l, $N(i)$ is the set of neighbors of node $i, W^{(l)}$ is the weight matrix, b^l is the bias, and σ is the non-linear activation function (ReLU). The GraphConv formula is shown in Equation (4:

$$h_i^{(l+1)} = \sigma\left(\sum_{j \in N(i)} W^{(l)} h_j^{(l)} + b^{(l)}\right) \quad (4)$$

After passing through two layers of GraphConv, each graph produces an embedding vector that is globally averaged using global mean pooling to obtain an overall representation of the body pose in that frame. The sequence of GNN embeddings from all frames is then sent to the LSTM layer to capture temporal dynamics or changes in body position over time. For more details, the GNN model architecture can be seen in Figure 2. GNN Architecture on Data Skeleton:

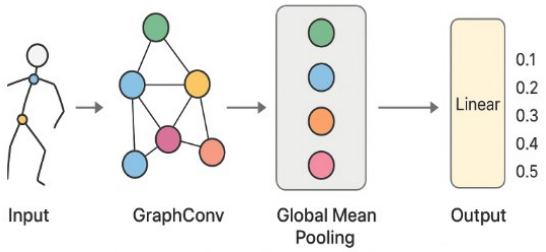

Figure 2. GNN Architecture on Data Skeleton

LSTM is used because of its ability to store long-term information and overcome the vanishing gradient problem in sequential data [16]. At this stage, LSTM processes the sequence of embedding vectors {z_1,z_2,...,z_T} and produces the final hidden state vector, h_T, which represents the overall body movement pattern throughout the video.

The LSTM output is then passed to a fully connected layer for classification. This layer converts the temporal-spatial representation into probabilities for each gesture class using the softmax function. This architecture allows the model to understand both the spatial relationships between body joints (via GNN) and the temporal relationships between frames (via LSTM), enabling it to recognize gesture patterns with high accuracy and stability.

3.5.1 Skeleton Data Representation and Graph Neural Network (GNN) Process

The skeleton data extracted using MediaPipe Pose is represented in graph form to utilize the spatial relationships between human joints. Each frame of the gesture video is converted into a graph G_t=(V,E,X_t), where node V represents body joints such as the head, shoulders, elbows, knees, and ankles, while edge E describes the relationships between joints according to the body's anatomical structure. Each node has three-dimensional features (x, y, z) that indicate the relative position of the joint to the camera, and these coordinate values have been normalized to be within a uniform scale range.

The connection structure between nodes follows the topology defined by MediaPipe, such as the relationship between the shoulder and elbow, hip and knee, and the vertical connection from the shoulder to the hip [17]. With this structure, each graph can represent the shape and pose of the body at any given time consistently.

The resulting graph then becomes the input for the Graph Neural Network (GNN) layer. Through the Graph Convolution (GraphConv) layer, the node feature update formula in the Graph Convolution layer, each node updates its representation based on information from connected neighboring nodes.

In this study, the GNN architecture employs two stacked GraphConv layers to progressively refine node representations. The first layer captures low-level spatial

DOI: 10.34148/teknika.v15i1.1416
TEKNIKA, Volume 15(1), March 2026, pp. 28-36
ISSN 2549-8037, EISSN 2549-8045

relationships between neighboring joints, while the second layer enhances higher-level structural patterns of the body pose. The ReLU activation function is applied after the graph convolution to introduce non-linearity and improve representation capacity.

3.5.2 Long Short-Term Memory (LSTM)

LSTM is used to learn the dynamics of body movements in a time sequence, so that the model not only understands the shape of a pose in a single frame, but also the changes between frames in sequence.

The LSTM network used in this model consists of a single LSTM layer with 128 hidden units, which is sufficient to capture temporal dependencies without introducing excessive computational overhead. A single-layer configuration was selected to balance model complexity and real-time inference requirements in interactive game environments.

LSTM has three main gates, namely the input gate, forget gate, and output gate, which regulate the flow of information within the memory cell. This mechanism allows LSTM to retain important information from the previous frame and forget irrelevant information.

In the context of this research, the sequence of embedding vectors generated by GNN ($z_1, z_2, \ldots, z_T$) is fed into LSTM as sequential input. The LSTM then combines the context from the entire sequence of body movements to produce the final output vector h_T, which represents the overall body movement pattern during a single gesture video. This representation is then used by the classification layer to determine the appropriate gesture label. The LSTM memory cell update process can be seen mathematically in Equation (5.

$$
\begin{aligned}
f_t &= \sigma(W_f[h_{t-1}, x_t] + b_f) \\
i_t &= \sigma(W_i[h_{t-1}, x_t] + b_i) \\
\tilde{C}_t &= \tanh(W_c[h_{t-1}, x_t] + b_c) \\
C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t \\
o_t &= \sigma(W_o[h_{t-1}, x_t] + b_o) \\
h_t &= o_t * \tanh(C_t)
\end{aligned}
\tag{5}
$$

In the memory cell update equation formula in the LSTM architecture shown in Equation 5, the value x_t is the input at time t, while h_t is the hidden state, and C_t is the cell memory state. The σ function represents sigmoid activation, while tanh f_t is the hyperbolic activation function.

Dropout regularization is not applied in the LSTM layer in this study. This decision is based on experimental observations showing stable convergence behavior and consistent validation accuracy across training epochs, indicating no significant overfitting. Therefore, additional regularization through dropout was considered unnecessary for the given dataset and model configuration.

3.5.3 Classification Layer

The final output of the LSTM is a hidden vector h_T, which serves as a comprehensive representation of body movement patterns in a single gesture video. This h_T vector is then passed to a fully connected layer (dense layer) to map it to the class space. This process can be written in the form of the equation $y = Softmax\,(W_C\, h_T + b_C)$.

W_c is the weight matrix, b_c is the bias, and the Softmax function converts linear scores into probability distributions for each gesture class. The class with the highest probability value will be selected as the final prediction result.

In this study, the classification layer produces five output classes, namely Jump In Place, Jump Left, Jump Right, Looking Down, and Still Pose. Each class represents a specific body movement pattern that can be recognized by the system.

The classification layer is also a key component that determines the final performance of the model during the training process, because the loss function value is calculated based on the difference between the prediction results and the actual labels using the Cross-Entropy Loss function. This approach is commonly used in multi-class classification problems, because it effectively measures the distance between the prediction probability and the ground truth label. An example of this is in the research conducted by Farhadpour in his study entitled Selecting and Interpreting Multiclass Loss and Accuracy Assessment Metrics for Classifications with Class Imbalance: Guidance and Best Practices [18].

3.5.4 Training Process

The dataset that has undergone pre-processing is divided into two parts, namely 80% training data and 20% test data, with stratified division to maintain the proportion of each class. The training process is carried out for 30 epochs using the Adam optimizer and Cross-Entropy Loss function.

Each training iteration consists of two main stages, namely forward pass and backward pass. In the forward pass stage, skeleton data in the form of a graph sequence is entered into the model to generate class probability predictions. The loss value is calculated based on the difference between the prediction results and the actual labels using Cross-Entropy Loss.

The backward pass stage is performed by calculating the gradient against the model parameters using the backpropagation through time (BPTT) method. The Adam optimizer then adaptively updates the model weights based on the gradient value with a learning rate of 0.001.

During the training process, the model is evaluated using accuracy and loss metrics on training data and validation data at each epoch. The evaluation results show that the GNN-LSTM model experienced a significant decrease in loss values from the initial to the final epoch, as well as a stable increase in validation accuracy above 95%. This pattern shows that the model is able to learn effectively without experiencing significant overfitting.

3.6 Game Environment Development

The development of the game environment and the creation of 3D assets were carried out using Blender and Unity Asset Store. In addition, the game environment in this study was also developed using a procedural content generation (PCG) approach to support endless runner-based games. The map in

DOI: 10.34148/teknika.v15i1.1416

the game was not designed statically, but was generated dynamically using the Perlin Noise algorithm, which produced variations in track shapes, obstacle distributions, and more natural visual elements.

Perlin noise is capable of producing pseudo-random but smooth patterns (smooth randomness), thereby creating continuous variations in the track, obstacle distribution, and additional objects. With this, the game always presents new challenges with non-repetitive patterns.

In addition, the game starts from the Start screen, then goes to the Splash Screen before the player is directed to the main menu. In the Menu section, players are given two options, namely Start or Exit. If the player chooses Start, the game starts immediately and the character moves according to body gesture controls.

During the game, the system records the player's score based on their interactions and responses to obstacles. After the game session ends, the score is displayed. Players can then choose to repeat the game or end the session. Conversely, if the player selects Exit from the main menu, the game is immediately stopped and redirected to the Finish screen. In general, the flow of player interaction with the game can be seen in Figure 3. Game Flowchart.

Integration with the gesture classification model is also carried out through real-time communication between GNN-LSTM and Unity Engine. For example, Jump In Place to jump, Jump Left and Jump Right to change lanes, Looking Down to slide, and Still Pose to maintain position. This process ensures that the game can be fully controlled by the player's gestures.

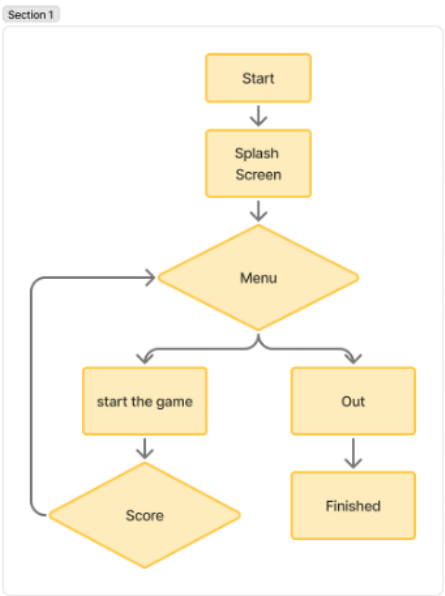

Figure 3. Game Flowchart

IV. RESULT AND DISCUSSION

4.1 Results of GNN-LSTM Model Training

The GNN-LSTM model was trained using 798 training data and 200 test data for 30 epochs with the Adam optimizer (learning rate 0.001) and Cross-Entropy Loss function. Of the total 1000 initial videos, two videos could not be fully extracted by MediaPipe, so only 998 samples were used. Data partitioning was performed using stratified sampling to maintain the proportion of each class.

The training results showed a significant decrease in loss values as the number of epochs increased, with the training and validation loss curves decreasing steadily until convergence at the end of training. The validation accuracy increased from 65% at the beginning of training to 96% at the 30th epoch, indicating that the model was able to learn effectively without significant overfitting.

Evaluation on the test data resulted in an overall accuracy of 96%, with average macro precision, recall, and F1-score values of 0.96, respectively.

Classification Report:

	precision	recall	f1-score	support
jump_in_place	0.95	0.95	0.95	40
jump_left	1.00	0.95	0.97	40
jump_right	0.95	0.95	0.95	40
looking_down	0.95	1.00	0.98	40
still_pose	0.95	0.95	0.95	40
accuracy			0.96	200
macro avg	0.96	0.96	0.96	200
weighted avg	0.96	0.96	0.96	200

Figure 4. GNN-lSTM Model Classification Report

Each gesture class, such as Jump In Place, Jump Left, Jump Right, Looking Down, and Still Pose, has a relatively balanced accuracy level, indicating the model's good generalization ability, as can be seen from the classification report in Figure 4. GNN-lSTM Model Classification Report.

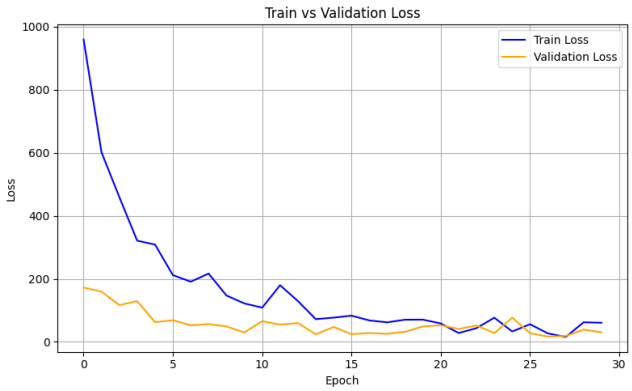

Figure 5. Train vs Validation Loss Model GNN-LSTM

Based on the Train vs Validation Loss graph in Figure 5. Train vs Validation Loss Model GNN-LSTM and the Confusion Matrix in Figure 6. Confusion Matrix Model GNN-LSTM, it can be concluded that the GNN-LSTM model successfully recognizes body movement patterns consistently. The graph structure allows the model to understand the spatial

DOI: 10.34148/teknika.v15i1.1416

relationships between joints, while LSTM captures temporal changes between frames, resulting in accurate and stable predictions for each movement sequence.

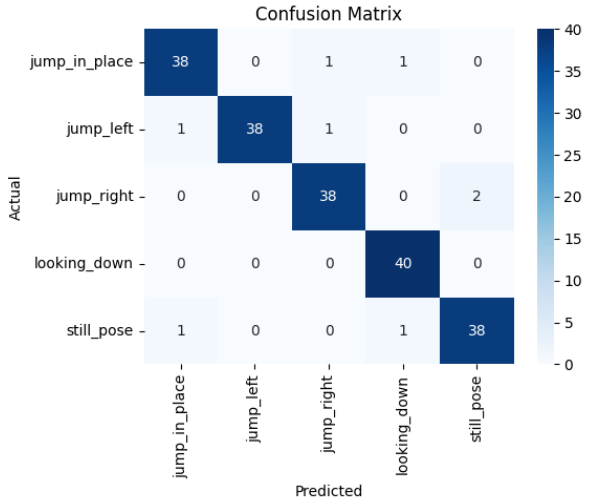

Figure 6. Confusion Matrix Model GNN-LSTM

4.2 Model Performance Comparison

To evaluate the effectiveness of the proposed GNN-LSTM architecture, a comparative experiment was conducted against four baseline models: CNN-LSTM, LSTM, Bi-LSTM, and Temporal Convolutional Network (TCN). All models were trained under identical conditions, including the Adam optimizer with a learning rate of 0.001, Cross-Entropy Loss, 30 training epochs, and the same dataset split ratio, ensuring a fair and objective comparison were also uniform to ensure objective evaluation results.

The experimental results show that CNN-LSTM achieved the highest test accuracy of 97.0%, followed by Bi-LSTM and GNN-LSTM with a 50-frame sequence length, both reaching 96.0%. In contrast, GNN-LSTM with a shorter temporal window of 30 frames obtained an accuracy of 93.0%, while the pure LSTM and TCN models achieved 92.0% and 74.0%, respectively. These results indicate that extending the temporal context from 30 to 50 frames consistently improves classification performance, particularly for the GNN-LSTM architecture.

Although CNN-LSTM slightly outperformed other models in terms of test accuracy, GNN-LSTM exhibited more stable validation behavior, achieving a validation accuracy of up to 97.5% in the 50-sequence configuration. In addition, GNN–LSTM showed more balanced macro-F1 scores and consistent class-wise performance, especially for dynamic gestures, indicating stronger generalization capability. These characteristics make GNN-LSTM more suitable for real-time skeleton-based gesture control, where prediction stability and robustness are more critical than marginal accuracy gains. GNN+LSTM showed excellent validation stability (val acc up to 97.5% in the 50-seq variant) and balanced macro-F1 performance, as shown in Figure 4. GNN-lSTM Model Classification Report.

This indicates strong generalization capabilities in skeleton-based gesture problems. Therefore, the selection of GNN-LSTM for real-time implementation is based on a trade-off between spatial-temporal stability and ease of integration into the skeleton pipeline, not just because of the hight test accuracy.

Table 2 summarizes the evaluation results of each model. It can be seen that the addition of a GNN-LSTM layer with a sequence length of 50 frames at the feature extraction stage successfully improved the model's ability to understand the spatial relationships between body joints, resulting in more accurate predictions compared to pure sequence-based models such as LSTM and Bi-LSTM, and also more stable compared to CNN-LSTM.

Table 2. Comparison of Accuracy Results on Various Model Architectures

Model	Train Acc (Best)	Val Acc (Best)	Test Acc (Final)
LSTM	98.95% (ep 30)	93.72% (ep 29)	92.00%
Bi-LSTM	99.74% (ep 28–29)	96.34% (ep 24)	96.00%
GNN-LSTM	99.50% (ep 28)	97.50% (ep 14,27)	96.00%
TCN	80.39% (ep 29)	78.01% (ep 30)	74.00%
CNN-LSTM	100% (ep 28)	95.33% (ep 25,26,30)	97.00%

4.3 Real-Time Error and Efficiency Analysis

The confusion matrix shown in Figure 6. Confusion Matrix Model GNN-LSTM indicates that the majority of misclassifications occur between the *jump_left* and *jump_right* gesture classes. This confusion arises because both gestures share highly similar spatial configurations, particularly in the lower-body landmarks (hips, knees, and ankles), with the main distinction being the horizontal displacement direction. Variations in camera angle, slight delays in body orientation changes, and differences in individual execution styles further contribute to ambiguity during inference.

Despite this challenge, the overall error rate remains low. The average precision and recall across all gesture classes exceed 0.95, indicating that misclassifications are limited and do not significantly affect overall system performance. Notably, static gestures such as *still_pose* and *looking_down* achieve near-perfect classification results, suggesting that the model is especially robust for poses with minimal temporal ambiguity.

From an efficiency perspective, the GNN-LSTM model with a 50-frame sequence length achieves an average inference time of approximately 0.08–0.12 seconds per sequence on an RTX 4050 GPU, corresponding to 8–12 frames per second (FPS). This performance is sufficient for real-time camera-based game interaction, where gesture commands are issued at discrete intervals rather than continuous frame-by-frame control.

DOI: 10.34148/teknika.v15i1.1416

Further analysis shows that increasing the temporal window from 30 to 50 frames does not result in a significant increase in inference latency. Instead, the longer sequence provides smoother temporal representations, reducing short-term prediction fluctuations and improving classification stability. This trade-off confirms that the 50-frame configuration offers a practical balance between computational efficiency and prediction robustness for real-time gesture-controlled applications.

4.4 Implementation and Integration of Gesture Classification System with Unity Game

The integration between the gesture classification system and the Unity game is implemented using a client–server architecture based on the Transmission Control Protocol (TCP). On the server side, a Python-based application is responsible for capturing real-time video input from a webcam using OpenCV, extracting 33 body keypoints per frame via MediaPipe Pose, applying normalization using the training scaler, and converting the extracted landmarks into a skeleton graph representation compatible with the GNN-LSTM model. Figure 7. Real-Time Gesture Inference Using GNN-LSTM illustrates the real-time gesture inference process, including the predicted gesture labels displayed during live camera input.

Figure 7. Real-Time Gesture Inference Using GNN-LSTM

Once a gesture prediction is generated, the output label (e.g., *jump_left*, *jump_right*, or *looking_down*) is transmitted to the Unity client through a TCP socket connection. On the Unity side, the received gesture label is mapped to predefined character actions, such as jumping, lane switching, or stopping movement. This modular communication design allows seamless integration without requiring modifications to the core game engine logic. An example of the resulting in-game character response to the predicted gestures is shown in Figure 8. Gesture-Based Character Control in Unity Endless Runner Game, demonstrating real-time control of the endless runner gameplay.

Latency analysis of the complete pipeline indicates that the total response time consists of three main components: (1) model inference time (approximately 0.08–0.12 s), (2) TCP communication overhead, which is negligible in a local environment (< 5 ms), and (3) Unity-side action rendering, which occurs within a single game frame. As a result, the end-to-end system response remains perceptually real-time.

Figure 8. Gesture-Based Character Control in Unity Endless Runner Game

Initial deployment tests were conducted on a system equipped with an RTX 4050 GPU using a local server. The system successfully established stable TCP communication, performed real-time pose extraction, and displayed consistent gesture predictions in the OpenCV visualization window. The observed runtime behavior aligns with the offline evaluation results, confirming that the proposed GNN–LSTM model can be reliably deployed for real-time gesture-based game control.

V. CONCLUSION

This study presented a real-time body gesture classification system based on a Graph Neural Network–Long Short-Term Memory (GNN–LSTM) architecture for controlling an endless runner game. The proposed model effectively captures spatial relationships between body joints through graph-based representation and temporal motion patterns through sequential modeling. Experimental results show that the GNN–LSTM model achieves competitive classification performance with stable validation behavior and consistent class-wise results, particularly for dynamic gestures.

Although the CNN–LSTM model achieved slightly higher test accuracy, the GNN–LSTM demonstrated stronger stability and robustness in skeleton-based representations, making it more suitable for real-time gesture-controlled applications. The integration with a Unity-based game via TCP communication further confirmed the feasibility of deploying the proposed system in an interactive environment, enabling responsive character control using only a standard camera without additional input devices.

Despite these promising results, several limitations remain. First, the system currently supports only five basic gesture classes, which limits the complexity of interactions that can be achieved in gameplay. Second, the evaluation focused on technical performance metrics, and no formal user study was conducted to assess usability, comfort, or user experience during prolonged interaction. In addition, experiments were conducted under controlled conditions with a fixed camera setup and lighting environment, which may affect generalization in more diverse real-world scenarios.

DOI: 10.34148/teknika.v15i1.1416

Future work will focus on extending the gesture vocabulary to include more complex and continuous motion patterns, as well as developing personalized gesture-to-action mappings to improve user adaptability. Further improvements may involve exploring transformer-based or attention-based skeleton models to enhance temporal representation. In addition, comprehensive user studies will be conducted to evaluate gameplay responsiveness, usability, and overall user experience, as well as testing the system across different hardware platforms and environmental conditions to improve robustness and scalability.

REFERENCES

[1] M. Wu, "Gesture Recognition Based on Deep Learning: A Review," *EAI Endorsed Transactions on e-Learning*, vol. 10, pp. 1–8, Mar. 2024, doi: 10.4108/eetel.5191.

[2] S. M. Saeed, H. Akbar, T. Nawaz, H. Elahi, and U. S. Khan, "Body-Pose-Guided Action Recognition with Convolutional Long Short-Term Memory (LSTM) in Aerial Videos," *Applied Sciences*, vol. 13, no. 16, p. 9384, Aug. 2023, doi: 10.3390/app13169384.

[3] S. Yan, Y. Xiong, and D. Lin, "Spatial temporal graph convolutional networks for skeleton-based action recognition," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, in AAAI'18/IAAI'18/EAAI'18. AAAI Press, 2018.

[4] H. Lee *et al.*, "Stretchable array electromyography sensor with graph neural network for static and dynamic gestures recognition system," *npj Flexible Electronics*, vol. 7, no. 1, p. 20, Apr. 2023, doi: 10.1038/s41528-023-00246-3.

[5] M. Nan, M. Trăscău, and A.-M. Florea, "Spatio-temporal neural network with handcrafted features for skeleton-based action recognition," *Neural Comput. Appl.*, vol. 36, no. 16, pp. 9221–9243, Jun. 2024, doi: 10.1007/s00521-024-09559-4.

[6] G. Abbate, A. Giusti, V. Schmuck, O. Celiktutan, and A. Paolillo, "Self-supervised prediction of the intention to interact with a service robot," *Rob. Auton. Syst.*, vol. 171, no. October 2023, p. 104568, 2024, doi: 10.1016/j.robot.2023.104568.

[7] H. Xia and X. Gao, "Multi-scale Mixed Dense Graph Convolution Network for Skeleton-based Action Recognition," vol. 4, pp. 1–10, 2020, doi: 10.1109/ACCESS.2020.3049029.

[8] Y. Wang, M. Yu, N. Li, E. Gao, G. Wang, and L. Zhu, "Difference-attention graph convolutional network for skeleton-based gesture recognition," vol. 123, no. 70, 2025.

[9] J. Sanchez, C. Neff, and H. Tabkhi, "Real-World Graph Convolution Networks (RW-GCNs) for Action Recognition in Smart Video Surveillance," in *2021 IEEE/ACM Symposium on Edge Computing (SEC)*, 2021, pp. 121–134. doi: 10.1145/3453142.3491293.

[10] X. Han, Y. Cui, X. Chen, Y. Lu, and W. Hu, "Spatio-Temporal Dynamic Attention Graph Convolutional Network Based on Skeleton Gesture Recognition," *Electronics (Basel).*, vol. 13, no. 18, p. 3733, Sep. 2024, doi: 10.3390/electronics13183733.

[11] D. D. N. Cahyo and S. Liyan, "Perancangan Desain Antarmuka Pengguna Game Endless Runner Matematika Dasar Berbasis Android Menggunakan Metode System Usability Scale," Abstrak, vol. 6, pp. 54–64, 2024.

[12] M. Pertiwi and Riwinoto, "Effect Of Playability On User Experience In Game 'Joko Run,'" *Journal of Applied Multimedia and Networking*, vol. 3, no. 1, pp. 1–14, 2018.

[13] T. Yu, I. Journal, T. Y. Jia, A. Hussain, Y. Yusof, and M. Motion, "Mobile Game using Hand Gesture," *International Journal of Emerging Trends in Engineering Research*, vol. 8, no. 10, pp. 6842–6848, 2020, doi: 10.30534/ijeter/2020/348102020.

[14] M. Schmidt and C. Pöpel, "HETZI-jump and run: Development and evaluation of a gesture controlled game," *CEUR Workshop Proc.*, vol. 1734, pp. 91–99, 2016.

[15] M. Nan and A. M. Florea, "Fast Temporal Graph Convolutional Model for Skeleton-Based Action Recognition," *Sensors*, vol. 22, no. 19, p. 7117, Sep. 2022, doi: 10.3390/s22197117.

[16] L. Kristiana and D. Miyanto, "Penambahan Parameter PM2 . 5 dalam Prediksi Kualitas Udara : Long Short Term Memory," vol. 8, no. 2, pp. 188–202, 2023.

[17] J.-W. Kim, J.-Y. Choi, E.-J. Ha, and J.-H. Choi, "Human Pose Estimation Using MediaPipe Pose and Optimization Method Based on a Humanoid Model," *Applied Sciences*, vol. 13, no. 4, p. 2700, Feb. 2023, doi: 10.3390/app13042700.

[18] S. Farhadpour, T. A. Warner, and A. E. Maxwell, "Selecting and Interpreting Multiclass Loss and Accuracy Assessment Metrics for Classifications with Class Imbalance: Guidance and Best Practices," *Remote Sens. (Basel).*, vol. 16, no. 3, 2024, doi: 10.3390/rs16030533.