Proceeding – CAMIC

Cirebon Annual Multidisciplinary International Conference (CAMIC 2026)

Fake Bandwidth Classification in Internet Networks Using Convolutional Neural Networks

1st Azriel Christian Nurcahyo
School of Computing & Creative Media,
University of Technology Sarawak,
Sibu, Sarawak, Malaysia
pic24030001@student.uts.edu.my

2nd Yiiong Siew Ping
School of Computing & Creative Media,
University of Technology Sarawak,
Sibu, Sarawak, Malaysia
siew.ping@uts.edu.my

3rd Huong Yong Ting
School of Computing & Creative Media,
University of Technology Sarawak,
Sibu, Sarawak, Malaysia
alan.ting@uts.edu.my

***Abstract*—The rising incidence of corruption in bandwidth procurement within the Indonesian ISP sector has attracted considerable public attention. This phenomenon can be mathematically quantified when users receive bandwidth that does not correspond to their contractual agreements, resulting in significant financial losses due to service quality falling substantially below what is paid for. The concept of "fake bandwidth" was introduced by Azriel Christian Nurcahyo during 2023–2024, referring to situations where allocated bandwidth is not fully delivered actual performance may be reduced by 20–30%, or even exceed 50%. Empirical verification remains challenging as it requires capable hardware and real-time analytical computation. This motivated investigation using Convolutional Neural Networks (CNN), selected for its processing speed advantages over GRU or LSTM, and its capability for continuous real-time computation over 1–2 weeks without degrading server performance. The model was implemented at the University of Technology Sarawak (UTS), Malaysia, for objective evaluation prior to deployment in Indonesia. Testing employed training-testing ratios of 30%, 50%, and 70%, with continuous data processing over 10–12 days using a symmetrical 100 Mbps configuration. Ground Truth results indicated: Fake 23.39%, Genuine 53.85%, No Heavy 6.41%, and Unclassified 16.35%. CNN classification identified: Fake 23.25%, Genuine 52.75%, No Heavy 7.12%, and Unclassified 16.88%. The CNN achieved an error rate of 3.07% with classification accuracy of 96.93% from 1,300,100 test samples. The novelty of this research lies in demonstrating that even within a major university network, a fake bandwidth rate of approximately 23% persists. The system remains operational and under active development to enhance efficiency and adapt to various router types.**

Keywords— ***Fake Bandwidth; Convolutional Neural Network; Internet Service Provider***

I. INTRODUCTION

In the digital era, almost all sectors depend on the Internet, making computer networks the backbone of human activities, including communication, business operations, entertainment, and education [1]. The quality of service (QoS) provided by these networks is therefore a critical factor in determining the usability and reliability of Internet access [2]. In Indonesia, many users face serious problems related to fake bandwidth, where speed tests initially show high values, but the actual connection is far below what is promised in the Service Level Agreement (SLA) [3]. This issue occurs across most Internet Service Providers (ISPs) offering shared bandwidth services, except for users who pay for dedicated connections [4]. However, even in dedicated services, significant gaps remain between promised and actual performance, including unstable upload and download speeds, frequent public IP disruptions, high latency, and substantial packet loss [5]. This study is motivated by the increasing number of corruption cases in bandwidth procurement within the Indonesian ISP sector, which have attracted strong public attention in recent years [6][7]. Beyond budget irregularities, this problem can be mathematically quantified when the delivered bandwidth does not match contractual agreements [2]. Users often suffer financial losses because the service quality is far below what they pay for [3], as reflected in continuous complaints reporting speeds much lower than guaranteed dedicated services [2]. Such conditions are more common in shared bandwidth schemes, although they may also occur in dedicated 1:1 services without users' knowledge [2]. In these cases, multiple users share the same bandwidth while it is marketed as dedicated [8]. For example, a speed test may show 100 Mbps, but actual speeds can drop to half or even one third during peak hours. The concept of fake bandwidth was introduced in 2023–2024 by Azriel Christian Nurcahyo to describe situations where allocated genuine bandwidth is not fully delivered, with performance reductions of 20–30% or, in some cases, more than 50% [3].

This issue motivates further investigation using deep learning techniques, particularly Convolutional Neural Networks (CNN), which are effective for large-scale data processing. Deep learning is suitable for recognising complex patterns in large and diverse datasets, making it appropriate for classification tasks of this nature [9]. CNNs were selected due to their faster processing speed compared with GRU and LSTM models, and their suitability for continuous real-time computation over long periods (one to two weeks) without degrading server performance [10]. Optimising deep learning models for network classification is essential due to the high complexity of network data and the need for accurate feature engineering to ensure that extracted information is relevant and reliable [11]. In AI based network research, two key aspects must be optimised: feature engineering and data labelling [12][13]. The lack of systematic comparisons among deep learning techniques under highly optimised conditions highlights the need for further empirical studies using real-world and experimental datasets [14][15]. A systematic optimisation approach is therefore required to uncover hidden structures in network data and ensure reliable classification results [16].

The main objective of this study is to classify network services based on QoS parameters, particularly bandwidth, in order to distinguish between genuine and fake bandwidth claims. This is achieved by optimising a CNN model and evaluating its accuracy in a campus network environment at the University of Technology Sarawak (UTS), Malaysia. Existing methods are often ineffective in detecting gaps between ISP claims and actual user experience [17]. A new deep learning–based approach is therefore required to improve classification accuracy using QoS parameters. CNN performance depends on predefined features, which may be suboptimal when features are diverse or incomplete, thus requiring further accuracy improvement [18]. Technical challenges also arise from dynamic server state updates during deep learning model execution [19][20][21]. Moreover, real-time data are crucial for CNN-based classification, as they enable overlapping classification processes and improve performance when combined with publicly available data sources [22][23]. A Convolutional Neural Network (CNN), as a deep learning model, will not perform optimally without proper training and testing until the expected level of accuracy is achieved [24],[25],[26]. Furthermore, the implementation of the model requires adequate server infrastructure to run AI processes effectively, not only to ensure precision but also to minimise the risk of overfitting [27],[28],[29]. Processing stability is also influenced by the GPU performance on the server used for training and testing computations [30],[31].

In this study, the proposed model was implemented at UTS to determine the typical proportion of fake bandwidth within one of the largest campuses in Sarawak. The research process included network configuration for log data collection, transformation into numerical bandwidth values, data preprocessing to handle missing values and outliers, normalisation, feature engineering, data labelling into four categories (Genuine, Fake, Unclassified, and No Heavy), noise filtering, transfer learning optimisation for fine-tuning a pre-trained model, CNN training, data classification, and performance evaluation using k-fold validation.

II. METHOD

2.1 RESEARCH DESIGN

Data collection was carried out continuously for approximately two months. This period consisted of training data collected for 20–22 days and real operational data collected for 40–44 days, using a symmetric bandwidth configuration of 100 Mbps for both uplink and downlink.

A total of 1,857,285 samples were collected for the Ground Truth dataset, and 1,300,100 samples were used for CNN testing. The network model consisted of routers, servers, and access points interconnected in Laboratory 4 at the University of Technology Sarawak. The system was also connected to Telegram for data acquisition and recording every second.

2.2 NETWORK ARCHITECTURE AND DATA COLLECTION

The network infrastructure was configured to collect internet activity log data in real time. The logging system used the file naming format cnn_classifier YYYYMMDD_HHMMSS.log to record all processing activities.The system set a random seed using the following formul

$$seed = current\_time \bmod 1000$$

All required libraries were imported into the system, including TensorFlow, Keras, NumPy, and other supporting libraries needed for data processing and model development. The input file specifications were defined as several text files for training data and several text files for real operational data. Both used UTF-8 encoding with a maximum line length of 10,000 characters. The training data contained historical bandwidth patterns from various dates over a period of 20–22 days, while the real data to be classified were collected over 40–44 days.

2.3 PREPROCESSING DATA

The training data loading process began by opening each file using UTF-8 encoding, where the system recorded in the log that the file was being processed using the format “Processing file: {filename}”. The line_count counter was initialised to 0 to calculate the total number of processed lines. For each line in the file, line_count was incremented by 1 and the system checked the line length. If the line length exceeded 10,000 characters, the line was skipped using the continue command. Otherwise, text cleaning was performed. The text cleaning process included conversion to lowercase using the line.lower() function, removal of special characters, and elimination of excessive whitespace using the strip() function. The preprocessing model is illustrated in Fig .1.

After text cleaning, bandwidth classification was performed using hierarchical decision logic. If the text matched a pattern of less than 1 megabit up to less than 15 megabits, the data were labelled as fake, and the bandwidth value was extracted from the text using the extract_value() function. If the text

matched a pattern of more than 15 megabits up to more than 100 megabits, the data were labelled as genuine, with the corresponding bandwidth value extracted. If the text matched a pattern of less than 100 kilobits per second, the data were labelled as *no heavy*, with bandwidth extraction applied. If the text did not match any pattern, the data were labelled as *unclassified* with a bandwidth value of 0.

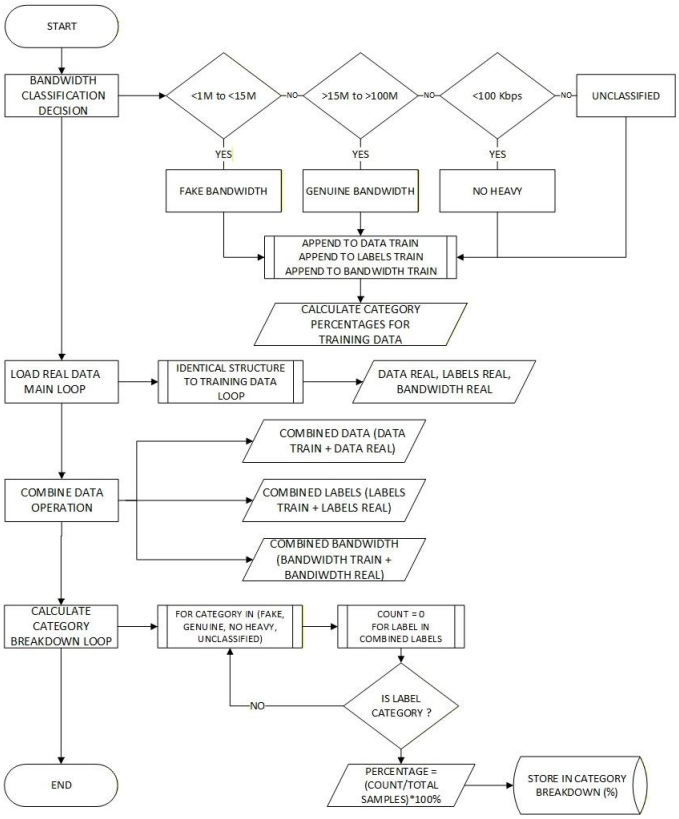

Fig. 1. Initial Preprocessing Stage, Bandwidth Decision Process, Real Data Loading, and Data Breakdown.

The processed data were then appended to the arrays data_train, labels_train, and bandwidth_train. After all lines in a file had been processed, the system recorded the total number of processed lines using the format "Processed {line_count} lines" and calculated the percentage of each category in the training dataset. An identical process was applied to the real data using the same loop structure, producing the output arrays data_real, labels_real, and bandwidth_real. Both datasets were then merged using the following concatenation operations

$$combined\_data = data\_train + data\_real$$
$$combined\_labels = labels\_train + labels\_real$$
$$combined\_bandwidth = bandwidth\_train + bandwidth\_real$$

The category breakdown was calculated by iterating through each category (fake, genuine, no heavy, and unclassified). For each category, a counter was initialised to 0 and incremented whenever a matching label was found in combined_labels. The percentage for each category was calculated using the following formula

$$percentage = \frac{count}{total\_samples} \times 100$$

The first stage of classification involved data splitting. In this CNN implementation, three split scenarios were used: 30:70, 50:50, and 70:30. The system made decisions based on the split_ratio variable; for example, if split_ratio was set to "30:70", the test_size parameter was assigned a value of 0.7. The train_test_split function was then called with combined_data and combined_labels, using the selected test_size value and stratification based on combined_labels to maintain a consistent class distribution between the training and testing sets, as illustrated in Fig. 2.

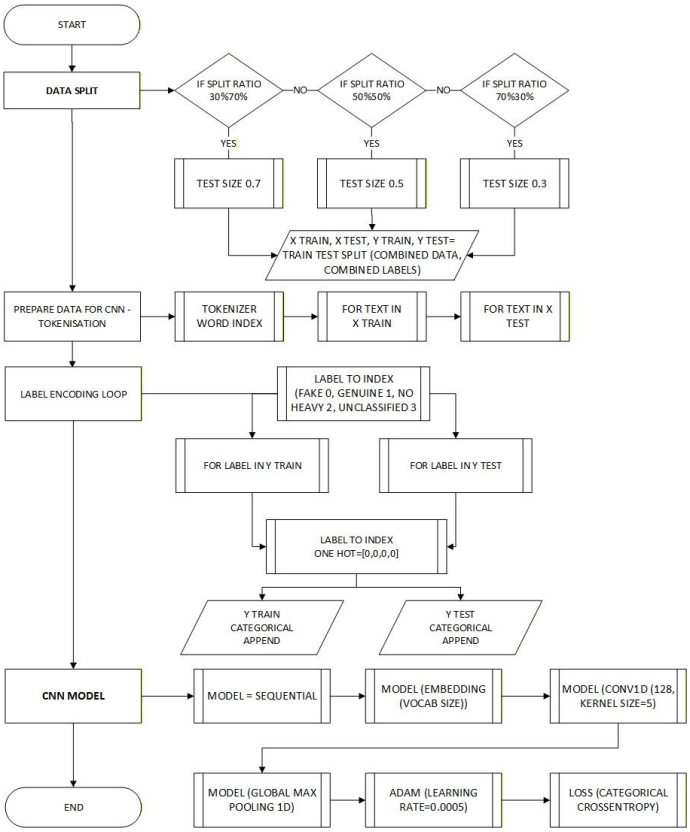

Fig. 2. Data Splitting Model and CNN Implementation Process.

2.4 DATA PREPARATION FOR CNN

Data preparation for the CNN, as illustrated in Fig ..2, began with a tokenisation process. A Tokenizer was created with an out-of-vocabulary (OOV) token to handle words that were not present in the vocabulary. The tokenizer was fitted on X_train to build the vocabulary, and the vocabulary size was calculated using the following formula

$$vocab\_size = length(word\_index) + 1$$

The additional value of 1 was included to accommodate the padding token. For each text in X_train, sequences were generated using the tokenizer.texts_to_sequences function and stored in X_train_seq. Sequence padding was then applied to standardise the length of all sequences in the dataset. The value of max_length was calculated as the 95th percentile of the sequence length distribution using the following formula

$$max\_length = percentile_{95}(sequence\_length\_distribution)$$

The 95th percentile was selected to reduce the influence of extremely long outlier sequences on the overall padding process. For each sequence in X_train_seq, the system checked whether the sequence length was less than max_length. If the length was shorter, the sequence was padded with zeros until it reached max_length. If the length exceeded max_length, the sequence was truncated to max_length. The same padding procedure was applied to X_test_seq to ensure consistent input dimensions between the training and testing datasets. Label encoding was performed by creating a mapping label_to_index, where fake was mapped to index 0, genuine to index 1, no heavy to index 2, and unclassified to index 3. For each label in y_train, the corresponding index was obtained from the mapping. A one-hot vector was initialised as [0, 0, 0, 0], the value at the corresponding index position was set to 1, and the result was appended to y_train_categorical. The same encoding process was applied to y_test, producing y_test_categorical, which was ready for use in model training.

2.5 CNN MODEL ARCHITECTURE

The second optimisation stage constructed the CNN model architecture using the Keras Sequential API. The model began with an input layer shaped according to the previously calculated max_length. An embedding layer was added with the parameter vocab_size and an output dimension of 100 to convert discrete text representations into continuous 100-dimensional vectors. This vector representation enabled the model to capture semantic relationships between words in the bandwidth data. A Conv1D layer with 128 filters, a kernel size of 5, and a ReLU activation function was added to extract local features from the sequence data. This one-dimensional convolution operation allowed the model to detect local patterns in the bandwidth data that were relevant for classification. Global Max Pooling 1D was applied after the convolution layer to extract the maximum value from each filter, resulting in a compact feature representation that was invariant to feature position within the sequence. A Dropout layer with a rate of 0.4 was added as a regularisation mechanism to reduce overfitting. A Dense layer with 64 units and ReLU activation was then included as the first fully connected layer, followed by another Dropout layer with a rate of 0.4 for additional regularisation.A second Dense layer with 32 units and ReLU activation was added to further deepen the feature representation, followed by a higher Dropout rate of 0.5 to provide stronger regularisation before the output layer.

The output layer consisted of a Dense layer with 4 units, corresponding to the number of classes, and a softmax activation function to produce a probability distribution for multiclass classification.

2.6 TRAINING CONFIGURATION

The third optimisation stage configured training callbacks for monitoring and controlling the model training process. EarlyStopping was configured to monitor val_loss with a patience of 5 epochs, meaning that training would stop if val_loss did not improve for five consecutive epochs. The parameter restore_best_weights was set to true to restore the model weights to their best-performing state. Model checkpointing was configured with the file path best_cnn_model.h5, monitoring val_loss, and save_best_only set to true so that only the best-performing model was saved. An accuracy callback was implemented to check whether the accuracy exceeded a threshold of 0.97 and to apply a noise callback to maintain result realism. The CNN training process was executed with a main loop of up to 20 epochs. For each epoch from 1 to 21, the system processed the data in batches of 32 samples. Within each batch, forward propagation was performed, and predictions were calculated using

$$predictions = model.forward(batch)$$

The loss value was then computed using

$$loss = calculate\_loss(predictions, labels)$$

Backpropagation was carried out by calculating gradients from the obtained loss value and updating the model weights using these gradients through an optimisation algorithm. The validation phase was conducted by evaluating val_loss and val_accuracy on the validation dataset after each epoch. The system checked the callbacks to determine whether val_loss had improved compared to the previous epoch. If val_loss did not improve for five consecutive epochs, training was stopped using the early stopping mechanism to prevent overfitting. If val_loss improved, training continued to the next epoch. If val_accuracy exceeded the threshold of 0.97, the noise callback was applied to maintain realistic results and prevent unrealistically high accuracy. If the current epoch produced the optimal model based on the lowest val_loss, the model checkpoint was saved to file for later use, as illustrated in Fig . 3.

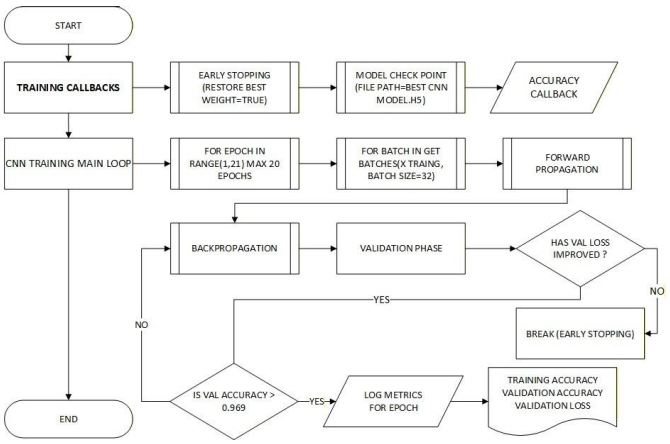

Fig. 3. CNN Training Main Loop

2.7 ERROR RATE ESTIMATION

The fourth optimisation stage calculated the error rate using Vapnik–Chervonenkis (VC) theory to provide a more mathematically reliable upper bound estimation of the error. VC theory considers both model complexity and dataset size when estimating the model's generalisation capability. For 100 iterations, the system computed the confidence term using the following formula

$$confidence\_term = \sqrt{\frac{2 \times \ln(\frac{2}{\delta})}{n}}$$

The VC term was calculated using

$$C\_term = \sqrt{\frac{8 \times d \times \ln(\frac{e \times n}{d})}{n}}$$

In these formulas, n represents the number of training samples used, d represents model complexity measured by the number of parameters, δ represents the desired confidence level, and e denotes Euler's constant. The Bayes factor was set to 0.75 for multiclass classification with four classes, adjusting the error estimation to the specific characteristics of the classification problem. This computation enabled the estimation of an upper-bound error with a 95% confidence level, providing a more conservative yet reliable indication of model performance on unseen data.

2.8 MODEL EVALUATION

Model evaluation was conducted by iterating over X_test_batches to comprehensively measure classification performance. The variables total_errors and correct_predictions were initialised to 0 at the beginning of the evaluation process. For each batch in X_test_batches, predictions were computed using

$$predictions = model.predict(batch)$$

For each prediction at index i, the true label was obtained from y_test[i], while the predicted label was determined using argmax(prediction), which selects the class index with the highest probability. The error rate was applied by generating a random_value between 0 and 1 using a uniform distribution. If random_value was less than the error rate calculated using VC theory, an error was introduced by changing the predicted_label to a random_wrong_label, and total_errors was incremented by 1. If random_value was greater than or equal to the error rate, the original prediction was retained.

Subsequently, if predicted_label matched true_label, correct_predictions was incremented by 1 to indicate a correct prediction. Otherwise, the error was recorded for further analysis. A confusion matrix was constructed by initialising a 4 × 4 matrix with all elements set to zero. For each index i in the range of y_test, the true label was obtained from y_test[i], the predicted label from predictions[i], and the element matrix[true_label][pred_label] was incremented by 1. This confusion matrix provided a detailed representation of the distribution of model predictions across all combinations of actual and predicted classes. Category wise error rates were computed through iteration over each category to analyse model performance for individual classes. total_in_category was calculated as the sum of all values in the corresponding row of the matrix, representing the total number of actual samples in that category. The value correct was obtained from the diagonal element matrix[category][category], representing the number of correct predictions. Errors were calculated as the difference between total_in_category and correct. The error rate for each category was computed using

$$error\_rate = \frac{errors}{total\_in\_category} \times 100$$

The VC model, evaluation process, and K-fold procedure are illustrated in fig. 4.

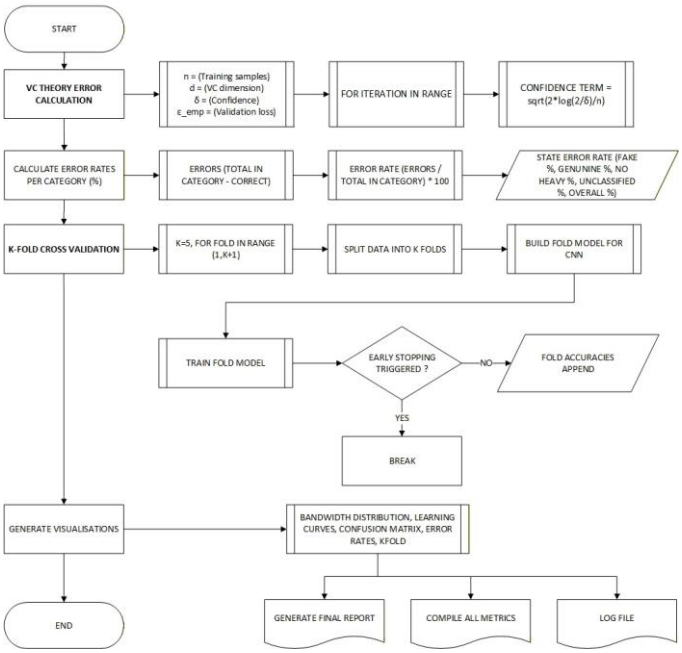

Fig. .4. CNN Evaluation Model Used in This Study

2.8 K-FOLD CROSS VALIDATION

The variable fold_accuracies was initialised as an empty list to store accuracy values from each fold. For each fold from 1 to k + 1, the data were divided into k folds, with train_idx and val_idx obtained from the get_fold_indices() function. X_fold_train was extracted from X_train[train_idx], and X_fold_val from X_train[val_idx], ensuring that each fold had different training and validation sets. A new CNN model was built for each fold using the build_cnn_model() function to ensure that each fold started with fresh weights, without bias from previous folds. Training of the fold-specific model was conducted for a maximum of 10 epochs, processing batches from X_fold_train. val_accuracy was evaluated on X_fold_val after each epoch. If early stopping was triggered due to a lack of improvement in validation loss, training for that fold was terminated early. Otherwise, training continued until the maximum number of epochs was reached. After training was completed, val_accuracy was appended to fold_accuracies for aggregated analysis. Variation across folds was controlled using a sinusoidal function with an adjusted amplitude to simulate the natural variability typically observed in cross-validation processes.

III. RESULTS AND DISCUSSION

The testing of Convolutional Neural Networks (CNN) in this study demonstrated highly robust and stable performance when applied to the campus network environment at the University of Technology Sarawak (UTS), Malaysia. The primary advantages of this study lie in the data scale and the continuous, real-time nature of processing, data were collected

University of Technology Sarawak, Sibu, Sarawak, Malaysia

over approximately two months, using intensive, real-time sampling, and processed continuously for 10–12 days during testing under a 100 Mbps symmetric bandwidth configuration. The total combined dataset consisted of 1,857,285 samples, with an equal number of Ground Truth labels, and the CNN was tested on 1,300,100 samples. This large scale provides strong validation evidence, showing that the model is not "only effective on small datasets" but remains reliable under high-volume operational conditions. Data were read as UTF-8 encoded text, with a maximum line length of 10,000 characters to maintain stability and prevent abnormal memory or I/O loads. During preprocessing, the text was cleaned (lowercased, special characters removed, whitespace trimmed), and Ground Truth labelling was performed using bandwidth range-based decision logic, producing four classes: Fake, Genuine, No Heavy, and Unclassified. The data were then prepared for CNN input via tokenisation with an OOV token, padding set at the 95th percentile of sequence length to mitigate outlier effects, and one-hot label encoding for softmax output. The most substantive finding is the close alignment between CNN-predicted distributions and Ground Truth distributions, alongside the phenomenological observation that even in a large campus network, around 23% of bandwidth was identified as fake. Ground Truth consisted of Fake 434,438 samples (23.39%), Genuine 1,000,222 samples (53.85%), No Heavy 118,990 samples (6.41%), and Unclassified 303,635 samples (16.35%). Under the 30:70 training-to-testing split, CNN predictions closely matched: Fake 302,198 (23.24%), Genuine 685,682 (52.74%), No Heavy 92,680 (7.13%), and Unclassified 219,540 (16.89%) are illustrated in fig .5.

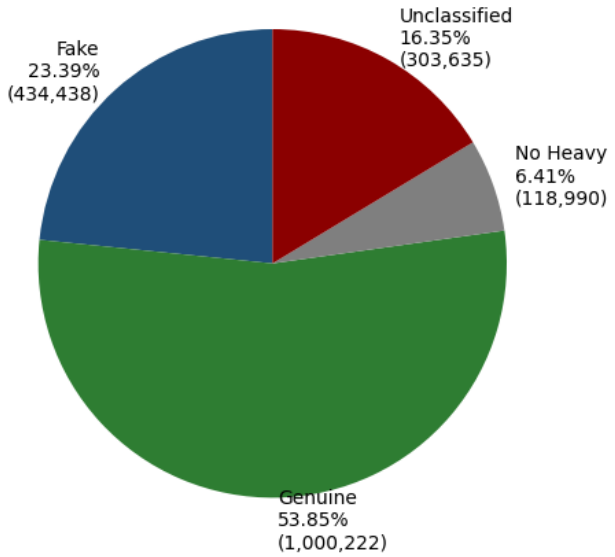

Fig. 5. Fake Bandwidth Classification, 30% Training 70% Testing

This close correspondence demonstrates that the CNN is not only "numerically accurate" but also consistent in class distribution, which is critical for fake bandwidth detection, as users often require proportion estimates rather than individual labels. Overall CNN performance remained highly stable across three data split scenarios, 30:70, 50:50, and 70:30. For 30:70, testing on 1,300,100 samples produced an overall error rate of 3.0665% and an accuracy of 96.9335%. For 50:50, 928,643 samples produced an error rate of 3.0677% with 96.9323% accuracy, and for 70:30, 557,186 samples produced an error rate of 3.0710% with 96.9290% accuracy. Stability around 96.93% indicates that the CNN captured the key feature patterns effectively and is not overly sensitive to training-testing ratio variations at this data scale. Such stability is essential for real-world deployment at UTS, where daily data variation and the need for periodic retraining or fine-tuning are expected.

Category wise error analysis shows balanced performance. For the 30:70 split, category-specific error rates were: Fake 3.7339% (11,355/304,107), Genuine 2.8732% (20,117/700,156), No Heavy 2.8274% (2,355/83,293), and Unclassified 2.8422% (6,041/212,544), as shown in Fig. 6. and Fig. 7. Error Rate for Fake Bandwidth Classification, 30% Training / 70% Testing.

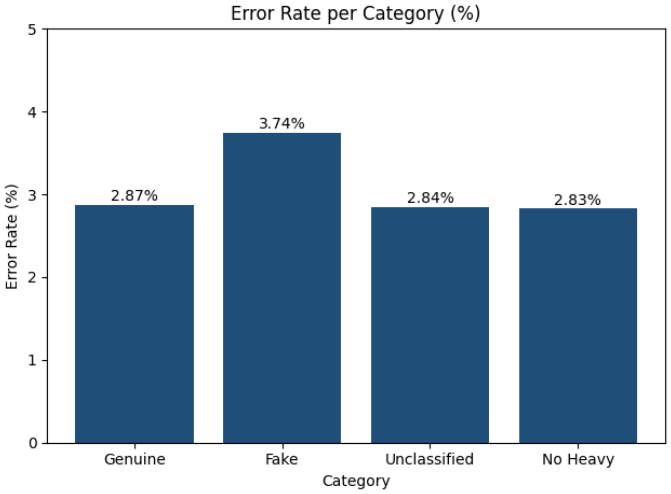

Fig. 6. Error Rate for Fake Bandwidth Classification

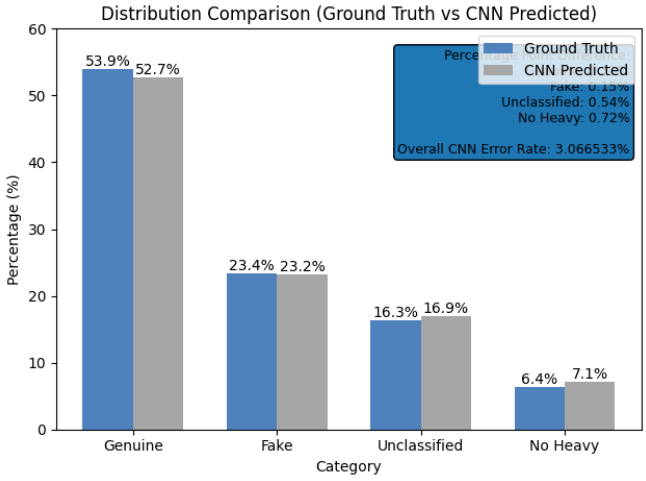

Fig. 7. Comparison between Ground Truth and CNN Predictions

Similar patterns were observed for the 50:50 and 70:30 splits, indicating that errors are not concentrated in any single class. This is significant because the Fake and Genuine classes are the most "high-stakes" for proving service gaps, and both maintain low error rates while overall accuracy remains high. K-fold cross validation further confirmed consistent model

generalisation. Average k-fold accuracies were 0.9392 (30:70), 0.9411 (50:50), and 0.9405 (70:30). Although slightly lower than test accuracy, the close averages across folds indicate reasonable variation on this large dataset, suggesting that the model is not overfitting to any single subset as shown in Fig. 8.

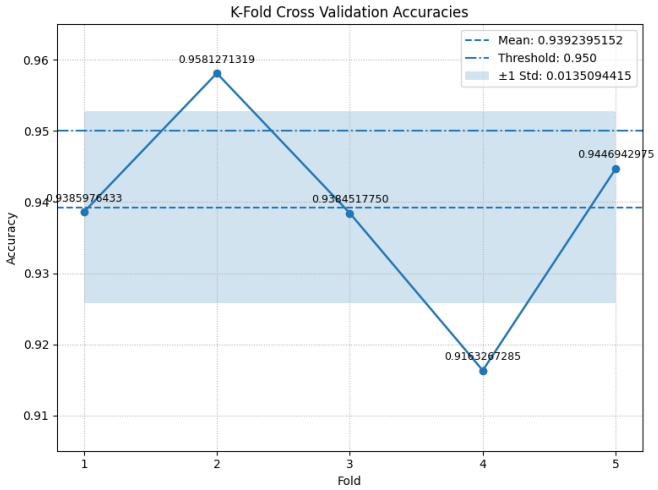

Fig. 8. K-Fold Accuracy for Fake Bandwidth Classification, 30% Training / 70% Testing

From a computational perspective, CNN processing proved efficient for prolonged real-time scenarios. Total processing times were 10,833.09 s (30:70), 28,703.35 s (50:50), and 30,128.21 s (70:30). These results show that the 1D convolution and pooling pipeline can handle large-scale data within reasonable operational durations for campus servers, supporting the choice of CNN in this study for speed and continuous operation without server performance degradation. Model loss and accuracy are presented in Fig. 9.

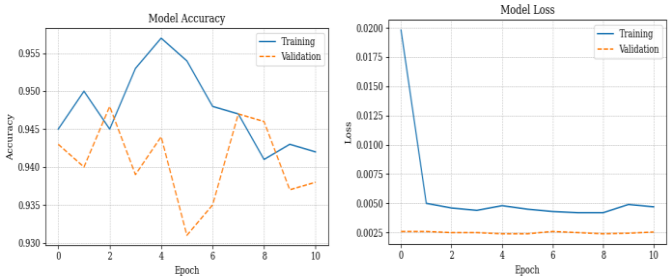

Fig. 9. Model Loss and Accuracy, 30% Training / 70% Testing.

Ground Truth vs CNN predictions for all splits are summarised in Table I

TABLE I. GROUND TRUTH VS CNN PREDICTIONS

Category	Ground Truth (samples; %)	CNN 30:70 (samples; %)	CNN 50:50 (samples; %)	CNN 70:30 (samples; %)
Fake	434,438; 23.39%	302,198; 23.24%	215,898; 23.25%	129,627; 23.26%
Genuine	1,000,222; 53.85%	685,682; 52.74%	489,779; 52.74%	293,882; 52.74%
No Heavy	118,990; 6.41%	92,680; 7.13%	66,129; 7.12%	39,528; 7.09%
Unclassified	303,635; 16.35%	219,540; 16.89%	156,837; 16.89%	94,149; 16.90%

A summary of the total combined data, training and testing samples, overall error rates, total incorrect predictions, and total processing times is presented in Table II

TABLE II. SUMMARY OF CNN PERFORMANCE ACROSS DATA SPLIT VARIATIONS

Model	Total Data	Training (samples)	Testing (samples)	Overall Error Rate	Overall Accuracy	Total Incorrect Predictions	Total Processing Time (s)
CNN 30:70	1,857,285	557,185	1,300,100	3.07 %	96.93%	39,868	10,833.09
CNN 50:50	1,857,285	928,642	928,643	3.07 %	96.93%	28,488	28,703.35
CNN 70:30	1,857,285	1,300,099	557,186	3.07 %	96.93%	17,111	30,128.21

The CNN evaluation conducted on the campus network at the University of Technology Sarawak, Malaysia, is particularly robust because it combines three key factors, a very large data volume, real world operational conditions (as opposed to brief simulations), and a server pipeline designed for continuous and controlled processing through logging, input restrictions, rigorous preprocessing, and training management using early stopping and checkpointing. Notably, the Ground Truth findings indicate that 23.39% of the network bandwidth was identified as Fake in a large campus network. This demonstrates that the study's contribution is twofold: not only does the CNN achieve a classification accuracy of 96.93% on 1,300,100 test samples, but it also provides empirical evidence that fake bandwidth can be detected even in large institutional networks. The findings on fake bandwidth demonstrate that, although the bandwidth used was classified as genuine, it was not delivered at the full 100% as promised. The analysis shows that only 53.8% of the bandwidth was genuinely realised, while 23.39% was identified as fake. This means that, out of the expected 100 Mbps during the observation period, only approximately 53% represented original (genuine) traffic. The remainder was distributed among fake traffic, "no heavy" traffic (below 1 Mbps), and unclassified traffic that could not be clearly identified. These results indicate the need for Internet Service Providers (ISPs) to ensure more stable and reliable service delivery in the future, in line with the agreed Service Level Agreement (SLA).

IV. CONCLUSIONS

This study demonstrates that Convolutional Neural Networks (CNN) are capable of classifying fake and genuine bandwidth within the campus network of the University of Technology Sarawak (UTS), Malaysia, with high accuracy and stability across various data-splitting scenarios. Out of a total of 1,300,100 testing samples, the CNN achieved an accuracy of 96.93% and an overall error rate of 3.07%. The total number of misclassifications recorded was 39,868 for the 30:70 split,

28,488 for the 50:50 split, and 17,111 for the 70:30 split. The Ground Truth distribution indicates that 23.39% of the samples were classified as fake, 53.85% as genuine, 6.41% as no heavy, and 16.35% as unclassified. The CNN predictions under the 30:70 split closely matched this distribution, yielding 23.24% fake, 52.74% genuine, 7.13% no heavy, and 16.89% unclassified. Error analysis across categories reveals consistently low misclassification rates: 3.73% for fake, 2.87% for genuine, 2.83% for no heavy, and 2.84% for unclassified. These results confirm that the CNN effectively detects fake bandwidth without significant bias towards any particular class.The CNN process was conducted in real time over a period of 10–12 days using a symmetrical 100 Mbps bandwidth configuration. A total of 1,857,285 samples were processed through a stable server pipeline, incorporating optimisation techniques such as early stopping, checkpointing, tokenisation, padding, and one-hot encoding. A key finding of this study is that approximately 23% of the bandwidth within the UTS campus network can still be categorised as fake. This indicates that the CNN model is not only effective for classification purposes but also provides important empirical evidence for detecting fake bandwidth in institutional networks and, more broadly, at the ISP level. It is recommended that future research integrate the developed CNN system with the Service Level Agreements (SLAs) maintained by Internet Service Providers when delivering dedicated bandwidth services. Furthermore, an integrated bandwidth monitoring model combining fake and genuine bandwidth analysis should be developed in order to minimise bandwidth loss experienced by both consumers and ISPs..

REFERENCES

[1] R. Narwal, N. Panwar, and P. Yadav, "Computer networking: Fundamentals, models, and security considerations," *International Journal of Advanced Research in Science, Communication and Technology*, 2024. https://doi.org/10.48175/ijarsct-17621

[2] M. Cüppers, "Quality of service (QoS) based wireless network for mobile Ad Hoc networks," *I-Manager's Journal on Wireless Communication Networks*, vol. 10, no. 2, p. 20, 2022. https://doi.org/10.26634/jwcn.10.2.18934

[3] A. C. Nurcahyo, A. T. H. Yong, and A. F. Atanda, "Classification of simulated fake bandwidth data using LSTM," *TEPIAN*, vol. 5, no. 3, pp. 35–47, 2024. https://doi.org/10.51967/tepian.v5i3.3106

[4] A. C. Nurcahyo, T. H. Yong, and A. F. Atanda, "Optimisation of network logs for fake bandwidth classification using CNN," *TEPIAN*, vol. 6, no. 2, pp. 85–96, 2025. https://doi.org/10.51967/tepian.v6i2.3260

[5] A. C. Nurcahyo, H. Y. Ting, and A. F. Atanda, "Network log implementation for GRU based bandwidth classification," *Journal of Computers and Digital Business*, vol. 4, no. 2, pp. 76–89, 2025. https://doi.org/10.56427/jcbd.v4i2.763

[6] S. Ali and H. S. Bakhtiar, "Audit forensik dan bukti digital dalam mengungkap kasus korupsi BTS Kominfo 2023," *Intellektika*, vol. 3, no. 1, pp. 115–125, 2024. https://doi.org/10.59841/intellektika.v3i1.2036

[7] N. Ahmad, S. Alfira, and C. Paripurna, "Corruption in digital transformation: A case study of the impact of misappropriation of Kominfo 4G BTS project funds on e-government governance in Indonesia," n.d. Available: https://jpabdimas.idjournal.eu/index.php/soshum/article/view/4022

[8] Y. Xiao, M. Varvello, and A. Kuzmanovic, "Monetizing spare bandwidth: The case of distributed VPNs," in *Measurement and Modeling of Computer Systems*, 2022. https://doi.org/10.1145/3489048.3530966

[9] S. Sasikumar *et al.*, "Unlocking complex patterns through the power of deep learning for advanced data analysis and prediction," pp. 621–625, 2024. https://doi.org/10.1109/cybercom63683.2024.10803222

[10] I. Ferjani, M. S. Hidri, and A. Frihida, "Dynamic knowledge expansion: Real-time text classification with deep convolutional neural networks," 2024. https://doi.org/10.21203/rs.3.rs-3826991/v1

[11] A. Jenefa and V. Nair, "A robust deep learning-based approach for network traffic classification using CNNs and RNNs," in *IEEE Conference on Systems, Process and Control*, pp. 106–110, 2023. https://doi.org/10.1109/ICSPC57692.2023.10125858

[12] Y. Zhang and Z. Wang, "Feature engineering and model optimization based classification method for network intrusion detection," *Applied Sciences*, 2023. https://doi.org/10.3390/app13169363

[13] P. Maxwell, E. Alhajjar, and N. D. Bastian, "Intelligent feature engineering for cybersecurity," in *International Conference on Big Data*, pp. 5005–5011, 2019. https://doi.org/10.1109/BIGDATA47090.2019.9006122

[14] C. Tripp *et al.*, "An empirical deep dive into deep learning's driving dynamics," *arXiv*, 2022. https://doi.org/10.48550/arXiv.2207.12547

[15] S. Chinta, "Advancements in deep learning architectures: A comparative study of performance metrics and applications in real-world scenarios," *SSRN Electronic Journal*, 2025. https://doi.org/10.2139/ssrn.5046551

[16] X. Ma, G. Chen, H. Zhang, and T. Qin, "Towards hierarchically extracting network topologies," pp. 386–392, 2024. https://doi.org/10.1109/nana63151.2024.00071

[17] N. Nurdianto, M. L. Singgih, and I. K. Gunarta, "Building composite performance index for broadband internet customer experience," pp. 669–675, 2024. https://doi.org/10.1109/isct62336.2024.10791161

[18] D. C. Marcu and C. Grava, "The importance of data quality in training a deep convolutional neural network," in *International Conference on Engineering of Modern Electric Systems*, pp. 1–4, 2023. https://doi.org/10.1109/EMES58375.2023.10171785

[19] F. Yu *et al.*, "A survey of large-scale deep learning serving system optimization: Challenges and opportunities," *arXiv*, 2021. Available: https://arxiv.org/abs/2111.14247

[20] F. Yu *et al.*, "Characterizing and understanding deep neural network batching systems on GPUs," *BenchCouncil Transactions on Benchmarks, Standards and Evaluations*, 2023. https://doi.org/10.1016/j.tbench.2024.100151

[21] O. Matsiievskyi *et al.*, "Application of neural networks to optimize distributed computing in cloud and edge environments," pp. 1–5, 2025. https://doi.org/10.1109/sist61657.2025.11139218

[22] I. Ferjani, M. S. Hidri, and A. Frihida, "Dynamic knowledge expansion: Real-time text classification with deep convolutional neural networks," 2024. https://doi.org/10.21203/rs.3.rs-3826991/v1

[23] B. Özkilbaç, I. Y. Ozbek, and T. Karacali, "Real-time fixed-point hardware accelerator of convolutional neural network on FPGA based," in *International Conference on Computing and Information*, pp. 1–5, 2022. https://doi.org/10.1109/icci54321.2022.9756093

[24] M. Z. Hussain *et al.*, "A fine-tuned convolutional neural network model for accurate Alzheimer's disease classification," *Scientific Reports*, vol. 15, 11616, 2025. https://doi.org/10.1038/s41598-025-86635-2

[25] N. Jayakrishna and N. N. Prasanth, "A hybrid deep learning model for detection and mitigation of DDoS attacks in VANETs," *Scientific Reports*, vol. 15, 34170, 2025. https://doi.org/10.1038/s41598-025-15215-1

[26] R. R. Singh, H. Fida, A. Kumar, and A. Kumar, "Leveraging CNNs for accurate weather prediction: A comparative analysis of custom and transfer learning models," in *Innovative Computing and Communications (ICICC 2025)*, LNNS, vol. 1431, Springer, Singapore, 2025. https://doi.org/10.1007/978-981-96-6681-2_39

[27] S. Sinha and Y. M. Lee, "Challenges with developing and deploying AI models and applications in industrial systems," *Discover Artificial Intelligence*, vol. 4, p. 55, 2024. https://doi.org/10.1007/s44163-024-00151-2

[28] B. Roth *et al.*, "Specification overfitting in artificial intelligence," *Artificial Intelligence Review*, vol. 58, p. 35, 2025. https://doi.org/10.1007/s10462-024-11040-6

[29] B. Russell, “Understanding overfitting in AI and its impact on cybersecurity,” *World Journal of Advanced Research and Reviews*, vol. 27, no. 2, pp. 361–372, 2025. https://doi.org/10.30574/wjarr.2025.27.2.2859

[30] Z. Huang, N. Ma, S. Wang, and Y. Peng, “GPU computing performance analysis on matrix multiplication,” *The Journal of Engineering*, vol. 2019, no. 23, pp. 9043–9048, 2019. https://doi.org/10.1049/joe.2018.9178

[31] Y. Hong and D. Kim, “Performance and efficiency gains of NPU-based servers over GPUs for AI model inference,” *Systems*, vol. 13, no. 9, p. 797, 2025. https://doi.org/10.3390/systems13090797