International Journal for Multidisciplinary Research (IJFMR)

E-ISSN: 2582-2160 • Website: www.ijfmr.com • Email: editor@ijfmr.com

Real Time Network Intrusion Detection With Optimized Inference

R.J.S.Surya[1], P. Purna Chandra Reddy[2], P. Kaushik.V.V.K[3], V. Asis Marceline[4], P.Himavanth Sai[5]

[1,2,3,4,5]Department Of Computational Intelligence, SRM Institute Of Science and Technology, Chennai, India

Abstract

Intrusion Detection Systems (IDS) are essential for protecting modern networked environments against increasingly sophisticated cyber threats. Although machine learning techniques have improved detection capabilities, their deployment in real-time systems is often limited by inference latency and scalability constraints. This paper presents a deep learning-based IDS using a multilayer perceptron (MLP) trained on the NSL-KDD dataset, with emphasis on optimizing inference performance for practical deployment scenarios.

To address deployment challenges, the trained model is exported to ONNX format and evaluated using multiple execution backends, including ONNX Runtime and TensorRT on both CPU and GPU platforms. Additionally, the model is deployed using the Triton Inference Server to assess performance in a production-oriented setting. The proposed pipeline enables a comprehensive comparison of latency and throughput across different inference frameworks. Experimental results show that optimized inference using TensorRT achieves up to 4.8 times reduction in latency compared to the baseline PyTorch implementation, while maintaining comparable classification performance. The findings demonstrate that deployment-aware optimization plays a crucial role in building efficient IDS solutions and provide insights into selecting appropriate inference strategies for low-latency, high-throughput environments.

Keywords: Intrusion Detection System (IDS), Deep Learning, Multilayer Perceptron (MLP), NSL-KDD, Model Optimization, ONNX Runtime, TensorRT, Triton Inference Server, Low-Latency Inference, Network Security.

INTRODUCTION

With the rapid expansion of digital communication and networked systems, ensuring the security of data and infrastructure has become increasingly important. Cyber threats such as unauthorized access, denial-of-service attacks, and data breaches continue to evolve in complexity, making traditional security mechanisms insufficient. Intrusion Detection Systems (IDS) have therefore emerged as a critical component in network security, enabling the identification of malicious activities through continuous monitoring of network traffic [4].

Conventional IDS approaches often rely on signature-based techniques, which are effective in detecting known attacks but struggle to identify new or evolving threats. To overcome these limitations, machine learning and deep learning methods have been widely explored for intrusion detection tasks. These

approaches can learn patterns from large-scale network data and adapt to variations in attack behavior, improving detection capability over traditional methods [6], [10]. In particular, supervised learning models trained on benchmark datasets such as NSL-KDD have demonstrated promising results in classifying normal and malicious traffic [2], [7].

Despite these advancements, deploying machine learning-based IDS models in real-world environments introduces several practical challenges. High inference latency, limited throughput, and inefficient utilization of hardware resources can hinder their effectiveness in time-sensitive applications. While many studies focus primarily on improving detection accuracy, comparatively less attention has been given to optimizing inference performance for deployment scenarios [1]. This gap becomes significant in production systems where rapid response and scalability are essential.

Recent developments in model optimization and inference frameworks have provided new opportunities to address these challenges. Technologies such as ONNX Runtime and TensorRT enable efficient execution of trained models across different hardware platforms, while inference servers like Triton facilitate scalable deployment in production environments. These tools allow models to be transformed and executed in optimized formats, reducing latency and improving throughput without requiring changes to the core model architecture [1], [9].

In this work, we present a deep learning-based intrusion detection system using a multilayer perceptron (MLP) trained on the NSL-KDD dataset. Unlike many existing studies that focus solely on detection performance, this work emphasizes deployment-aware optimization. The trained model is exported to ONNX format and evaluated using multiple inference backends, including ONNX Runtime and TensorRT on both CPU and GPU platforms. Furthermore, the model is deployed using the Triton Inference Server to analyze its performance in a production-oriented setting.

The main contributions of this work are as follows:

1. Design and implementation of a deep learning-based IDS using a lightweight MLP architecture.
2. Development of an end-to-end optimization pipeline incorporating ONNX, TensorRT, and Triton.
3. Comprehensive evaluation of inference performance in terms of latency and throughput across multiple deployment configurations.

BACKGROUND AND RELATED WORK

Intrusion Detection Systems (IDS) are designed to monitor network activity and identify malicious behavior that may compromise system security. Broadly, IDS techniques can be categorized into signature-based and anomaly-based approaches. Signature-based systems rely on predefined patterns to detect known attacks, whereas anomaly-based systems model normal network behavior and identify deviations as potential threats. While signature-based methods are efficient for known attack patterns, they are less effective against previously unseen or evolving threats, which has led to increased interest in machine learning-based approaches [10].

Machine learning techniques have been widely applied to intrusion detection due to their ability to learn patterns from large volumes of network data. Various supervised and unsupervised algorithms, including support vector machines, decision trees, and neural networks, have been evaluated for detecting and classifying network intrusions. Comparative studies have shown that ensemble methods and deep learning models can achieve improved detection accuracy compared to traditional approaches, particularly when trained on benchmark datasets such as NSL-KDD and UNSW-NB15 [6], [7]. These datasets provide

labeled network traffic records that enable systematic evaluation of detection models under controlled conditions.

Recent research has also explored deep learning architectures for intrusion detection, including recurrent neural networks and convolutional neural networks, which are capable of capturing complex patterns in network traffic. For instance, optimization-based deep learning approaches have demonstrated high detection accuracy when applied to the NSL-KDD dataset, highlighting the effectiveness of feature selection and parameter tuning in improving model performance [2]. However, many of these studies primarily focus on classification accuracy and do not address the practical challenges associated with deploying such models in real-world environments.

Alongside improvements in detection models, there has been growing interest in enhancing the efficiency of machine learning inference. The increasing complexity of deep learning models has introduced computational overhead, making real-time deployment more challenging. Model optimization techniques such as quantization, hardware acceleration, and optimized inference engines have been proposed to address these issues. Frameworks like ONNX Runtime enable cross-platform model execution, while TensorRT provides hardware-specific optimizations for GPU-based inference, significantly reducing latency and improving throughput [9].

In addition to model optimization, scalable deployment frameworks have become an important area of research. Inference servers such as Triton allow models to be deployed in production environments with features such as dynamic batching, multi-model management, and hardware-aware scheduling. Comparative analyses of deployment frameworks have shown that while lightweight APIs offer flexibility, dedicated inference servers provide better scalability and resource utilization for large-scale applications [1].

Despite these advancements, a gap remains between high-accuracy detection models and their efficient deployment in real-world systems. Many existing works treat model performance and deployment efficiency as separate concerns, without providing an integrated evaluation of both aspects. This limitation motivates the need for a unified approach that considers not only detection capability but also inference latency, throughput, and scalability.

In this context, the present work focuses on bridging this gap by combining a deep learning-based IDS with an optimized inference pipeline. By integrating ONNX Runtime, TensorRT, and Triton Inference Server within a single workflow, this study provides a comprehensive analysis of deployment-oriented performance, offering practical insights into building efficient and scalable intrusion detection systems.

LITERATURE SURVEY

Several research efforts have explored the use of machine learning techniques for improving intrusion detection performance across different datasets and environments. Earlier studies focused on evaluating classical algorithms such as Support Vector Machines, Decision Trees, and Naïve Bayes for detecting network intrusions. These methods were typically applied to benchmark datasets like NSL-KDD, where they demonstrated the ability to classify network traffic with reasonable accuracy while maintaining relatively low computational complexity [7], [11].

More recent works have shifted towards deep learning-based approaches to better capture complex and non-linear relationships within network data. Models such as multilayer perceptrons, recurrent neural networks, and convolutional architectures have been applied to intrusion detection tasks with improved classification results. For instance, deep learning-based intrusion detection systems using optimized

feature selection techniques have reported high accuracy when trained on NSL-KDD, highlighting the effectiveness of combining neural models with data preprocessing strategies [2].

In addition to model design, several studies have conducted comparative evaluations of multiple algorithms to identify the most suitable approach for intrusion detection. These comparisons often reveal that while certain models achieve higher accuracy, they may also introduce increased computational overhead. Ensemble methods such as Random Forest have been found to provide a balance between performance and efficiency, whereas deep learning models tend to require more resources for both training and inference [6].

Another important direction in the literature involves the use of alternative datasets and feature engineering techniques. The UNSW-NB15 dataset has been widely used to assess the robustness of intrusion detection models under more realistic traffic conditions. Studies utilizing this dataset have shown that preprocessing techniques, including feature selection and normalization, play a crucial role in improving detection capability and generalization across different network environments [8].

While these studies have contributed significantly to improving detection accuracy, relatively fewer works have addressed the practical challenges of deploying these models in real-time systems. The increasing complexity of deep learning models often leads to higher inference latency, which can limit their applicability in time-sensitive environments. Recent research has therefore started to explore model optimization techniques and efficient inference frameworks to reduce computational overhead. Approaches involving optimized execution engines and hardware acceleration have demonstrated improvements in inference speed while maintaining acceptable levels of accuracy [9].

Overall, the existing literature highlights substantial progress in both model development and performance evaluation. However, there remains a lack of integrated studies that jointly consider detection effectiveness and deployment efficiency. This gap motivates the need for approaches that not only achieve reliable intrusion detection but also ensure efficient execution in practical deployment scenarios.

DATASET DESCRIPTION

The experiments in this study use the NSL-KDD dataset, which is popular in intrusion detection research. This dataset was released as an improved version of the earlier KDD Cup 1999 dataset to fix problems like duplicate entries and uneven data distribution that affected earlier evaluations [15]. By minimizing these limitations, NSL-KDD provides a more balanced and reliable dataset for training and testing machine learning models.

The dataset consists of network connection records, with each entry described using 41 features and a label indicating whether the activity is normal or an attack. These features highlight different aspects of network behavior, including connection properties, traffic-related information, and statistical patterns. This mix allows the model to learn how normal and malicious activities differ.

NSL-KDD includes predefined training and testing sets, called KDDTrain and KDDTest. The training set is used to train the model, while the test set evaluates its performance on new data. In this study, the training set has 125,973 samples, and the test set has 22,544 samples, which is enough for building and validating the model.

In this study, the original multi-class labels are simplified into a binary classification problem. All normal traffic is labeled as 0, while all attack types are grouped under the label 1. This simplification helps the model focus on identifying whether a connection is safe or potentially harmful, which is often needed in various practical applications.

Before inputting the data into the model, several preprocessing steps are taken. Categorical features such as protocol type, service, and flag are converted into numerical values using label encoding. Then, feature scaling is done using standard normalization to ensure all input values fall within a similar range. These steps help improve the learning process and ensure stable model performance during training.

PROPOSED MODEL ARCHITECTURE

A. Data Collection and Preprocessing

The proposed system utilizes the NSL-KDD dataset as a benchmark dataset for intrusion detection. This dataset provides standardized network traffic records, enabling reproducible evaluation of machine learning models. The dataset is divided into training and testing subsets to ensure proper validation on unseen data.

Categorical features such as protocol type, service, and flag are converted into numerical form using label encoding. Subsequently, all features are normalized using standard scaling to achieve zero mean and unit variance. This ensures uniform feature contribution during training and improves numerical stability. The preprocessing parameters are saved and reused during inference to maintain consistency

B. Neural Network Training

After preprocessing, the data is transformed into tensor format and organized into mini-batches for efficient training. A multilayer perceptron (MLP) is employed to learn patterns from network traffic data. The model is trained to distinguish between normal and malicious activity using labeled examples from the dataset.

To address class imbalance, weighted binary cross-entropy loss is used. The Adam optimizer is applied for parameter updates, along with a learning rate scheduler to improve convergence. The model is trained over multiple epochs, and the best-performing model is saved based on evaluation performance.

C. Input Representation and Model Architecture

Each network connection is represented as a 41-dimensional feature vector after preprocessing. The MLP model consists of three hidden layers with 256, 128, and 64 neurons, respectively. ReLU activation functions are used to introduce non-linearity, while batch normalization and dropout are applied to improve training stability and reduce overfitting.

The final layer produces a single logit value, which is converted into a probability score for binary classification.

D. ONNX Conversion

Once training is completed, the model is exported to the Open Neural Network Exchange (ONNX) format. This conversion enables interoperability across different inference frameworks and hardware platforms. The ONNX representation preserves the model structure and allows it to be executed efficiently using optimized runtimes.

E. TensorRT Optimization

The ONNX model is further optimized using TensorRT to improve inference performance. Platform-specific optimizations, including kernel fusion and precision calibration, are applied to reduce latency and increase throughput. Both FP32 and FP16 precision modes are evaluated to analyze the trade-off between performance and computational efficiency.

F. Triton Deployment

For production-level evaluation, the optimized model is deployed using the Triton Inference Server. Triton provides a scalable serving environment with support for multiple models, hardware acceleration, and

efficient request handling. This step allows the system to simulate real-world deployment conditions and evaluate performance under a serving framework.

G. Performance Evaluation Pipeline

The overall system follows a structured pipeline:

1. Benchmark dataset selection (NSL-KDD) for standardized evaluation
2. Neural network training on preprocessed network traffic data
3. Conversion of trained model to ONNX format
4. Optimization using TensorRT for reduced latency
5. Deployment using Triton Inference Server
6. Performance evaluation based on accuracy, latency, and throughput

This pipeline ensures that the model is not only trained effectively but also optimized and evaluated for real-world deployment scenarios. By integrating training, optimization, and deployment into a unified workflow, the system provides a comprehensive approach to building efficient intrusion detection solutions.

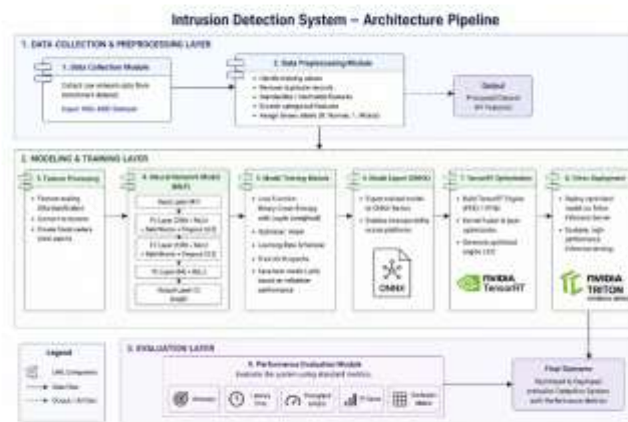

Fig. 1. Architecture pipeline of the proposed intrusion detection system

METHODOLOGY

The proposed system follows a structured pipeline that integrates data preprocessing, model training, optimization, and deployment to enable efficient intrusion detection. The methodology is designed not only to achieve reliable classification performance but also to ensure low-latency inference suitable for real-world applications.

A. Data Acquisition and Preprocessing

The NSL-KDD dataset is used as the input source for this study. It provides labeled network traffic records that represent both normal and malicious activities. The dataset is divided into predefined training and testing sets to evaluate model generalization.

Preprocessing is applied to convert raw data into a suitable format for model training. Categorical attributes, including protocol type, service, and flag, are transformed into numerical values using label encoding. To maintain consistency, encoding is performed using combined data from both training and testing sets.

Feature normalization is performed using standard scaling, ensuring that each feature has zero mean and unit variance. This step improves numerical stability and accelerates convergence during training. The

preprocessing parameters are stored and reused during inference to maintain consistency across all stages of the pipeline.

B. Data Preparation

After preprocessing, the dataset is transformed into numerical tensors compatible with the deep learning framework. The labels are converted into binary form, where normal traffic is represented as 0 and all attack types are grouped under 1.

The processed data is then organized into mini-batches using data loaders. This enables efficient computation and improves memory utilization during training. Batch processing also helps stabilize gradient updates and improves overall learning performance.

C. Model Design and Training

The intrusion detection model is implemented as a multilayer perceptron (MLP) designed for tabular data classification. The network consists of an input layer corresponding to 41 features, followed by three hidden layers with 256, 128, and 64 neurons, respectively. ReLU activation functions are applied to introduce non-linearity, while batch normalization and dropout are used to improve generalization and prevent overfitting.

The model is trained using Binary Cross-Entropy loss with logits, which combines sigmoid activation and loss computation in a numerically stable manner. To address class imbalance, a weighting factor is applied to the positive class.

The Adam optimizer is used for parameter updates due to its adaptive learning capability. A learning rate scheduler is employed to reduce the learning rate when performance improvements plateau. The model is trained for multiple epochs, and the best-performing model is selected based on evaluation accuracy.

D. Model Export and Interoperability

Once training is completed, the model is exported to the ONNX (Open Neural Network Exchange) format. This step enables interoperability across different platforms and inference engines. The ONNX representation preserves the trained model structure and allows it to be executed using optimized runtimes without modifying the original architecture.

E. Inference Optimization

To improve inference efficiency, the ONNX model is evaluated using ONNX Runtime on both CPU and GPU environments. This provides a baseline for comparing performance across different execution backends.

Further optimization is performed using TensorRT, which applies hardware-specific optimizations such as kernel fusion and precision tuning. Both FP32 and FP16 precision modes are considered to analyze performance improvements in terms of latency and throughput.

F. Deployment Using Triton Inference Server

The optimized model is deployed using the Triton Inference Server to simulate a production environment. Triton provides features such as model management, request handling, and hardware acceleration, enabling scalable and efficient inference. This stage evaluates how the optimized model performs under real-world serving conditions.

H. Performance Evaluation

The final stage of the methodology involves evaluating the system using standard performance metrics. Classification performance is measured using accuracy and related metrics, while system efficiency is assessed using latency and throughput.

Latency is measured as the time required to process a single request, and throughput is measured as the

number of requests processed per second. These metrics provide a comprehensive understanding of both detection effectiveness and deployment efficiency.

RESULTS AND PERFORMANCE ANALYSIS

This section evaluates the performance of the proposed intrusion detection system from two perspectives: how well it detects intrusions and how efficiently it performs during inference. The analysis is based on experimental results obtained using different inference frameworks, including PyTorch, ONNX Runtime, TensorRT, and the Triton Inference Server.

A. Evaluation Metrics

The proposed intrusion detection system's performance is evaluated using a mix of classification and system-level measures. For classification, overall accuracy indicates how many predictions are correct across the dataset. To better understand how the model deals with different classes, precision, recall, and F1-score are included. Precision shows how many predicted attack instances are correct. Recall reflects how well the model identifies real attacks. The F1-score offers a balanced measure of these two factors.

To assess how the system works in real situations, latency and throughput are examined. Latency measures the time needed to make a single prediction, with lower values showing faster responses. Throughput counts the number of requests processed each second, reflecting how efficiently the system handles larger workloads.

Additionally, performance improvement is measured by speedup compared to the baseline PyTorch implementation. This allows for a direct comparison of how different optimization techniques affect execution efficiency. Together, these metrics give a well-rounded view of both detection performance and real-world system behavior.

B. Quantitative Performance Comparison

The performance of the proposed system is evaluated across different inference frameworks, including the PyTorch baseline and optimized approaches such as ONNX Runtime, TensorRT, and Triton Inference Server. Although the same trained MLP model is used in all cases, the difference lies in how efficiently each framework executes the model during inference. The results of this comparison are summarized in Table 1.

Method	Latency (ms)	Throughput (req/s)
PyTorch (GPU)	0.52	1,917
ONNX Runtime (CPU)	0.035	28,867
ONNX Runtime (CUDA)	0.26	3,873
TensorRT (FP32)	0.107	9,338
TensorRT (FP16)	0.120	8,315
Triton (FP32)	1.18	847
Triton (FP16)	1.20	827

Table 1. Quantitative performance comparison of inference frameworks.

From the observations, ONNX Runtime on CPU delivers the best performance, achieving very low latency along with high throughput. This suggests that for relatively lightweight models, optimized CPU execution can be more efficient than GPU-based methods. TensorRT also improves performance significantly compared to the baseline, reducing inference time while maintaining strong throughput. In contrast, the

PyTorch baseline provides moderate performance, and ONNX Runtime with CUDA shows only limited gains, indicating that GPU-related overhead can affect efficiency in such cases.
The Triton Inference Server shows comparatively higher latency and lower throughput, mainly due to additional overhead from request handling and system-level operations. However, it is designed for deployment scenarios where scalability and efficient request management are important. Overall, the results show that ONNX Runtime is well suited for maximum efficiency, TensorRT is effective for optimized GPU inference, and Triton is more appropriate for real-world deployment environments.

C. Inference Latency Analysis

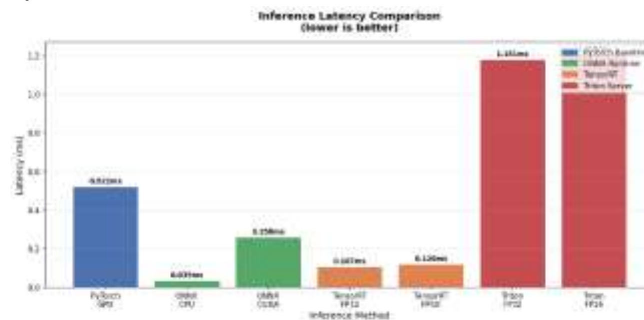

Fig. 2. Latency comparison of inference methods.

Fig. 2 compares the latency of different inference methods. The baseline PyTorch implementation on GPU records a latency of about 0.52 ms, which serves as a reference for further comparison.
Among all methods, ONNX Runtime on CPU achieves the lowest latency of 0.035 ms, showing a clear advantage for lightweight models. ONNX Runtime with CUDA performs moderately, with a latency of 0.26 ms, which suggests that GPU execution introduces some overhead in this case. TensorRT further improves GPU performance, achieving 0.107 ms (FP32) and 0.120 ms (FP16), indicating effective optimization with minimal impact from reduced precision.
On the other hand, Triton deployment results in higher latency values of around 1.18 ms (FP32) and 1.20 ms (FP16). This increase is expected, as Triton involves additional processing such as request handling and communication overhead.

D. Throughput Analysis

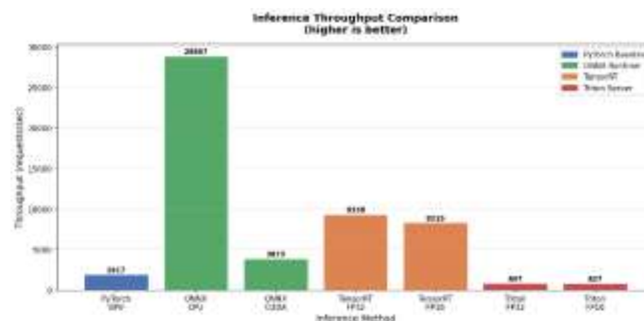

Fig. 3. Throughput comparison across different inference frameworks.

Fig. 3 presents the throughput comparison across different inference frameworks. ONNX Runtime on CPU achieves the highest throughput, reaching approximately 28,867 requests per second, which highlights its efficiency for models of this size.

TensorRT also performs well, achieving 9,338 req/s (FP32) and 8,315 req/s (FP16), providing a good balance between speed and GPU utilization. ONNX Runtime with CUDA reaches 3,873 req/s, which is lower than CPU performance, again indicating overhead in GPU execution for smaller models.
The PyTorch baseline achieves 1,917 req/s, while Triton records the lowest throughput, with 847 req/s (FP32) and 827 req/s (FP16). Despite this, Triton remains useful for deployment scenarios where scalability and request management are important.

E. Speedup Analysis

Fig. 4. Speedup comparison relative to the PyTorch baseline.

Fig. 4 shows the speedup of each method compared to the PyTorch GPU baseline. ONNX Runtime on CPU achieves the highest improvement, with a speedup of about 15.1×, demonstrating that optimized CPU inference can outperform GPU execution for lightweight models.
TensorRT provides strong gains, with speedups of 4.9× (FP32) and 4.3× (FP16), confirming the benefits of GPU-level optimization. ONNX Runtime with CUDA shows a moderate improvement of around 2×.
Triton shows speedup values below 1×, which reflects the additional overhead introduced by the serving framework rather than inefficiency in the model itself.

F. Precision Comparison

Fig. 5. Performance comparison between FP32 and FP16 precision modes.

Fig. 5 compares the performance of FP32 and FP16 precision modes for both TensorRT and Triton. The results show only small differences between the two configurations.
In the case of TensorRT, FP32 slightly outperforms FP16, suggesting that reducing precision does not significantly improve performance for this model. A similar trend is observed with Triton, where both precision modes produce nearly identical results. This indicates that the model size and structure do not strongly benefit from lower precision in this setup.

DISCUSSION AND LIMITATIONS

A. Discussion

The results show that the proposed system benefits significantly from the use of optimized inference frameworks. In particular, ONNX Runtime on CPU achieves very low latency and high throughput, making it a strong choice for lightweight models. TensorRT also improves GPU performance by reducing inference time while maintaining stable throughput. Similar observations have been reported in earlier studies, where optimized runtimes help reduce computational overhead and improve execution efficiency [1], [9]. These results suggest that the choice of inference framework has a direct impact on performance, and in some cases, well-optimized CPU execution can even outperform GPU-based approaches.

The results obtained using the Triton Inference Server provide a more practical view of deployment. Although Triton introduces higher latency due to additional steps such as request handling and communication, it offers useful features like scalability and support for handling multiple requests at once. Prior work has also highlighted this trade-off, where inference servers may reduce raw performance but improve system-level capabilities [1]. This makes Triton more suitable for real-world deployment scenarios rather than direct performance comparison.

B. Limitations

Despite the improvements in inference performance, there are some limitations in the proposed system. The model achieves only moderate classification performance, especially with lower recall for detecting attacks. This means some malicious activities may not be identified, which is a critical concern in intrusion detection systems. Similar challenges have been observed in earlier studies, where improving recall without affecting overall performance remains difficult [6], [7].

Another limitation is the use of a simple multilayer perceptron, which may not fully capture complex patterns in network traffic. More advanced models, such as deep learning or graph-based approaches, are better suited for handling relationships and temporal behavior in data [2], [8]. In addition, the NSL-KDD dataset, although widely used, does not fully reflect current network conditions. Previous research suggests that newer datasets provide more realistic evaluation settings [8]. Finally, the current work focuses mainly on single-request inference and does not explore advanced deployment techniques such as batching or distributed inference, which could further improve performance in real-world applications.

C. Future work

Future work can focus on improving the model by moving beyond the current multilayer perceptron to architectures that better capture patterns in network traffic. Approaches such as recurrent models or graph-based methods have shown stronger capability in learning temporal and relational information in intrusion detection tasks [2], [8]. In addition, evaluating the system on more recent and realistic datasets can provide a better understanding of its performance in modern network environments, as several studies have pointed out the limitations of older benchmark datasets [8]. Incorporating data balancing and feature refinement techniques may also help improve detection performance, particularly for identifying attack instances [6], [7].

From a deployment perspective, further improvements can be achieved by exploring advanced inference strategies such as dynamic batching, asynchronous execution, and distributed serving. Prior work has shown that optimized inference frameworks and scalable deployment solutions can significantly improve system efficiency in real-world settings [1], [9]. Extending the current system to handle real-time data streams and evaluating it under high-load conditions would provide more practical insights into its scalability and robustness.

CONCLUSION

This paper presented an intrusion detection system based on a multilayer perceptron model, with a primary focus on improving inference efficiency for real-world deployment. The system was evaluated using the NSL-KDD dataset, and the trained model was optimized using multiple frameworks, including ONNX Runtime, TensorRT, and Triton Inference Server. The results demonstrate that significant improvements in latency and throughput can be achieved through appropriate optimization techniques, with ONNX Runtime on CPU providing the best overall efficiency and TensorRT offering strong GPU performance. These findings are consistent with prior work highlighting the effectiveness of optimized inference engines in accelerating model execution [1], [9].

The study also highlights the importance of considering deployment aspects alongside model performance. While the baseline model achieves moderate detection accuracy, the optimized inference pipeline enables faster and more scalable execution. At the same time, the results show that deployment frameworks such as Triton introduce additional overhead but provide important benefits for real-world applications, as observed in earlier studies on scalable model serving [1]. Overall, this work emphasizes that both model design and deployment strategy play a key role in building effective and practical intrusion detection systems, aligning with recent research trends in machine learning-based cybersecurity [6], [7].

REFERENCES

1. R. Ali, "Comparing FastAPI and Triton Inference Server for ML Model Deployment," Oct. 2025.
2. M. S. Islam, S. Saha, and M. A. U. Alam, "Intrusion Detection System: An Optimization-Based Deep Learning Approach Using the NSL-KDD Dataset," Proc. IEEE 2nd Int. Conf. Computing, Applications and Systems (COMPAS), 2025.
3. C. Hutabarat and Y. Asnar, "Developing a Machine Learning Module for Intrusion Detection Systems to Detect Unknown Threats," Proc. IEEE Int. Conf. Data and Information, 2024.
4. R. Udayakumar et al., "Machine Learning-Based Intrusion Detection System," Proc. 3rd Int. Conf. Technological Advancements in Computational Sciences (ICTACS), 2023.
5. N. Louloudakis and A. Rajan, "Selective Quantization Tuning for ONNX Models," 2024.
6. A. Quintáns-Fernández et al., "From PyTorch to CUDA and TensorRT: Improving the Deployment of EfficientAD for Real-Time Visual Anomaly Detection," 2024.
7. R. S. Koranga et al., "Comparison of Machine Learning Algorithms for Intrusion Detection Using the NSL-KDD Dataset," 2023.
8. A. Mohamed, J. Heilala, and N. S. Madonsela, "Machine Learning-Based Intrusion Detection Systems for Enhancing Cybersecurity," 2023.
9. D. Gupta and A. K. Saxena, "Using Machine Learning for Network Intrusion Detection," Proc. Int. Conf. Advanced Technologies in Intelligent Control, Environment, Computing & Communication Engineering (ICATIECE), 2022.
10. M. Leon, T. Markovic, and S. Punnekkat, "Evaluation of Machine Learning Algorithms for Network Intrusion Detection and Attack Classification," Proc. IEEE Int. Joint Conf. Neural Networks (IJCNN), 2022.
11. V. Kathiresan et al., "A Study of Intrusion Detection Methods Using Machine Learning Techniques," Proc. Int. Conf. Computer Communication and Informatics (ICCCI), 2022.
12. B. S. Babu et al., "Network Intrusion Detection Using Machine Learning Algorithms," 2022.
13. A. Halimaa and K. Sundarakantham, "Machine Learning-Based Intrusion Detection System," 2021.

14. C. Chen, J. Zhong, and W. Chen, “Web Security Intrusion Detection Using Machine Learning,” Proc. Int. Academic Exchange Conf. Science and Technology Innovation (IAECST), 2021.
15. M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, “A Detailed Analysis of the KDD Cup 99 Data Set,” 2009.