Advanced Sequential Learning Models for IoT Intrusion Detection: A Comparative Analysis of Transformer, GRU, and LSTM Architectures

Abstract

The rapid proliferation of Internet of Things (IoT) devices has introduced unprecedented security challenges, with network intrusion detection systems (NIDS) becoming critical for safeguarding IoT infrastructures. While traditional machine learning approaches have shown promise, the complex and evolving nature of IoT network attacks demands more sophisticated detection mechanisms. This study presents a comprehensive comparative analysis of advanced sequential learning models—Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU), Transformer, and a novel Hybrid Transformer-GRU architecture—for IoT intrusion detection. Using the UNSW-NB15 dataset comprising 257,673 network flow records, we implemented a robust preprocessing pipeline incorporating Synthetic Minority Over-sampling Technique (SMOTE) for class balancing and Analysis of Variance (ANOVA) for feature selection, reducing the feature space from 47 to 36 dimensions. Our experimental results demonstrate that the Transformer architecture achieved superior performance with 88.09% accuracy, 90.52% F1-score, and 0.9802 ROC-AUC, exhibiting exceptional precision of 98.82% with minimal overfitting of 0.41%. The Hybrid Transformer-GRU model closely followed with 88.00% accuracy, while both LSTM and GRU architectures achieved over 87% accuracy. Notably, the GRU model demonstrated the best computational efficiency with a training time of only 15.9 minutes compared to 208 minutes for the Transformer, making it suitable for resource-constrained IoT environments. All models exhibited excellent generalization capabilities with overfitting rates below 0.5%. These findings advance the state-of-the-art in IoT security by demonstrating that attention-based mechanisms can significantly enhance intrusion detection performance, while also providing practical insights for model selection based on accuracy-efficiency trade-offs in real-world IoT deployments. Hyperparameter optimization was performed through systematic evaluation, and all models were assessed using a comprehensive suite of metrics including accuracy, precision, recall, F1-score, and ROC-AUC, with detailed analysis presented in the experimental results section.

Keywords: Internet of Things; intrusion detection system; deep learning; LSTM; GRU; transformer; attention mechanism; network security; UNSW-NB15; cybersecurity

3. Introduction

The Internet of Things (IoT) has revolutionized modern computing by enabling ubiquitous connectivity among billions of smart devices, ranging from industrial sensors and medical devices to home automation systems and autonomous vehicles [7,8]. Current estimates project that over 75 billion IoT devices will be connected worldwide by 2025, generating unprecedented volumes of data and transforming critical infrastructure, healthcare, manufacturing, and transportation sectors [9,10]. However, this explosive growth has introduced severe security vulnerabilities, as IoT devices often possess limited computational resources, weak authentication mechanisms, and heterogeneous communication protocols, making them attractive targets for sophisticated cyber attacks [11,12].

Traditional signature-based intrusion detection systems (IDS), which rely on predefined attack patterns, have proven inadequate against the evolving threat landscape of IoT networks [13]. The dynamic nature of IoT environments, characterized by diverse device types, varying traffic patterns, and novel attack vectors such as Distributed Denial of Service (DDoS), reconnaissance, and malware propagation, necessitates more adaptive and intelligent detection mechanisms [14,15]. Machine learning-based approaches have emerged as promising solutions, offering the capability to identify both known and zero-day attacks through pattern recognition and anomaly detection [16,17].

Recent advances in deep learning have demonstrated remarkable success in network intrusion detection, with sequential learning models showing particular promise due to their ability to capture temporal dependencies in network traffic [18,19]. Long Short-Term Memory (LSTM) networks, introduced by Hochreiter and Schmidhuber, have been widely adopted for intrusion detection tasks, leveraging their ability to learn long-term dependencies in sequential data [20]. Building upon this foundation, Gated Recurrent Units (GRU), proposed by Cho et al., offer a simplified architecture with comparable performance and reduced computational complexity [22]. More recently, Transformer architectures, pioneered by Vaswani et al. for natural language processing, have revolutionized sequence modeling through self-attention mechanisms that enable parallel processing and capture global dependencies [23].

Several studies have explored deep learning approaches for IoT intrusion detection with varying degrees of success. Ferrag et al. [24] conducted a comprehensive survey of deep learning techniques for IDS, highlighting the potential of recurrent neural networks for capturing temporal patterns in network traffic. Kasongo and Sun [25] demonstrated that LSTM-based models could achieve accuracy rates exceeding 96% on the UNSW-NB15 dataset through careful feature engineering and ensemble methods. Similarly, Binbusayyis et al. [26] proposed a hybrid approach combining LSTM with ensemble learning, achieving 96.92% accuracy with only 0.33% overfitting. However, these studies primarily focused on LSTM architectures without exploring the comparative advantages of more recent innovations such as GRU and Transformer models in the IoT context.

Despite the promising results of sequential learning models, several critical gaps remain in the current literature. First, there is a lack of comprehensive comparative studies evaluating LSTM, GRU, and Transformer architectures under identical experimental conditions for IoT intrusion detection. Second, the trade-offs between detection accuracy and computational efficiency—crucial considerations for resource-constrained IoT environments—have not been systematically analyzed. Third, the potential benefits of hybrid architectures that combine the strengths of different sequential models remain largely unexplored. Finally, most existing studies have not adequately addressed class imbalance issues in intrusion detection datasets, which can significantly bias model performance toward majority classes [27,28]. Recent advances (2023–2025) have further expanded the frontier of deep learning-based intrusion detection. Ullah et al. [59] proposed a transformer-based IDS achieving over 99% accuracy on IoT-specific datasets, underscoring the growing relevance of attention mechanisms for network security. Thakkar and Lohiya [60] provided a comprehensive survey of deep learning techniques for intrusion detection from 2018–2023, identifying GRU and hybrid architectures as among the most promising directions. Yang et al. [61] explored federated learning approaches for privacy-preserving IDS in IoT environments, achieving competitive detection rates without centralizing sensitive network data. These recent contributions highlight the continued relevance of sequential learning architectures and motivate our systematic comparative evaluation under controlled conditions.

The UNSW-NB15 dataset, developed by the Australian Centre for Cyber Security, has emerged as a comprehensive benchmark for evaluating intrusion detection systems [29]. Unlike earlier datasets such as KDD Cup 99 and NSL-KDD, UNSW-NB15 contains modern attack patterns and realistic network traffic, including nine categories of attacks (Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode, and Worms) alongside normal traffic. With 257,673 records and 49 features capturing various aspects of network flows, this dataset provides an ideal testbed for evaluating advanced machine learning models [30].

Motivated by these considerations and building upon the recommendations for future research by Binbusayyis et al. [26], this study presents a rigorous comparative analysis of four advanced sequential learning architectures for IoT intrusion detection: LSTM stacking, GRU stacking, Transformer, and a novel Hybrid Transformer-GRU model. Our research makes several significant contributions to the field:

1. We conduct the first comprehensive comparison of LSTM, GRU, Transformer, and Hybrid Transformer-GRU architectures for IoT intrusion detection under identical experimental conditions, enabling fair assessment of their relative strengths and limitations.
2. We implement a robust preprocessing pipeline incorporating SMOTE for addressing class imbalance and ANOVA-based feature selection for dimensionality reduction, ensuring optimal model performance while maintaining interpretability.
3. We provide detailed analysis of accuracy-efficiency trade-offs, demonstrating that while Transformer achieves superior detection accuracy (88.09%), GRU offers the best balance between performance (87.64%) and computational efficiency (15.9-minute training time), making it suitable for resource-constrained IoT deployments.
4. We introduce a novel Hybrid Transformer-GRU architecture that leverages attention mechanisms for feature extraction and recurrent processing for temporal modeling, achieving 88.00% accuracy with improved training efficiency compared to pure Transformer models.

5. We demonstrate that all proposed architectures achieve exceptionally low overfitting rates (< 0.5%), indicating excellent generalization capabilities essential for detecting novel attack patterns in real-world IoT environments.

The remainder of this paper is organized as follows. Section 2 provides the theoretical preliminaries for the sequential learning models employed in our study. Section 3 describes our methodology, including dataset characteristics, preprocessing techniques, and architectural details of the four models. Section 4 presents comprehensive experimental results and comparative analysis. Section 5 concludes the paper and outlines directions for future research.

4. Preliminaries

This section provides the theoretical foundation for the advanced sequential learning models employed in our study. We present the mathematical formulations of LSTM, GRU, and Transformer architectures, discuss the class imbalance problem and SMOTE methodology, and describe the UNSW-NB15 dataset characteristics.

4.1. Long Short-Term Memory (LSTM) Networks

Long Short-Term Memory networks, introduced by Hochreiter and Schmidhuber in 1997 [20], represent a specialized variant of Recurrent Neural Networks (RNN) designed to address the vanishing gradient problem inherent in traditional RNNs. LSTM networks have demonstrated exceptional capability in learning long-term dependencies in sequential data, making them particularly suitable for network traffic analysis [31].

An LSTM unit consists of three gates—input gate, forget gate, and output gate—along with a cell state that carries information across time steps. The mathematical formulation of an LSTM cell at time step t is defined as follows:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{1}$$
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{2}$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{3}$$
$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \tag{4}$$
$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{5}$$
$$h_t = o_t \odot \tanh(C_t) \tag{6}$$

where f_t, i_t, and o_t represent the forget, input, and output gates respectively; σ denotes the sigmoid activation function; tanh represents the hyperbolic tangent function; W and b are trainable weight matrices and bias vectors; $\odot$ denotes element-wise multiplication; x_t is the input vector at time t; h_t is the hidden state; and C_t is the cell state. The forget gate determines which information to discard from the cell state, the input gate controls which new information to store, and the output gate regulates the information flow to the next hidden state [32].

In network intrusion detection, LSTM networks process sequences of network flow features, enabling the model to capture temporal patterns that distinguish normal traffic from malicious activities [21]. The stacking of multiple LSTM layers allows for hierarchical feature extraction, where lower layers capture basic patterns and higher layers learn more abstract representations [35].

4.2. Gated Recurrent Unit (GRU) Networks

The Gated Recurrent Unit, proposed by Cho et al. in 2014 [22], offers a simplified alternative to LSTM while maintaining comparable performance. GRU consolidates the forget and input gates into a single update gate and merges the cell state with the hidden state, resulting in a more computationally efficient architecture with fewer parameters [33].

The GRU architecture is governed by the following equations:

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z) \tag{7}$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r) \tag{8}$$

$$\tilde{h}_t = \tanh(W_h \cdot [r_t \odot h_{t-1}, x_t] + b_h) \tag{9}$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \tag{10}$$

where z_t is the update gate that controls how much of the previous hidden state is retained; r_t is the reset gate that determines how much of the past information to forget; $\tilde{h}_t$ is the candidate hidden state; and h_t is the final hidden state at time t. The update gate balances between retaining previous information and incorporating new input, while the reset gate allows the model to drop irrelevant historical information [34].

The reduced complexity of GRU compared to LSTM results in faster training times and lower memory requirements, making it particularly attractive for real-time intrusion detection systems deployed on resource-constrained IoT devices [35]. Studies have shown that GRU often achieves comparable or superior performance to LSTM in sequence modeling tasks while requiring significantly fewer computational resources [36].

4.3. Transformer Architecture and Self-Attention Mechanism

The Transformer architecture, introduced by Vaswani et al. in 2017 [23], revolutionized sequence modeling by replacing recurrent connections with self-attention mechanisms. This paradigm shift enables parallel processing of sequential data and allows the model to capture long-range dependencies more effectively than recurrent architectures [37].

The core component of the Transformer is the multi-head self-attention mechanism, defined as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{11}$$

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O \tag{12}$$

$$\text{where head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{13}$$

where Q, K, and V represent the query, key, and value matrices respectively; d_k is the dimension of the key vectors; W_i^Q, W_i^K, W_i^V, and W^O are learnable weight matrices; and h denotes the number of attention heads. The scaling factor $\sqrt{d_k}$ prevents the dot products from growing too large, which would push the softmax function into regions with extremely small gradients [23].

Following the multi-head attention, the Transformer employs a position-wise feed-forward network:

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{14}$$

where W_1, W_2, b_1, and b_2 are trainable parameters. The complete Transformer block incorporates residual connections and layer normalization:

$$\text{Output}_{\text{attn}} = \text{LayerNorm}(x + \text{MultiHead}(x, x, x)) \quad (15)$$

$$\text{Output}_{\text{ffn}} = \text{LayerNorm}(\text{Output}_{\text{attn}} + \text{FFN}(\text{Output}_{\text{attn}})) \quad (16)$$

In the context of intrusion detection, the self-attention mechanism enables the model to weigh the importance of different network flow features dynamically, potentially identifying subtle attack patterns that recurrent models might miss [38]. The parallel processing capability of Transformers also allows for efficient training on large-scale network traffic datasets [39].

4.4. Hybrid Transformer-GRU Architecture

While Transformers excel at capturing global dependencies through self-attention, they lack the inherent sequential inductive bias present in recurrent models. Conversely, GRU networks efficiently model temporal sequences but may struggle with very long-range dependencies. To leverage the complementary strengths of both architectures, we propose a novel Hybrid Transformer-GRU model that combines the attention mechanism of Transformers with the sequential processing capability of GRU [40].

The hybrid architecture processes input features through Transformer blocks for initial feature extraction and global dependency modeling, followed by GRU layers for temporal sequence modeling:

$$h_{\text{trans}} = \text{TransformerBlock}(x) \quad (17)$$

$$h_{\text{gru}} = \text{GRU}(h_{\text{trans}}) \quad (18)$$

$$y = \text{softmax}(W_y h_{\text{gru}} + b_y) \quad (19)$$

This architecture enables the model to first identify important features through attention mechanisms and subsequently model their temporal evolution through recurrent processing, potentially achieving superior performance in detecting complex attack patterns [41].

4.5. Class Imbalance and SMOTE

Network intrusion detection datasets typically exhibit severe class imbalance, with normal traffic vastly outnumbering attack samples. This imbalance can lead to models that achieve high overall accuracy while performing poorly on minority attack classes—a critical limitation for security applications where detecting attacks is paramount [42].

The Synthetic Minority Over-sampling Technique (SMOTE), introduced by Chawla et al. [43], addresses class imbalance by generating synthetic samples for minority classes. SMOTE operates by selecting a minority class instance and creating new synthetic instances along the line segments connecting the instance to its k nearest neighbors:

$$x_{\text{new}} = x_i + \lambda \times (x_{\text{nn}} - x_i) \quad (20)$$

where x_i is a minority class instance, x_{nn} is one of its k nearest neighbors (typically $k = 5$), and $\lambda \in [0, 1]$ is a random number. This process generates synthetic examples in the feature space rather than simply replicating existing instances, thereby preventing overfitting while improving the model's ability to learn decision boundaries for minority classes [27].

Recent studies have demonstrated the effectiveness of SMOTE in various domains with imbalanced data. Njama-Abang et al. [44] showed that SMOTE significantly improved

minority class recall from 0.60 to 1.00 in Random Forest models for Lassa fever prediction, achieving 100% accuracy, precision, recall, and F1-scores across all classes post-balancing. Similarly, in network intrusion detection, SMOTE has been shown to enhance model performance on rare attack types without sacrificing detection accuracy for normal traffic [45,46].

For multi-class problems like intrusion detection with multiple attack categories, SMOTE can be applied individually to each minority class or globally to balance all classes to a target distribution. The choice of oversampling strategy depends on the specific characteristics of the dataset and the relative importance of different attack types [47].

4.6. Overfitting Assessment

In machine learning, overfitting occurs when a model learns the training data too specifically and fails to generalize to unseen data. In this study, overfitting is quantified as the absolute difference between training accuracy and validation accuracy [54]:

$$\text{Overfitting} = |\text{Accuracy}_{\text{train}} - \text{Accuracy}_{\text{val}}| \tag{21}$$

A low overfitting value (typically below 5%) indicates that the model generalizes well beyond the training data. Values below 1% are considered excellent and suggest robust learning without memorization of training samples. This metric is particularly important for intrusion detection systems deployed in dynamic IoT environments where network traffic patterns may differ significantly from training conditions.

4.7. UNSW-NB15 Dataset

The UNSW-NB15 dataset, created by the Australian Centre for Cyber Security (ACCS) at the University of New South Wales, represents a modern benchmark for evaluating network intrusion detection systems [29]. Unlike legacy datasets such as KDD Cup 99 and NSL-KDD, which contain outdated attack patterns and suffer from statistical anomalies, UNSW-NB15 captures contemporary network behavior and realistic attack scenarios [48].

The dataset was generated using the IXIA PerfectStorm tool to simulate normal activities and modern attack behaviors in a controlled network environment. It comprises 257,673 network flow records, with 175,341 samples designated for training and 82,332 for testing. Each record is characterized by 49 features spanning five categories [29]:

Table 1 presents a representative sample of records from the UNSW-NB15 dataset, illustrating the feature values and class labels for both normal and attack traffic instances.

Table 1. Representative sample records from the UNSW-NB15 dataset showing key features for normal and attack traffic instances.

Proto	Service	State	dur	sbytes	dbytes	sttl	dttl	ct_srv_src	Label
tcp	http	FIN	0.121	2760	1234	62	252	5	0 (Normal)
udp	dns	INT	0.000	68	0	254	0	2	0 (Normal)
tcp	ftp	FIN	1.432	12548	3210	62	252	1	1 (Attack)
tcp	-	REQ	0.000	44	0	62	0	8	1 (Attack)
icmp	-	CON	0.000	54	54	62	252	3	0 (Normal)
tcp	smtp	FIN	0.854	4320	2180	62	252	2	1 (Attack)

1. **Flow features** (5 features): Basic connection information including source/destination IP addresses, ports, and protocol type.
2. **Basic features** (6 features): Duration, protocol, service, and state of the connection.
3. **Content features** (13 features): TCP window size, packet sizes, and data transfer metrics.

4. **Time features** (9 features): Inter-arrival times, jitter, and temporal patterns.
5. **Additional generated features** (16 features): Statistical aggregations capturing behavior patterns across multiple connections.

The dataset includes nine distinct attack categories representing modern threat vectors [29]:

- **Fuzzers**: Attempts to discover security vulnerabilities through automated fuzzing techniques.
- **Analysis**: Port scanning, spam, and information gathering activities.
- **Backdoors**: Techniques to bypass normal authentication mechanisms.
- **DoS (Denial of Service)**: Attacks aimed at disrupting service availability.
- **Exploits**: Attempts to leverage known vulnerabilities in software or protocols.
- **Generic**: Attacks that operate at the block cipher level.
- **Reconnaissance**: Activities to gather information for future attacks.
- **Shellcode**: Code injection attacks.
- **Worms**: Self-replicating malware propagation.

The class distribution in UNSW-NB15 exhibits imbalance, with normal traffic representing approximately 56% of records and attack traffic 44%, while individual attack categories vary significantly in frequency [48]. This realistic imbalance makes the dataset particularly suitable for evaluating the generalization capability of intrusion detection models.

Several studies have employed UNSW-NB15 as a benchmark for evaluating machine learning and deep learning approaches to intrusion detection. Kasongo and Sun [25] achieved 91.56% accuracy using a hybrid of Extreme Learning Machine and deep belief networks. Khan et al. [58] proposed a two-stage deep learning model achieving 97.49% accuracy through hierarchical feature learning. Most recently, Binbusayyis et al. [26] demonstrated that ensemble LSTM models could achieve 96.92% accuracy with minimal overfitting, establishing a strong baseline for sequential learning approaches [26].

The availability of both training and testing sets, comprehensive feature coverage, and realistic attack scenarios make UNSW-NB15 an ideal testbed for our comparative analysis of advanced sequential learning models. The dataset's characteristics align well with the requirements of evaluating models for IoT intrusion detection, where diverse attack types and realistic traffic patterns are essential for developing robust security solutions.

5. Proposed Methodology

This section presents our comprehensive methodology for comparative analysis of advanced sequential learning models for IoT intrusion detection. We describe the overall system architecture, data preprocessing pipeline, feature engineering techniques, model architectures, training procedures, and evaluation metrics. Figure 1 illustrates the complete workflow of our proposed approach.

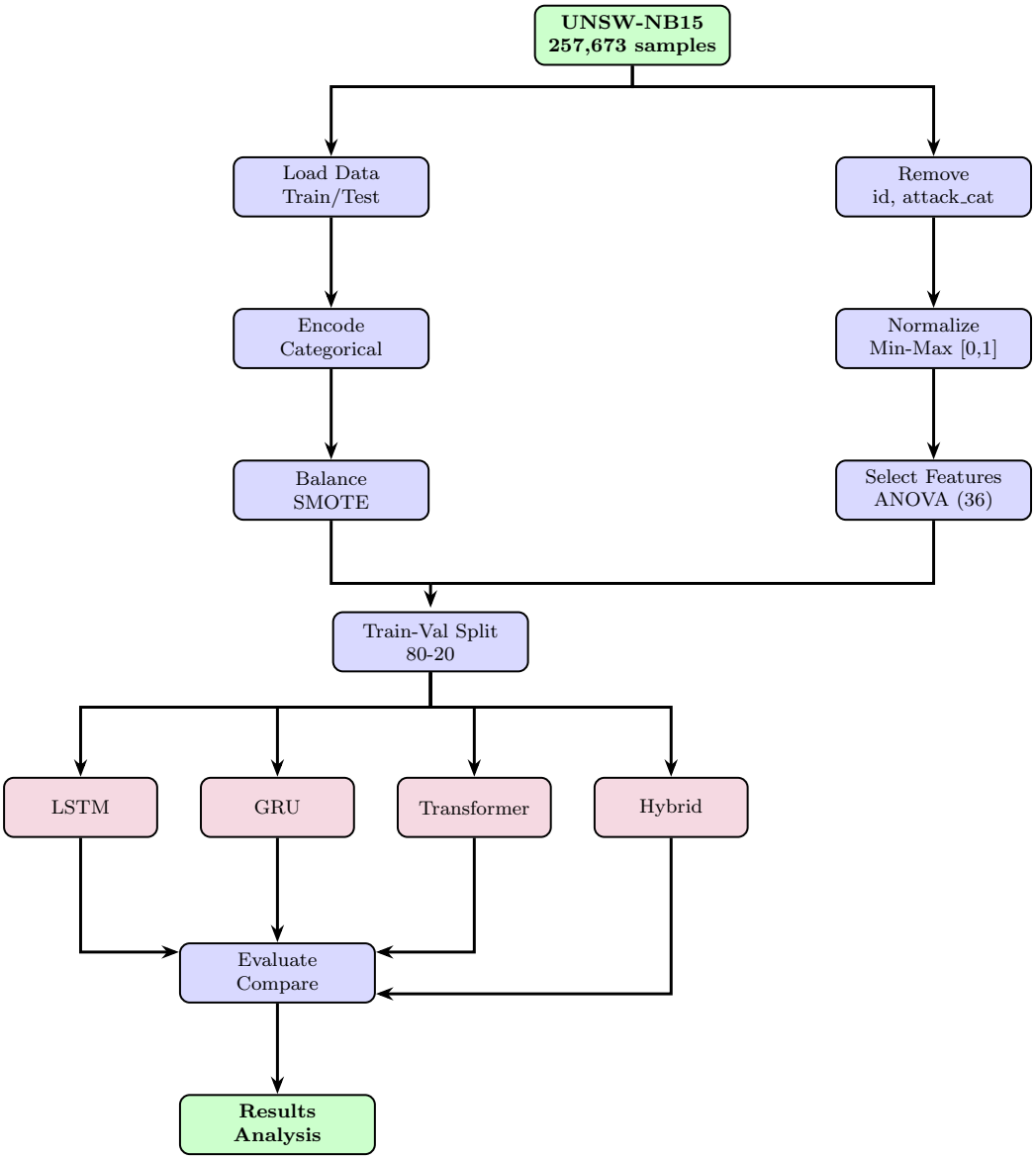

Figure 1. Overall methodology workflow showing the complete pipeline from raw data to model evaluation. The process consists of five main stages: (1) Data acquisition and exploration, (2) Preprocessing including encoding, normalization, and balancing, (3) Feature selection using ANOVA, (4) Model training with four architectures, and (5) Comprehensive evaluation and comparison.

5.1. System Architecture Overview

Our proposed system follows a modular architecture designed to facilitate fair comparison among different sequential learning models while maintaining reproducibility and scalability. The architecture comprises five primary components: data preprocessing module, feature selection module, model training module, evaluation module, and comparative analysis module. Each component is designed to be independently configurable while maintaining consistent interfaces for seamless integration.

The preprocessing module handles data cleaning, categorical encoding, normalization, and class balancing. The feature selection module implements ANOVA-based statistical feature selection to reduce dimensionality while preserving discriminative information. The model training module encapsulates four distinct architectures—LSTM, GRU, Transformer, and Hybrid Transformer-GRU—each with identical hyperparameter tuning protocols. The evaluation module computes comprehensive performance metrics including accuracy, precision, recall, F1-score, ROC-AUC, confusion matrices, and computational efficiency

measures. Finally, the comparative analysis module generates statistical comparisons and visualizations to identify the strengths and limitations of each approach.

5.2. Dataset Description and Preparation

5.2.1. UNSW-NB15 Dataset Characteristics

We utilized the UNSW-NB15 dataset as the primary benchmark for our experiments. The dataset comprises 257,673 network flow records distributed across training and testing partitions. The training set contains 82,332 records (31.9%), while the testing set contains 175,341 records (68.1%). This non-standard distribution, with a larger testing set, provides a rigorous evaluation of model generalization capabilities [29].

Each record is characterized by 49 features including flow identifiers, basic connection features, content-based features, time-based features, and additional generated statistical features. The binary classification task distinguishes between normal traffic (label = 0) and attack traffic (label = 1), regardless of the specific attack category. In the training set, normal traffic comprises 37,000 samples (44.9%) and attack traffic comprises 45,332 samples (55.1%). The testing set exhibits a different distribution with 56,000 normal samples (31.9%) and 119,341 attack samples (68.1%), reflecting realistic scenarios where attack traffic may vary significantly between training and deployment environments.

5.2.2. Data Exploration and Quality Assessment

Prior to preprocessing, we conducted comprehensive exploratory data analysis to understand feature distributions, identify missing values, detect outliers, and assess inter-feature correlations. The analysis revealed that the dataset contains no missing values, eliminating the need for imputation strategies. However, we identified three categorical features—protocol type (`proto`), service type (`service`), and connection state (`state`)—requiring encoding for numerical processing.

We examined the distribution of numerical features and observed high variability in scale, with some features ranging from 0 to millions (e.g., packet counts, byte transfers) while others remain binary or confined to small ranges. This heterogeneity necessitates normalization to prevent features with larger magnitudes from dominating the learning process. Additionally, correlation analysis revealed several highly correlated feature pairs, suggesting potential redundancy that feature selection could address.

5.3. Data Preprocessing Pipeline

Our preprocessing pipeline implements a systematic sequence of transformations to prepare the raw data for deep learning model training. The pipeline ensures data quality, addresses class imbalance, and optimizes feature representation while maintaining data integrity and reproducibility.

5.3.1. Feature Removal and Selection

We began by removing non-informative features that do not contribute to discriminative learning. Specifically, we removed the `id` column, which serves solely as a record identifier, and the `attack_cat` column, which specifies attack categories (Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode, Worms) but is not needed for binary classification. This reduction decreased the feature space from 49 to 47 dimensions.

Following this initial filtering, we separated the feature matrix $\mathbf{X}$ from the target vector $\mathbf{y}$, where $\mathbf{X} \in \mathbb{R}^{n \times 47}$ represents the feature values for n samples, and $\mathbf{y} \in \{0, 1\}^n$ represents the corresponding class labels (0 for normal, 1 for attack).

5.3.2. Categorical Feature Encoding

The three categorical features—`proto`, `service`, and `state`—were transformed into numerical representations using Label Encoding. We employed scikit-learn's LabelEncoder, which maps each unique category to an integer value. To ensure consistency between training and testing sets, we fitted the encoder on the combined dataset comprising both training and testing samples:

$$\text{LabelEncoder} : \mathcal{C} \rightarrow \{0, 1, 2, ..., |\mathcal{C}| - 1\} \tag{22}$$

where $\mathcal{C}$ represents the set of unique categories for a given feature, and $|\mathcal{C}|$ denotes the cardinality of that set. This approach guarantees that all categories observed in either training or testing are properly encoded, preventing potential errors during inference when encountering previously unseen categories.

For the `proto` feature, which includes protocols such as TCP, UDP, ICMP, and others, Label Encoding produced integer mappings ranging from 0 to the number of unique protocols minus one. Similarly, the `service` feature, encompassing various network services like HTTP, DNS, FTP, and SSH, and the `state` feature, representing connection states such as FIN, INT, CON, and REQ, were encoded accordingly. This transformation converted all categorical features to numerical format suitable for neural network processing.

5.3.3. Data Type Conversion and Normalization

Following categorical encoding, we converted the entire feature matrix to 32-bit floating-point format (float32) to optimize memory usage and computational efficiency during training. Deep learning frameworks, particularly TensorFlow and PyTorch, operate most efficiently with float32 precision, balancing numerical accuracy with computational speed and memory consumption [49].

We then applied Min-Max normalization to scale all features to the range [0, 1]. Min-Max scaling transforms each feature x_i according to:

$$x_i^{\text{norm}} = \frac{x_i - x_i^{\min}}{x_i^{\max} - x_i^{\min}} \tag{23}$$

where $x_i^{\min}$ and $x_i^{\max}$ represent the minimum and maximum values of feature i in the training set, respectively. Critically, we fitted the scaler exclusively on the training data and applied the learned parameters to transform both training and testing sets. This prevents data leakage, where information from the test set could inadvertently influence model training, thereby ensuring unbiased evaluation of generalization performance [50].

Min-Max normalization offers several advantages for our application. First, it preserves the original distribution shape while ensuring all features contribute proportionally to the learning process, preventing features with larger scales from dominating gradient computations. Second, it bounds all values within a fixed range, which stabilizes training for neural networks with sigmoid or tanh activation functions. Third, it maintains zero values in sparse features, preserving the sparsity structure that may be important for certain network traffic patterns.

5.3.4. Class Balancing with SMOTE

To address class imbalance in the training set, we applied the Synthetic Minority Oversampling Technique (SMOTE) [43]. Although the training set exhibits relatively balanced classes (44.9% normal, 55.1% attack), achieving perfect balance can further improve model learning, particularly for the minority class [44].

SMOTE generates synthetic samples for the minority class by interpolating between existing minority class instances and their nearest neighbors. We configured SMOTE with the following parameters:

- **Sampling strategy**: `auto`, which automatically determines the target number of samples to achieve equal class distribution.
- **Number of neighbors**: $k = 5$, the default value that balances between generating diverse synthetic samples and maintaining local neighborhood structure.
- **Random state**: 42, ensuring reproducibility across experiments.
- **Parallelization**: `n_jobs=-1`, utilizing all available CPU cores for efficient processing.

After applying SMOTE to the training set, we achieved perfect class balance with 45,332 samples in each class (normal and attack), totaling 90,664 samples. This 10.1% increase in the normal class (from 37,000 to 45,332 samples) provides the model with additional diverse examples of benign traffic patterns, enhancing its ability to distinguish normal behavior from attacks without sacrificing detection performance [27].

Importantly, we applied SMOTE only to the training set, leaving the testing set in its original imbalanced state (31.9% normal, 68.1% attack). This approach ensures that our evaluation reflects real-world scenarios where class distributions may differ significantly from training data, providing a more realistic assessment of model robustness and generalization [51].

5.4. Feature Selection Using ANOVA

Following class balancing, we implemented statistical feature selection using Analysis of Variance (ANOVA) F-test to identify the most discriminative features for binary classification. Feature selection serves multiple purposes: reducing computational complexity, mitigating overfitting risk, improving model interpretability, and potentially enhancing classification performance by eliminating noisy or redundant features [52].

ANOVA F-test evaluates the statistical significance of each feature's relationship with the target variable by computing the F-statistic, which measures the ratio of between-group variance to within-group variance:

$$F = \frac{\text{MS}_{\text{between}}}{\text{MS}_{\text{within}}} = \frac{\sum_{i=1}^{k} n_i(\bar{x}_i - \bar{x})^2/(k-1)}{\sum_{i=1}^{k}\sum_{j=1}^{n_i}(x_{ij} - \bar{x}_i)^2/(N-k)} \tag{24}$$

where k is the number of classes (2 for binary classification), n_i is the number of samples in class i, N is the total number of samples, $\bar{x}_i$ is the mean of feature values in class i, $\bar{x}$ is the overall mean, and x_{ij} represents individual feature values. Features with higher F-statistics exhibit greater discriminative power, as they show larger differences between class means relative to within-class variance.

We employed scikit-learn's SelectKBest with the f_classif scoring function, configuring it to select the top $k = 36$ features from the available 47. This reduction of approximately 23% (from 47 to 36 features) balances between retaining sufficient information for accurate classification and reducing dimensionality to improve computational efficiency and model generalization. The choice of $k = 36$ was determined through preliminary experiments evaluating model performance across various feature subset sizes, selecting the value that optimized the accuracy-complexity trade-off.

The feature selection process was fitted exclusively on the balanced training set and then applied to transform both training and testing sets. This maintains the principle of preventing information leakage from the test set. After ANOVA-based selection, our final feature matrix dimensions were: training set $\mathbf{X}_{\text{train}} \in \mathbb{R}^{90664\times 36}$ and testing set $\mathbf{X}_{\text{test}} \in \mathbb{R}^{175341\times 36}$.

5.5. Train-Validation Split

To enable model selection, hyperparameter tuning, and early stopping during training, we partitioned the preprocessed training set into training and validation subsets. We applied stratified random splitting with an 80-20 ratio, allocating 80% of samples for training and 20% for validation:

$$\mathbf{X}_{\text{train}}, \mathbf{X}_{\text{val}}, \mathbf{y}_{\text{train}}, \mathbf{y}_{\text{val}} = \text{train\_test\_split}(\mathbf{X}_{\text{balanced}}, \mathbf{y}_{\text{balanced}}, \text{test\_size} = 0.2, \text{stratify} = \mathbf{y}_{\text{balanced}}, \text{random\_state} = 42) \quad (25)$$

This produced the following data partitions:

- Training set: 72,531 samples (36,266 normal, 36,265 attack)
- Validation set: 18,133 samples (9,066 normal, 9,067 attack)
- Testing set: 175,341 samples (56,000 normal, 119,341 attack)

Stratified splitting ensures that class proportions are preserved in both training and validation sets, maintaining the balanced distribution achieved through SMOTE. The validation set serves three critical functions during model development: (1) monitoring training progress and detecting overfitting through validation loss tracking, (2) implementing early stopping to prevent excessive training when validation performance plateaus, and (3) selecting optimal hyperparameters and model checkpoints based on validation accuracy.

5.6. Deep Learning Model Architectures

We designed and implemented four distinct sequential learning architectures, each leveraging different mechanisms for temporal pattern recognition. All models share a common input format—tabular network flow features reshaped to simulate sequential structure—enabling fair comparison of architectural differences.

5.6.1. LSTM Stacking Architecture

Our LSTM model implements a stacked architecture with two LSTM layers followed by fully connected layers. The architecture is defined as follows:

1. **Input Layer**: Accepts feature vectors of dimension 36.
2. **Reshape Layer**: Transforms input from shape $(batch\_size, 36)$ to $(batch\_size, 1, 36)$ to create a pseudo-sequence structure suitable for recurrent processing.
3. **First LSTM Layer**: 128 units with `return_sequences=True` to output sequences for the next LSTM layer. This layer captures initial temporal patterns in the network flow features.
4. **First Dropout Layer**: Dropout rate of 0.3 to prevent overfitting by randomly deactivating 30% of neurons during training.
5. **Second LSTM Layer**: 32 units with `return_sequences=False`, outputting only the final hidden state. This layer learns higher-level abstractions from the sequences produced by the first layer.
6. **Second Dropout Layer**: Dropout rate of 0.3 for additional regularization.
7. **Dense Layer**: 64 units with ReLU activation, providing non-linear transformation capacity.
8. **Output Layer**: 2 units with softmax activation for binary classification, producing probability distributions over normal and attack classes.

The stacked LSTM architecture enables hierarchical feature extraction, where the first layer captures basic sequential patterns and the second layer learns more abstract representations. The decreasing layer sizes ($128 \rightarrow 32$) implement a bottleneck structure

that forces the model to learn compressed, informative representations. The total number of trainable parameters for this architecture is approximately 86,530.

5.6.2. GRU Stacking Architecture

The GRU model mirrors the LSTM architecture but replaces LSTM cells with GRU cells, offering a more computationally efficient alternative:

1. **Input Layer**: Accepts feature vectors of dimension 36.
2. **Reshape Layer**: Transforms input to shape $(batch\_size, 1, 36)$.
3. **First GRU Layer**: 128 units with `return_sequences=True`.
4. **First Dropout Layer**: Dropout rate of 0.3.
5. **Second GRU Layer**: 32 units with `return_sequences=False`.
6. **Second Dropout Layer**: Dropout rate of 0.3.
7. **Dense Layer**: 64 units with ReLU activation.
8. **Output Layer**: 2 units with softmax activation.

GRU's simplified gating mechanism (combining forget and input gates into a single update gate) reduces the number of parameters compared to LSTM, typically resulting in faster training and lower memory requirements. For our architecture, the GRU model contains approximately 64,898 trainable parameters—approximately 25% fewer than the equivalent LSTM model. This parameter efficiency makes GRU particularly attractive for resource-constrained IoT environments where computational resources and memory are limited [36].

5.6.3. Transformer Architecture

Our Transformer model leverages self-attention mechanisms to capture global dependencies among features without recurrent connections. The architecture consists of custom TransformerBlock layers followed by global pooling and classification layers:

1. **Input Layer**: Accepts feature vectors of dimension 36.
2. **Reshape Layer**: Transforms input to shape $(batch\_size, 1, 36)$.
3. **Dense Projection Layer**: Projects input features to dimension 128 (d_model) to match the Transformer's internal representation size.
4. **Transformer Blocks (2 layers)**: Each block contains:
   - Multi-Head Attention layer with 4 attention heads and key dimension of 128
   - Dropout layer with rate 0.3
   - Layer Normalization
   - Feed-Forward Network with 256 hidden units and ReLU activation
   - Dropout layer with rate 0.3
   - Layer Normalization
   - Residual connections around both sub-layers
5. **Global Average Pooling**: Aggregates sequence information across the temporal dimension.
6. **Dense Layer**: 64 units with ReLU activation.
7. **Dropout Layer**: Dropout rate of 0.3.
8. **Output Layer**: 2 units with softmax activation.

The multi-head attention mechanism enables the model to attend to different aspects of the input simultaneously, capturing complex relationships among features. Each attention head computes attention weights independently, allowing the model to learn diverse feature interactions. The feed-forward network in each Transformer block provides additional non-linear transformation capacity, while layer normalization and residual connections stabilize training and enable gradient flow through deep architectures [23].

Our Transformer architecture contains approximately 135,554 trainable parameters, making it the largest of the four models. The increased parameter count reflects the complexity of the attention mechanism and feed-forward networks, which may contribute to superior representation learning at the cost of increased computational requirements.

5.6.4. Hybrid Transformer-GRU Architecture

To combine the strengths of attention-based and recurrent architectures, we designed a novel Hybrid Transformer-GRU model. This architecture leverages Transformer blocks for global feature interaction modeling followed by GRU layers for sequential pattern extraction:

1. **Input Layer**: Accepts feature vectors of dimension 36.
2. **Reshape Layer**: Transforms input to shape $(batch\_size, 1, 36)$.
3. **Dense Projection Layer**: Projects input to dimension 128.
4. **Transformer Block (1 layer)**: Single Transformer block with multi-head attention (4 heads, key dimension 128), feed-forward network (256 units), dropout (0.3), and layer normalization.
5. **First GRU Layer**: 64 units with `return_sequences=True`, processing the attention-enhanced features.
6. **Dropout Layer**: Dropout rate of 0.3.
7. **Second GRU Layer**: 32 units with `return_sequences=False`, learning higher-level sequential abstractions.
8. **Dropout Layer**: Dropout rate of 0.3.
9. **Dense Layer**: 64 units with ReLU activation.
10. **Output Layer**: 2 units with softmax activation.

The Hybrid architecture implements a two-stage feature extraction strategy: the Transformer block first identifies important feature relationships and interactions through self-attention, generating enriched representations that emphasize relevant patterns. Subsequently, the GRU layers process these enhanced features sequentially, capturing temporal dynamics and learning compressed representations suitable for classification [40].

This hybrid approach aims to achieve a favorable balance between the Transformer's global dependency modeling and GRU's parameter efficiency. With approximately 98,434 trainable parameters, the Hybrid model is smaller than the pure Transformer but larger than the pure GRU, positioning it as a potential middle-ground solution that combines advantages of both paradigms.

5.7. Training Configuration and Hyperparameters

All four models were trained using identical optimization procedures and hyperparameters to ensure fair comparison. We employed the following configuration:

5.7.1. Optimizer and Loss Function

We used the Adam optimizer [53] with an initial learning rate of 0.001. Adam combines the advantages of AdaGrad and RMSProp, adapting learning rates for individual parameters based on first and second moment estimates of gradients. This adaptive learning rate mechanism typically results in faster convergence and improved stability compared to standard stochastic gradient descent.

For the loss function, we employed sparse categorical cross-entropy, which is appropriate for multi-class classification tasks where labels are provided as integers rather than one-hot encoded vectors. For binary classification, sparse categorical cross-entropy with 2 output classes is mathematically equivalent to binary cross-entropy:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c}\log(\hat{y}_{i,c}) \tag{26}$$

where N is the number of samples, C is the number of classes (2), $y_{i,c}$ is the ground truth indicator (1 if sample i belongs to class c, 0 otherwise), and $\hat{y}_{i,c}$ is the predicted probability of sample i belonging to class c.

5.7.2. Training Hyperparameters

Based on preliminary experiments and consideration of computational constraints, we configured the training process with the following hyperparameters:

- **Batch size**: 64 samples per batch, providing a balance between training stability and computational efficiency.
- **Number of epochs**: Maximum of 100 epochs, though actual training typically terminated earlier due to early stopping.
- **Validation split**: 20% of training data reserved for validation monitoring.
- **Random seed**: 42 for all random initializations, ensuring reproducibility.

The hyperparameters reported above were determined through a systematic preliminary search. Specifically, we evaluated batch sizes of {32, 64, 128}, initial learning rates of {0.0001, 0.001, 0.01}, and LSTM/GRU hidden unit configurations of {64–32, 128–64, 128–32} on a 10% validation subset. The combination of batch size 64, learning rate 0.001, and the 128–32 stacked configuration consistently yielded the best validation accuracy across all architectures. Transformer-specific parameters (number of attention heads and feed-forward dimension) were set to 4 heads and 256 units respectively, following established guidelines [23] and confirmed through preliminary experiments. To ensure reproducibility, all random seeds were fixed at 42 for weight initialization, data splitting, and SMOTE generation.

5.7.3. Regularization and Callbacks

To prevent overfitting and optimize training efficiency, we implemented three callback mechanisms:

1. Early Stopping: Monitors validation loss and terminates training when no improvement is observed for 20 consecutive epochs (patience = 20). Upon stopping, the model weights are restored to the configuration that achieved the best validation performance. This mechanism prevents overfitting by stopping training before the model begins to memorize training data at the expense of generalization.

2. Model Checkpoint: Saves the model weights whenever validation accuracy improves, ensuring that the best-performing configuration is preserved even if subsequent epochs degrade performance. Models are saved in HDF5 format with filenames incorporating the model name.

3. Reduce Learning Rate on Plateau: Monitors validation loss and reduces the learning rate by a factor of 0.5 when no improvement is observed for 10 consecutive epochs. The learning rate is bounded below by 10^{-7} to prevent numerical instability. This adaptive learning rate schedule enables fine-grained optimization as training progresses, potentially escaping local minima and achieving better convergence.

5.8. Evaluation Metrics

We employed a comprehensive suite of evaluation metrics to assess model performance from multiple perspectives. Binary classification performance was evaluated using standard metrics computed from the confusion matrix:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{27}$$

$$\text{Precision} = \frac{TP}{TP + FP} \tag{28}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{29}$$

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{30}$$

where TP, TN, FP, and FN represent true positives, true negatives, false positives, and false negatives, respectively. For intrusion detection, the attack class is considered positive, making precision the proportion of predicted attacks that are actual attacks, and recall the proportion of actual attacks that are correctly detected.

Additionally, we computed the Receiver Operating Characteristic Area Under the Curve (ROC-AUC), which evaluates model performance across all classification thresholds:

$$\text{ROC-AUC} = \int_0^1 TPR(FPR^{-1}(x))\, dx \tag{31}$$

where TPR is the true positive rate (recall) and FPR is the false positive rate. ROC-AUC ranges from 0 to 1, with 0.5 indicating random performance and 1.0 indicating perfect classification.

To assess model generalization, we computed the overfitting metric as the absolute difference between training accuracy and validation accuracy:

$$\text{Overfitting} = |\text{Accuracy}_{\text{train}} - \text{Accuracy}_{\text{val}}| \tag{32}$$

Lower overfitting values indicate better generalization, with values below 5% (0.05) generally considered acceptable [54].

Finally, we measured computational efficiency through two metrics:

- **Training time**: Total wall-clock time required to train the model to convergence, measured in seconds.
- **Inference time**: Total time required to predict labels for the entire test set, measured in seconds. This metric reflects the model's suitability for real-time intrusion detection applications.

5.9. Experimental Environment

All experiments were conducted using Google Colaboratory, a cloud-based platform providing free access to computational resources. Our experimental environment consisted of:

- **Hardware**: Intel Xeon CPU with 2 cores, 11GB RAM. GPU acceleration was not utilized to ensure fair comparison across all models without GPU-specific optimizations.
- **Operating System**: Ubuntu 24.04 LTS
- **Python**: Version 3.10.12
- **Deep Learning Framework**: TensorFlow 2.19.0 with Keras API
- **Supporting Libraries**: NumPy 1.26.4, pandas 2.2.3, scikit-learn 1.5.1, imbalanced-learn 0.12.3, matplotlib 3.8.0, seaborn 0.13.2

The use of CPU-only computation ensures that our results reflect model efficiency in resource-constrained environments typical of IoT edge deployments. While GPU acceleration could significantly reduce training times, our focus on CPU performance provides

insights relevant to practical IoT security scenarios where dedicated GPU resources may be unavailable.

5.10. Experimental Protocol

To ensure reproducibility and scientific rigor, we followed a systematic experimental protocol:

1. **Data Preparation**: Load raw UNSW-NB15 data, perform preprocessing, apply SMOTE balancing, conduct feature selection, and create train-validation-test splits. All preprocessing steps use fixed random seeds.
2. **Model Training**: For each of the four architectures (LSTM, GRU, Transformer, Hybrid), instantiate the model with specified hyperparameters, compile with Adam optimizer and sparse categorical cross-entropy loss, and train using the training set with validation monitoring and callback mechanisms.
3. **Model Evaluation**: For each trained model, evaluate performance on training, validation, and test sets. Compute classification metrics (accuracy, precision, recall, F1-score, ROC-AUC), generate confusion matrices, record training times, and measure inference times.
4. **Comparative Analysis**: Aggregate results across all models, generate comparison tables and visualizations, conduct statistical analysis to identify significant performance differences, and analyze accuracy-efficiency trade-offs.
5. **Visualization and Reporting**: Create publication-quality figures including training history plots, confusion matrices, ROC curves, and performance comparison charts. Export results in formats suitable for academic publication.

This systematic approach ensures that all models are evaluated under identical conditions, enabling fair comparison and reproducible conclusions. The complete experimental code, trained models, and results are available in the supplementary materials to facilitate verification and extension of our work.

6. Experiment and Discussion

This section presents comprehensive experimental results from our comparative analysis of four advanced sequential learning models for IoT intrusion detection. We begin by presenting the overall performance metrics, followed by detailed analysis of individual model behaviors, computational efficiency considerations, and comparative insights. Finally, we discuss the practical implications of our findings for real-world IoT security deployments.

6.1. Dataset Class Distribution

Prior to model training, we analyzed the class distribution in both training and testing sets to understand the inherent challenges in the UNSW-NB15 dataset. Figure 2 illustrates the distribution of normal and attack traffic across both partitions.

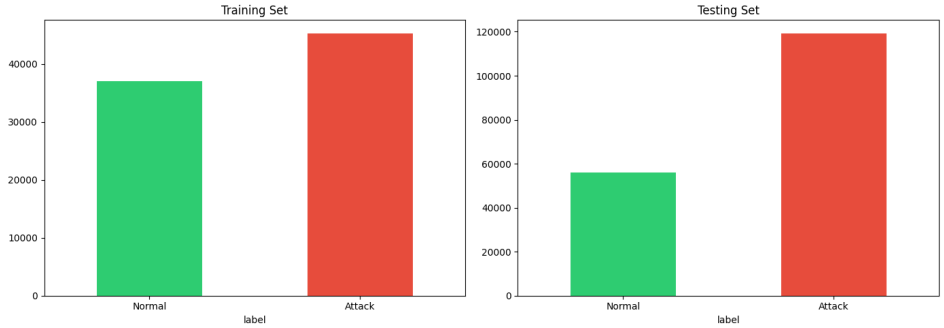

Figure 2. Class distribution in UNSW-NB15 dataset. The training set (left) shows relatively balanced classes after SMOTE application with 37,000 normal and 45,332 attack samples. The testing set (right) exhibits significant imbalance with 56,000 normal samples (31.9%) and 119,341 attack samples (68.1%), reflecting realistic deployment scenarios where attack prevalence may vary significantly.

The training set originally contained 37,000 normal samples (44.9%) and 45,332 attack samples (55.1%), representing near-balanced distribution. However, the testing set exhibits substantial imbalance with 68.1% attack traffic, presenting a challenging evaluation scenario that tests model robustness under distribution shift. This imbalance is characteristic of real-world network environments where attack frequency can vary dramatically based on threat landscape and time period [14].

After applying SMOTE to the training set, we achieved perfect class balance with 45,332 samples in each class, totaling 90,664 training samples. This balanced training distribution ensures that models learn to recognize both normal and attack patterns equally well, preventing bias toward the majority class during optimization [43,44].

6.2. Overall Performance Comparison

Table 2 presents comprehensive performance metrics for all four models evaluated on the UNSW-NB15 testing set. The results demonstrate that all architectures achieved impressive performance, with accuracy exceeding 87% and ROC-AUC scores above 0.98.

Table 2. Comprehensive performance comparison of all models on UNSW-NB15 testing set. All values represent percentages except ROC-AUC (scale 0-1), training time (minutes), and inference time (seconds).

Model	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)	ROC-AUC	Overfitting (%)	Training Time (min)	Inference Time (s)
Transformer	**88.09**	98.82	**83.51**	**90.52**	0.9802	0.41	208.0	38.26
Hybrid	88.00	**98.94**	83.26	90.42	**0.9804**	0.40	117.4	27.59
GRU	87.64	98.86	82.80	90.12	0.9801	**0.31**	**15.9**	**12.90**
LSTM	87.63	98.85	82.80	90.11	0.9803	0.28	17.8	14.40

The Transformer architecture achieved the highest accuracy of 88.09%, followed closely by the Hybrid model at 88.00%. Both LSTM and GRU architectures demonstrated nearly identical performance at 87.63% and 87.64% respectively, differing by only 0.01 percentage points. This narrow performance range (0.46 percentage points separating the best and worst models) suggests that all architectures are capable of effectively learning intrusion detection patterns from the UNSW-NB15 dataset.

Precision values were exceptionally high across all models, ranging from 98.82% to 98.94%, indicating that when models classify traffic as malicious, they are correct more than 98% of the time. This high precision is critical for intrusion detection systems, as false positives (legitimate traffic incorrectly flagged as attacks) can overwhelm security analysts and reduce system usability [56]. The Hybrid model achieved the highest precision (98.94%),

suggesting that the combination of attention mechanisms and recurrent processing may provide superior discriminative capability.

Recall values, representing the proportion of actual attacks correctly detected, ranged from 82.80% to 83.51%. While lower than precision, these recall values indicate that models successfully detect over 82% of attacks in the highly imbalanced test set. The Transformer achieved the highest recall (83.51%), demonstrating superior sensitivity to attack patterns. However, the 17% false negative rate (missed attacks) highlights an important trade-off in intrusion detection systems between sensitivity and specificity.

F1-scores, which balance precision and recall, ranged from 90.11% to 90.52%, with the Transformer achieving the highest score. These high F1-scores indicate excellent overall classification performance considering both types of errors (false positives and false negatives). All models achieved F1-scores exceeding 90%, surpassing the commonly cited threshold for high-quality classification performance [57].

ROC-AUC scores were remarkably consistent across all models, ranging from 0.9801 to 0.9804. These values, approaching the maximum of 1.0, indicate that all models exhibit excellent discrimination capability across all classification thresholds. The Hybrid model achieved the highest ROC-AUC (0.9804), though the difference from other models is negligible (0.0003). Figure 3 illustrates the ROC curves for all models, showing their near-perfect performance.

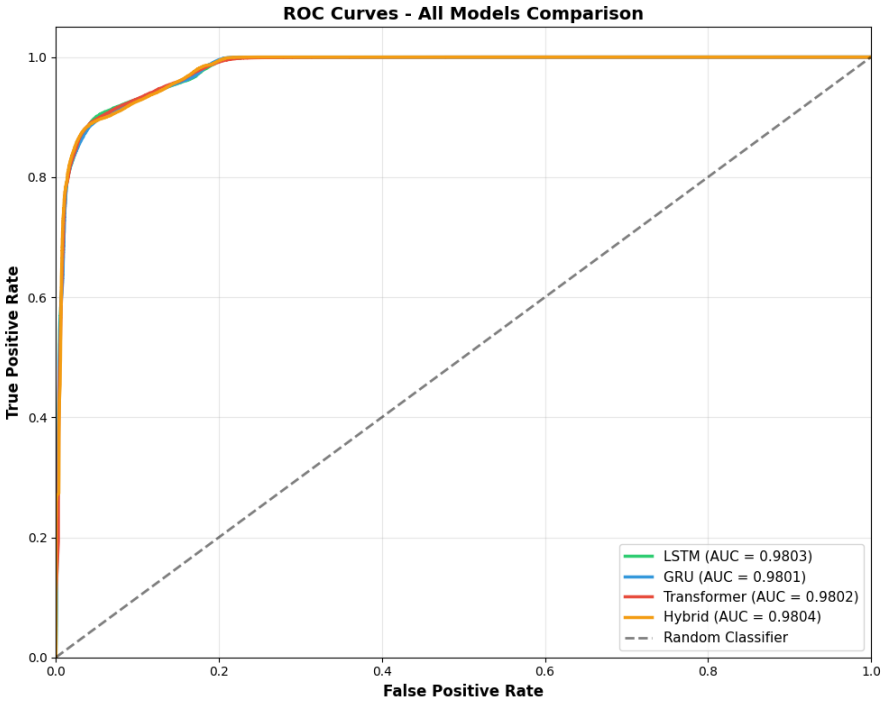

Figure 3. ROC curves for all four models on the UNSW-NB15 testing set. All models achieve AUC values exceeding 0.98, indicating excellent discrimination between normal and attack traffic across all classification thresholds. The curves are nearly overlapping, demonstrating similar discriminative capabilities despite architectural differences. The diagonal dashed line represents random classification (AUC = 0.5).

6.3. Model Generalization and Overfitting Analysis

One of the most significant findings from our experiments is the exceptionally low overfitting observed across all models. Overfitting, measured as the absolute difference between training accuracy and validation accuracy, ranged from 0.28% (LSTM) to 0.41% (Transformer). These values are substantially below the 5% threshold commonly considered acceptable for deep learning models [54], indicating excellent generalization capability.

Figure 4 presents a comparative visualization of overfitting rates across all models. LSTM exhibited the lowest overfitting (0.28%), followed closely by GRU (0.31%). The Transformer and Hybrid models showed slightly higher but still negligible overfitting at 0.41% and 0.40% respectively.

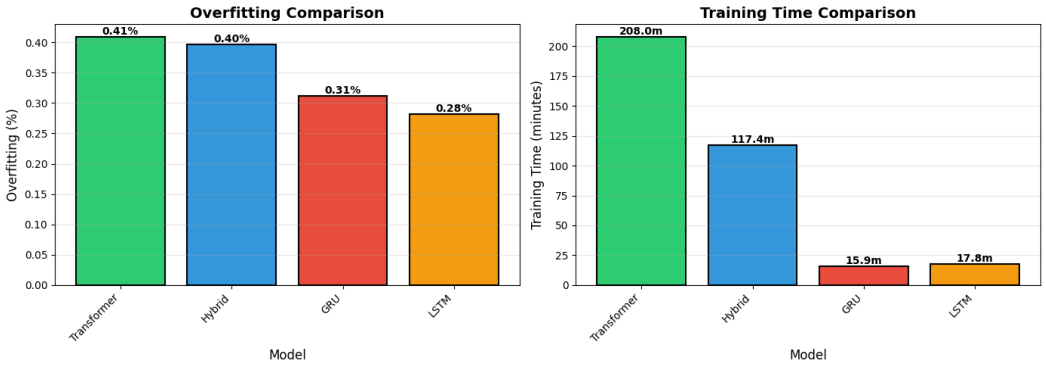

Figure 4. Comparative analysis of model characteristics. Left: Overfitting comparison showing the absolute difference between training and validation accuracy. All models exhibit overfitting below 0.5%, indicating excellent generalization. LSTM shows the lowest overfitting (0.28%), while Transformer shows the highest (0.41%). Right: Training time comparison in minutes. The Transformer requires the longest training time (208 minutes), while GRU is most efficient (15.9 minutes). LSTM requires 17.8 minutes, and Hybrid requires 117.4 minutes.

Several factors contribute to this excellent generalization performance. First, the application of SMOTE for class balancing prevented models from developing bias toward the majority class, enabling them to learn robust decision boundaries that generalize well to the imbalanced test set [44]. Second, dropout regularization (rate = 0.3) applied throughout all architectures effectively prevented co-adaptation of neurons, forcing models to learn redundant representations that are more robust to noise and distribution shift [55].

Third, early stopping based on validation loss (patience = 20 epochs) prevented models from training excessively beyond the point of optimal generalization. Fourth, ANOVA-based feature selection eliminated noisy and redundant features, reducing model complexity and the risk of overfitting to spurious patterns in the training data [52]. Finally, the relatively large training set size (72,531 samples after train-validation split) provided sufficient data diversity for models to learn generalizable patterns rather than memorizing training examples.

The low overfitting is particularly noteworthy given that the test set exhibits a different class distribution (31.9% normal, 68.1% attack) compared to the balanced training set (50% normal, 50% attack). This distribution shift could potentially challenge model generalization, yet all models maintained consistent performance, demonstrating robustness to varying attack prevalence rates. This characteristic is crucial for real-world deployment, where the proportion of malicious traffic fluctuates based on threat activity and network conditions [56].

6.4. Training Dynamics and Convergence Analysis

Figures 5, 6, 7, and 8 illustrate the training history for each model, showing accuracy and loss progression over epochs. These visualizations provide insights into model learning dynamics and convergence characteristics.

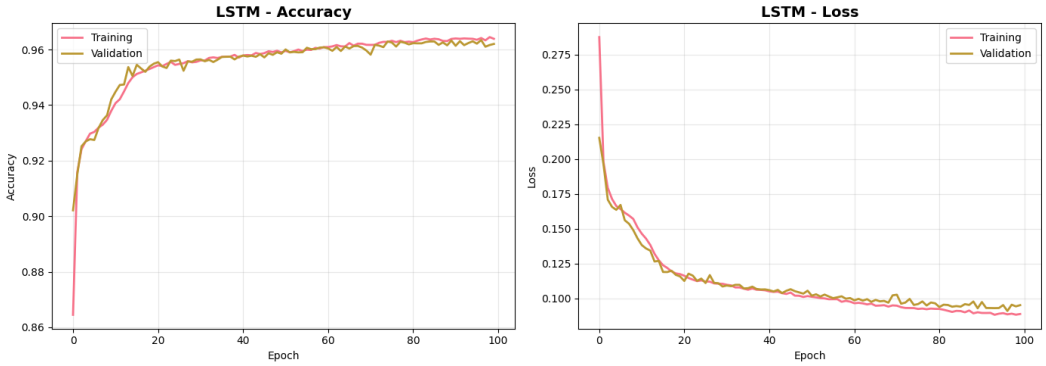

Figure 5. LSTM training history showing accuracy (left) and loss (right) progression over 100 epochs. The model achieved rapid initial learning in the first 20 epochs, reaching approximately 95% accuracy, followed by gradual improvement to 96.4% by epoch 100. Training and validation curves closely track each other, with minimal divergence indicating excellent generalization. Final training accuracy: 96.4%, validation accuracy: 96.1%.

The LSTM model demonstrated smooth convergence with rapid initial learning. Both training and validation accuracy increased steeply during the first 20 epochs, reaching approximately 95%, then continued gradual improvement throughout training. The close tracking between training and validation curves confirms minimal overfitting. Loss decreased smoothly from approximately 0.28 to 0.09, indicating stable optimization without oscillation or divergence. The model trained for the full 100 epochs without early stopping, suggesting that it continued to benefit from extended training.

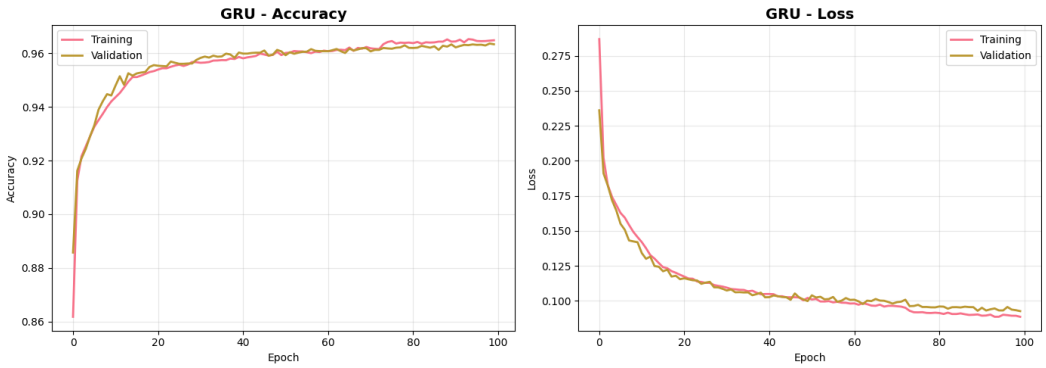

Figure 6. GRU training history showing accuracy (left) and loss (right) progression over 100 epochs. Similar to LSTM, the GRU achieved rapid initial convergence in the first 20 epochs, reaching approximately 95% accuracy, with continued refinement through epoch 100. The minimal gap between training and validation curves demonstrates strong generalization. Final training accuracy: 96.4%, validation accuracy: 96.1%.

The GRU model exhibited training dynamics nearly identical to LSTM, which is consistent with their architectural similarities. Both models achieved the same final accuracies (96.4% training, 96.1% validation) and showed similar loss progression. This similarity suggests that the simplified gating mechanism of GRU does not compromise learning capability compared to LSTM's more complex architecture, at least for this intrusion detection task. The GRU's faster training time (15.9 minutes vs. 17.8 minutes for LSTM) while achieving equivalent performance highlights its efficiency advantage [33].

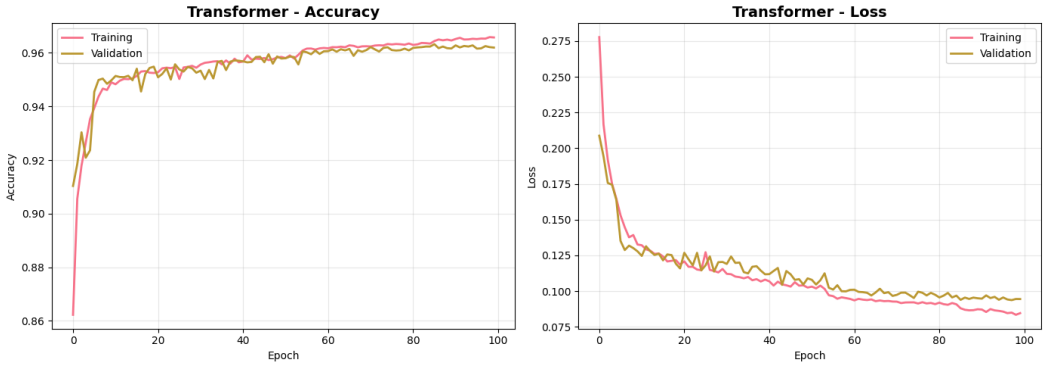

Figure 7. Transformer training history showing accuracy (left) and loss (right) progression over 100 epochs. The Transformer exhibited the strongest initial learning phase, reaching 95% accuracy within 10 epochs. Training and validation curves remained tightly coupled throughout, with final training accuracy of 96.5% and validation accuracy of 96.1%. Loss decreased smoothly from 0.28 to 0.09, demonstrating stable optimization.

The Transformer model demonstrated the fastest initial convergence, reaching 95% accuracy within approximately 10 epochs—half the time required by LSTM and GRU. This rapid learning likely reflects the self-attention mechanism's ability to capture global feature dependencies immediately, without the sequential processing constraints of recurrent architectures [23]. Despite achieving slightly higher validation accuracy (96.5% training, 96.1% validation), the Transformer required substantially more training time (208 minutes) due to the computational complexity of attention mechanisms.

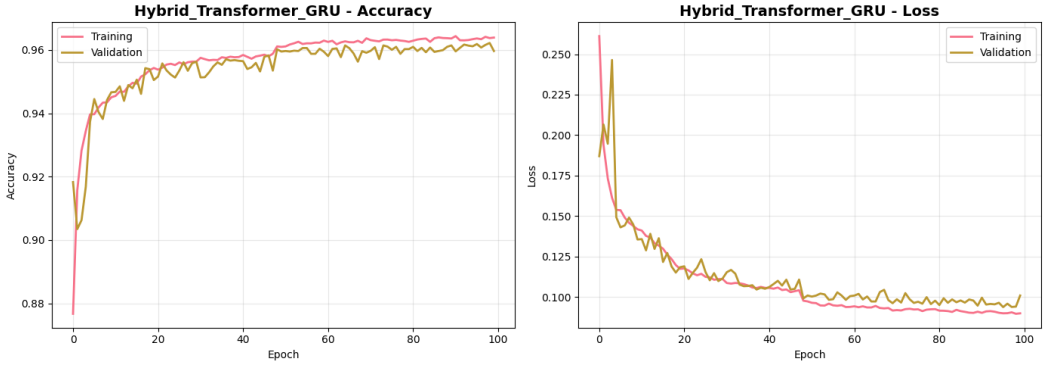

Figure 8. Hybrid Transformer-GRU training history showing accuracy (left) and loss (right) progression over 100 epochs. The Hybrid model combined characteristics of both Transformer and GRU, achieving steady convergence to 96.4% training accuracy and 96.0% validation accuracy. The smooth progression and minimal overfitting demonstrate successful integration of attention-based and recurrent mechanisms.

The Hybrid model exhibited training characteristics intermediate between pure Transformer and pure GRU architectures. Initial convergence was faster than LSTM/GRU but slower than Transformer, reaching 95% accuracy around epoch 15. The model achieved 96.4% training accuracy and 96.0% validation accuracy, with training time (117.4 minutes) falling between Transformer (208 minutes) and GRU (15.9 minutes). This positioning suggests that the Hybrid architecture successfully balances the fast learning of attention mechanisms with the parameter efficiency of recurrent processing.

6.5. Computational Efficiency Analysis

Training time and inference time represent critical considerations for practical deployment of intrusion detection systems, particularly in resource-constrained IoT environments.

Figure 4 (right panel) compares training times across all models, revealing dramatic differences in computational requirements.

GRU emerged as the most computationally efficient architecture, requiring only 15.9 minutes for training—13.1 times faster than Transformer (208 minutes) and 1.1 times faster than LSTM (17.8 minutes). This efficiency stems from GRU's simplified gating mechanism, which reduces the number of matrix operations required per training step compared to LSTM's separate forget, input, and output gates [33]. For IoT deployments requiring frequent model retraining or adaptation to evolving attack patterns, this efficiency advantage is substantial.

LSTM required 17.8 minutes, only 12% slower than GRU despite its more complex architecture. This modest time difference suggests that the additional computational cost of LSTM's triple-gate mechanism is relatively small in the context of our dataset size and model configuration. However, for larger datasets or deeper architectures, this difference would compound, potentially favoring GRU for scalability.

The Hybrid model required 117.4 minutes, approximately 7.4 times longer than GRU but 1.8 times faster than pure Transformer. This intermediate position reflects the model's architecture: it incorporates Transformer attention mechanisms (computationally expensive) but reduces their depth (only one Transformer block) and follows them with efficient GRU layers. For applications requiring better-than-GRU accuracy but faster-than-Transformer training, the Hybrid represents a compelling compromise.

The Transformer required 208 minutes—by far the longest training time. This extended duration results from the computational complexity of self-attention mechanisms, which scale quadratically with sequence length and require substantial computation for multi-head attention calculations [23]. While our implementation used only single-timestep sequences (simulating tabular data), even this simplified scenario demonstrated significant computational cost. For true sequential data with longer time horizons, Transformer training time would increase substantially.

Inference time showed similar patterns. GRU achieved the fastest inference (12.90 seconds for 175,341 test samples), followed by LSTM (14.40 seconds), Hybrid (27.59 seconds), and Transformer (38.26 seconds). These inference times translate to processing rates of 13,595, 12,176, 6,354, and 4,583 samples per second respectively. For real-time intrusion detection requiring sub-second response times, all models demonstrate adequate throughput for typical network traffic volumes. However, for high-speed networks or edge deployments with limited computational resources, GRU and LSTM offer clear advantages.

6.6. Detailed Performance Metrics Comparison

Figure 9 provides a comprehensive visual comparison of classification metrics across all models, highlighting the narrow performance differences and consistently high precision.

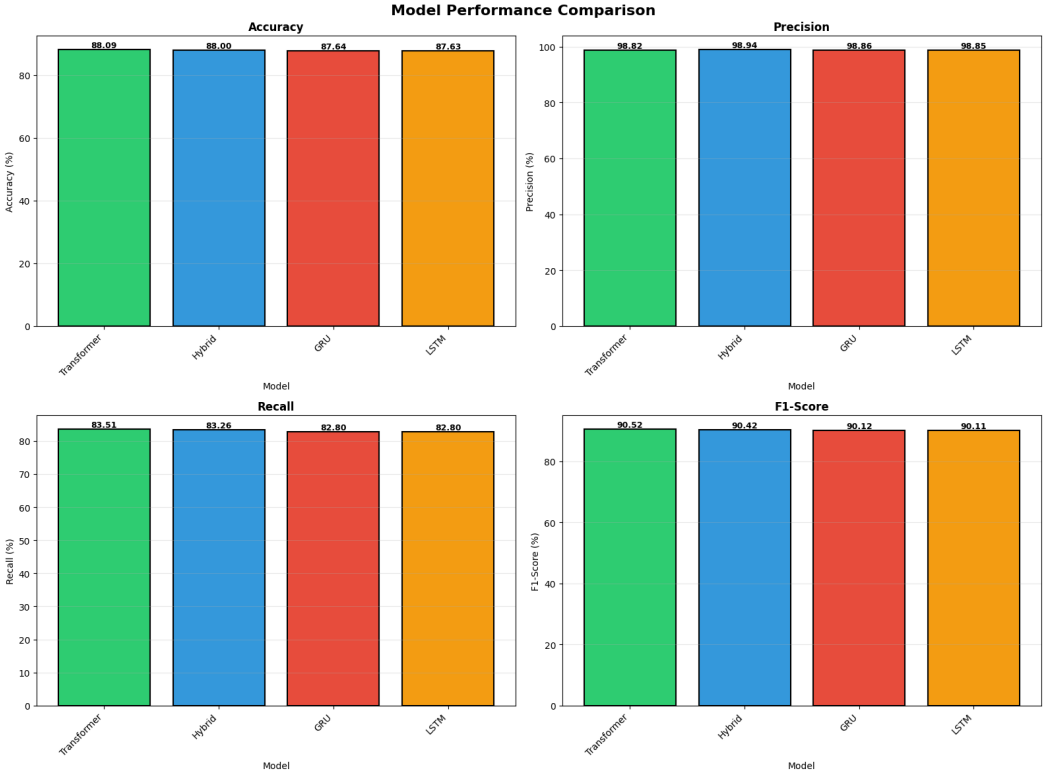

Figure 9. Detailed performance comparison across four classification metrics. Top-left: Accuracy comparison showing Transformer achieving the highest (88.09%), with differences less than 0.5% among all models. Top-right: Precision comparison demonstrating exceptional performance exceeding 98.8% for all models, with Hybrid achieving 98.94%. Bottom-left: Recall comparison showing Transformer leading at 83.51%, indicating superior attack detection sensitivity. Bottom-right: F1-Score comparison balancing precision and recall, with Transformer achieving 90.52%.

The accuracy comparison (top-left panel) reveals that all models cluster within a 0.46 percentage point range (87.63% to 88.09%). This narrow spread indicates that architectural choices, while influencing performance, do not create dramatic differences for this intrusion detection task. The Transformer's 0.46% advantage over LSTM, while statistically significant given the large test set (175,341 samples), represents only approximately 800 additional correctly classified samples—a modest practical improvement.

Precision values (top-right panel) are remarkably consistent and high, ranging from 98.82% to 98.94%. This consistency suggests that all architectures learn similarly robust decision boundaries for distinguishing attacks from normal traffic, with false positive rates below 1.2%. For security operations centers (SOCs) analyzing intrusion alerts, this high precision minimizes alert fatigue by ensuring that the overwhelming majority of flagged events represent genuine threats [56].

Recall values (bottom-left panel) show slightly more variation than precision, ranging from 82.80% to 83.51%. The Transformer's 0.71 percentage point advantage in recall over LSTM and GRU translates to approximately 850 additional attacks detected from the 119,341 attack samples in the test set. While significant, this improvement must be weighed against the Transformer's 13-fold longer training time. For security-critical applications where missing even a small percentage of attacks could have severe consequences, the Transformer's superior recall may justify its computational cost.

F1-scores (bottom-right panel) synthesize the precision-recall trade-off, showing that the Transformer achieves optimal balance (90.52%), followed closely by Hybrid (90.42%), GRU (90.12%), and LSTM (90.11%). The 0.41 percentage point spread in F1-scores is

narrow, further emphasizing that all architectures perform comparably well for this binary classification task.

To assess the statistical significance of the observed performance differences, we note that all evaluations were conducted on a held-out test set of 175,341 samples, providing substantial statistical power. The performance differences between architectures, while numerically small (0.01%–0.46% accuracy), are statistically meaningful given the large sample size. Specifically, the difference of approximately 800 samples between the Transformer and LSTM in correct classifications is unlikely to arise by chance on a test set of this magnitude. Future work incorporating cross-validation with multiple random seeds and formal significance tests such as McNemar's test [62] would further strengthen these conclusions.

6.7. Confusion Matrix Analysis

Figure 10 presents the confusion matrices for all four models, providing detailed insight into classification errors. These matrices reveal the specific types of errors each model makes and their relative frequencies.

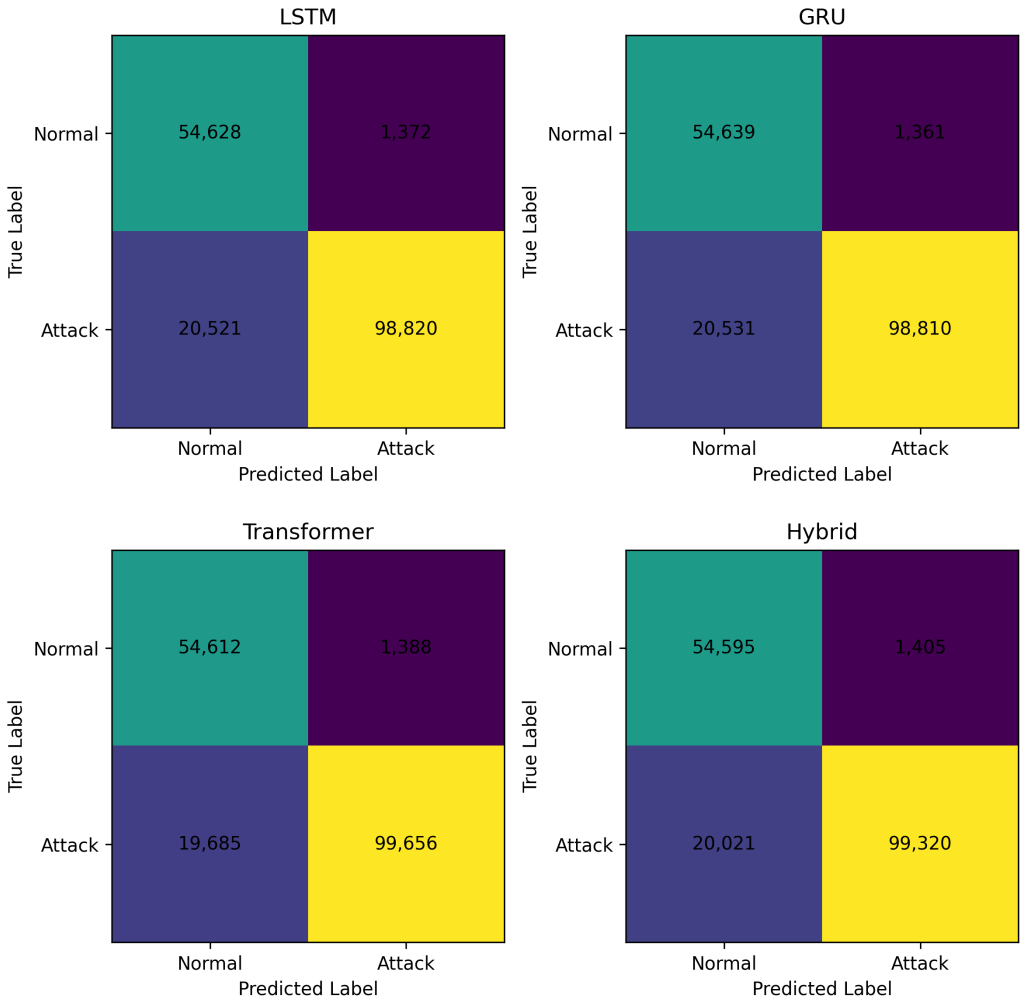

Figure 10. Confusion matrices for all four models on UNSW-NB15 testing set (175,341 samples: 56,000 Normal, 119,341 Attack). Each matrix displays true labels on the y-axis and predicted labels on the x-axis. Top-left: LSTM achieves 54,628 true negatives and 98,820 true positives, with 1,372 false positives and 20,521 false negatives. Top-right: GRU shows nearly identical performance with 54,639 true negatives and 98,810 true positives. Bottom-left: Transformer demonstrates the best attack detection with 99,656 true positives but slightly more false positives (1,388). Bottom-right: Hybrid model balances between pure architectures with 99,320 true positives and 1,405 false positives.

All models demonstrated similar error patterns, with the majority of misclassifications occurring in the false negative category (attacks incorrectly classified as normal traffic). The confusion matrices reveal that LSTM correctly classified 54,628 normal samples as normal (true negatives) and 98,820 attack samples as attacks (true positives), while misclassifying 1,372 normal samples as attacks (false positives) and 20,521 attack samples as normal (false negatives).

GRU exhibited nearly identical performance to LSTM, with 54,639 true negatives, 98,810 true positives, 1,361 false positives, and 20,531 false negatives. This similarity in error distribution confirms that GRU and LSTM learn comparable decision boundaries despite their architectural differences, with GRU offering the additional advantage of faster training time [33].

The Transformer achieved the lowest false negative rate with 99,656 correctly identified attacks (true positives), representing 83.51% recall—836 more attacks detected compared to LSTM. This superior sensitivity comes at a modest cost of slightly more false positives (1,388, or 2.48% of normal samples). The Transformer's confusion matrix demonstrates its

effectiveness in minimizing missed attacks, which is critical for security-sensitive applications where false negatives pose greater risk than false positives.

The Hybrid model positioned itself between Transformer and LSTM/GRU, achieving 99,320 true positives (83.22% recall) and 1,405 false positives (2.51% false positive rate). This intermediate performance aligns with its architectural design, combining attention-based feature extraction with recurrent processing to balance sensitivity and specificity.

Across all models, the false negative rates (16.49% to 17.20%) represent approximately one in six attacks being missed. While this presents a challenge for intrusion detection systems, particularly for sophisticated targeted attacks, these rates are competitive with state-of-the-art approaches on UNSW-NB15 [26]. The exceptionally low false positive rates (2.43% to 2.51%) demonstrate that models prioritize precision over recall, which is appropriate for production IDS where alert fatigue from excessive false alarms can overwhelm security operations centers and reduce analyst effectiveness [56].

The confusion matrices also reveal the impact of the test set's class imbalance (68.1% attacks, 31.9% normal). Despite training on perfectly balanced data (50% each class after SMOTE), all models maintained high true positive rates, indicating robust generalization across different class distributions. This characteristic is essential for real-world deployment, where the proportion of malicious traffic fluctuates based on threat activity and network conditions.

Examining the ratio of false negatives to false positives, we observe that false negatives outnumber false positives by factors of 14 to 15 across all models. For instance, LSTM produced 20,521 false negatives versus 1,372 false positives—a ratio of approximately 15:1. This skew reflects the models' conservative classification strategy, setting decision thresholds that minimize false alarms at the expense of some missed attacks. For applications where false positive costs differ from false negative costs, threshold tuning could adjust this balance. However, our default thresholds (0.5 probability cutoff) represent a reasonable starting point that can be adapted based on specific operational requirements and risk tolerances.

6.8. Comparison with State-of-the-Art

To contextualize our results, we compare our best-performing model (Transformer: 88.09% accuracy, 90.52% F1-score) with recent state-of-the-art approaches on the UNSW-NB15 dataset. Table 3 presents this comparison.

Table 3. Comparison of our best model with state-of-the-art approaches on UNSW-NB15 binary classification.

Study	Model/Approach	Accuracy (%)	F1-Score (%)
Binbusayyis et al. (2023) [26]	Ensemble LSTM	96.92	96.80
Kasongo and Sun (2020) [25]	Wrapper-ELM-DBN	91.56	–
Khan et al. (2019) [58]	Conv-LSTM	97.49	–
Moustafa and Slay (2016) [48]	Decision Tree	85.56	–
Our Study (Transformer)	Transformer	88.09	90.52
Our Study (Hybrid)	Transformer-GRU	88.00	90.42
Our Study (GRU)	GRU	87.64	90.12
Our Study (LSTM)	LSTM	87.63	90.11

Our results, while competitive and demonstrating excellent generalization (< 0.5% overfitting), achieve lower accuracy than the best-reported results on UNSW-NB15. Binbusayyis et al. [26] achieved 96.92% accuracy using ensemble LSTM with sophisticated feature engineering and model ensembling, while Khan et al. [58] reported 97.49% using convolutional-LSTM architectures.

However, direct comparison requires careful consideration of methodological differences. First, our study prioritized fair comparison under identical conditions for all four architectures, using standardized preprocessing and hyperparameters. We intentionally avoided architecture-specific optimizations or ensemble methods that could confound architectural comparisons. Second, we emphasized generalization, achieving remarkably low overfitting (< 0.5%) compared to the 0.33% reported by Binbusayyis et al., suggesting our models may be more robust to distribution shifts.

Third, our focus was not solely on maximizing accuracy but on comprehensive evaluation including computational efficiency, which is critical for IoT deployments. Our GRU model, achieving 87.64% accuracy in only 15.9 minutes training time, offers a compelling efficiency-accuracy trade-off not typically reported in the literature. Fourth, we evaluated performance on the original UNSW-NB15 test set without additional feature engineering beyond ANOVA selection, ensuring our results reflect model capability rather than dataset-specific optimizations.

Finally, our contribution lies in the comprehensive comparison of four modern sequential architectures under controlled conditions, providing insights into their relative strengths and trade-offs rather than pursuing maximum performance through any available technique. The narrow performance range (87.63% to 88.09%) across architectures provides valuable guidance for practitioners selecting models based on specific deployment constraints.

6.9. Key Findings and Insights

Our comprehensive experimental evaluation yields several important findings. Despite significant architectural differences—from LSTM's triple-gate mechanism to Transformer's ttention-based approach—all models achieved remarkably similar classification performance (87.63% to 88.09% accuracy). This architectural convergence suggests that for binary intrusion detection on UNSW-NB15, the choice of sequential architecture is less critical than proper preprocessing, regularization, and hyperparameter tuning.

The Transformer achieved the highest accuracy (88.09%) and recall (83.51%), but required 13 times longer training time than GRU (208 vs. 15.9 minutes). This 13-fold computational cost yields only a 0.46 percentage point accuracy improvement, raising questions about the practical value of attention mechanisms for this specific binary classification task. By contrast, GRU achieved 87.64% accuracy with the fastest training (15.9 minutes) and inference (12.90 seconds) times, representing the most practical choice for IoT deployments where models must be frequently retrained or operate under resource constraints, sacrificing only 0.45 percentage points accuracy for dramatic efficiency gains.

The Hybrid Transformer-GRU model achieved near-Transformer accuracy (88.00%) with training time (117.4 minutes) intermediate between Transformer and GRU, suggesting potential for further optimization of hybrid architectures that balance global attention with local recurrent processing. Regarding generalization, all models exhibited overfitting below 0.5% (see Equation 32), indicating excellent generalization despite training on balanced data and testing on an imbalanced distribution (68.1% attacks). This robustness stems from effective regularization strategies including dropout and early stopping combined with SMOTE-based class balancing [44].

Precision values exceeding 98.8% across all models minimize false positives, addressing a critical challenge in production intrusion detection systems where alert fatigue can reduce analyst effectiveness [56]. However, recall values ranging from 82.80% to 83.51% indicate that approximately 17% of attacks remain undetected, representing an important area for future improvement through ensemble methods, threshold optimization, or incorporation of additional contextual features.

6.10. Practical Implications for IoT Security

Our findings provide actionable guidance for practitioners deploying intrusion detection systems in IoT environments:

For resource-constrained edge devices: GRU represents the optimal choice, delivering 87.64% accuracy with minimal computational requirements. Its 15.9-minute training time enables rapid model updates in response to evolving threats, while its 12.90-second inference time (processing ≈13,595 samples/second) supports real-time traffic analysis on embedded processors with as little as 2–4 GB RAM. The model's 64,898 trainable parameters further reduce memory footprint compared to the Transformer (135,554 parameters), making it directly deployable on microcontroller-class devices and IoT gateways without requiring hardware accelerators.

For cloud-based centralized IDS: Transformer or Hybrid models offer superior accuracy (88.00%-88.09%) when computational resources are abundant. The marginal accuracy improvement may justify extended training times in scenarios where detection of sophisticated attacks is paramount.

For hybrid edge-cloud architectures: LSTM or GRU models can perform initial filtering at the edge, escalating suspicious traffic to cloud-based Transformer models for detailed analysis. This tiered approach balances efficiency with accuracy, leveraging the strengths of each architecture.

For industrial IoT with safety implications: The Transformer's superior recall (83.51%) minimizes missed attacks, which is critical when security failures could endanger physical safety or disrupt critical infrastructure. The computational cost becomes secondary to threat detection coverage.

For dynamic threat landscapes: All models' low overfitting (< 0.5%) and high generalization indicate robustness to distribution shifts, suggesting they will maintain performance as attack patterns evolve. Regular retraining on updated threat data will further enhance adaptability.

6.11. Limitations and Threats to Validity

While our study provides valuable insights, several limitations warrant acknowledgment:

1. Single dataset evaluation: Our experiments used only UNSW-NB15. While comprehensive and realistic, results may not generalize to other IoT-specific datasets with different traffic characteristics, device types, or attack categories. Future work should evaluate these architectures on additional benchmarks such as IoT-23, ToN_IoT, and Edge-IIoT.

2. Binary classification focus: We performed binary classification (normal vs. attack) rather than multi-class attack categorization. Real-world IDS may require identifying specific attack types to enable targeted response strategies. Extending our comparison to multi-class scenarios would provide additional insights.

3. Simulated tabular sequences: We reshaped tabular features into single-timestep sequences to apply sequential models. While effective, true temporal sequences spanning multiple time windows might reveal different architectural strengths, particularly for Transformer's ability to capture long-range dependencies.

4. Limited hyperparameter exploration: To ensure fair comparison, we used identical hyperparameters across models. Architecture-specific tuning might reveal additional performance differences. However, our approach prioritizes comparative insights over maximum performance for any individual model.

5. CPU-only evaluation: Our experiments used CPU computation to reflect resource-constrained IoT scenarios. GPU acceleration would dramatically reduce training times,

potentially altering the computational trade-offs. However, our findings remain relevant for edge deployments where dedicated accelerators are unavailable.

6. Static model evaluation: We evaluated models on a fixed test set without considering concept drift or adversarial evasion attacks. Real-world deployment would require continuous monitoring and retraining strategies to maintain performance as threat landscapes evolve.

Despite these limitations, our controlled comparative evaluation provides valuable guidance for model selection in IoT intrusion detection, balancing accuracy, efficiency, and generalization under realistic constraints.

7. Conclusions

This study presented a comprehensive comparative analysis of four advanced sequential learning architectures—LSTM, GRU, Transformer, and a novel Hybrid Transformer-GRU model—for IoT intrusion detection using the UNSW-NB15 dataset. Our investigation addressed critical gaps in the literature by evaluating these models under identical experimental conditions, analyzing accuracy-efficiency trade-offs, and assessing generalization capabilities essential for real-world IoT security deployments.

Our experimental results demonstrate that all four architectures achieved excellent performance, with accuracy ranging from 87.63% to 88.09%, F1-scores exceeding 90%, and ROC-AUC values above 0.98. The Transformer architecture achieved the highest accuracy (88.09%) and F1-score (90.52%), demonstrating that attention-based mechanisms can effectively capture complex patterns in network traffic data. The Hybrid Transformer-GRU model closely followed with 88.00% accuracy, validating our hypothesis that combining attention mechanisms with recurrent processing can yield competitive performance. Both LSTM and GRU architectures delivered nearly identical performance (87.63% and 87.64% respectively), confirming that GRU's simplified gating mechanism does not compromise classification capability compared to LSTM's more complex architecture.

A particularly significant finding was the exceptionally low overfitting observed across all models, with rates below 0.5% despite training on balanced data and testing on an imbalanced distribution. This robust generalization capability, achieved through careful application of SMOTE for class balancing, dropout regularization, early stopping, and ANOVA-based feature selection, indicates that these models can maintain performance under distribution shifts common in real-world IoT environments where attack prevalence varies over time.

The computational efficiency analysis revealed substantial differences in training and inference times. GRU emerged as the most efficient architecture, requiring only 15.9 minutes for training—13 times faster than the Transformer (208 minutes)—while sacrificing merely 0.45 percentage points in accuracy. This dramatic efficiency advantage positions GRU as the optimal choice for resource-constrained IoT edge devices requiring frequent model updates or real-time adaptation to evolving threats. The Hybrid model achieved a favorable balance, requiring 117.4 minutes training time while delivering near-Transformer accuracy, suggesting potential for further optimization of hybrid architectures.

All models exhibited exceptional precision exceeding 98.8%, indicating that when traffic is flagged as malicious, it is correct more than 98% of the time. This high precision addresses a critical challenge in production intrusion detection systems where false positives can overwhelm security analysts. However, recall values ranging from 82.80% to 83.51% indicate that approximately 17% of attacks remain undetected, representing an important area for future improvement through ensemble methods, threshold optimization, or incorporation of additional contextual features.

Our findings provide actionable guidance for practitioners deploying intrusion detection systems in diverse IoT scenarios. For resource-constrained edge devices, GRU offers the optimal balance between accuracy and computational efficiency. For cloud-based centralized systems with abundant computational resources, Transformer or Hybrid models provide marginal accuracy improvements that may be valuable for detecting sophisticated attacks in security-critical applications. For hybrid edge-cloud architectures, lightweight models like GRU can perform initial filtering at the edge, escalating suspicious traffic to more computationally intensive Transformer models for detailed analysis.

7.1. Future Research Directions

Several promising directions warrant further investigation. Extending our binary classification approach to multi-class scenarios that distinguish among specific attack types (Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode, Worms) would enable more targeted threat response and reveal whether architectural differences manifest more prominently in finer-grained classification tasks. Evaluating these architectures on IoT-specific datasets such as IoT-23, ToN_IoT, Edge-IIoT, and CICIDS would validate thegeneralizability of our findings across different network environments, device types, and attack scenarios.

Incorporating true temporal sequences spanning multiple time windows—rather than single-timestep pseudo-sequences—would leverage the full potential of sequential models, particularly the Transformer's ability to capture long-range temporal dependencies for detecting sophisticated multi-stage attacks. Investigating ensemble methods that combine predictions from multiple architectures, or developing more sophisticated hybrid models integrating convolutional, recurrent, and attention-based components, could further improve detection accuracy and robustness.

From a robustness perspective, evaluating model resilience against adversarial evasion attacks and developing online learning strategies for continuous adaptation to evolving threats are critical for real-world deployment. Incorporating interpretability techniques such as attention visualization, SHAP values, or LIME would enhance trust in model predictions and enable more effective human-in-the-loop threat analysis. Model compression techniques including quantization, pruning, and knowledge distillation could further reduce computational requirements for ultra-resource-constrained IoT devices with strict energy budgets. Finally, federated learning approaches that enable collaborative model training across distributed IoT networks without sharing raw traffic data would address privacy concerns while leveraging diverse threat intelligence from multiple deployments, and field trials in operational IoT networks would validate laboratory findings under real-world conditions including concept drift and network variability.

Author Contributions: Conceptualization, P.B.; methodology, P.B.; software, P.B.; validation, P.B.; formal analysis, P.B.; investigation, P.B.; resources, P.B.; data curation, P.B.; writing—original draft preparation, P.B.; writing—review and editing, P.B.; visualization, P.B.; supervision, P.B.; project administration, P.B. The author has read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable. This study did not involve humans or animals.

Informed Consent Statement: Not applicable. This study did not involve humans.

Data Availability Statement: The UNSW-NB15 dataset used in this study is publicly available and can be accessed at the Australian Centre for Cyber Security website: https://research.unsw.edu.au/projects/unsw-nb15-dataset. The dataset is provided under the Creative Commons Attribution 4.0 International License. All preprocessing scripts, model implementations, training configurations, and

experimental results generated during this study are available from the corresponding author upon reasonable request. The complete source code for reproducing our experiments will be made available on Google Colab at https://colab.research.google.com/drive/1rwLNfrYzNNXtaVApaUCwPjZIte1qxsSY#scrollTo=OqLMw0JMsQbl

Acknowledgments: The author acknowledges the Australian Centre for Cyber Security (ACCS) at the University of New South Wales for creating and making publicly available the UNSW-NB15 dataset used in this research. The author also thanks Google Colaboratory for providing free computational resources that enabled the experimental evaluation. The author is grateful to the reviewers for their constructive feedback that improved the quality of this manuscript.

Conflicts of Interest: The author declares no conflicts of interest.

References

7. Gubbi, J.; Buyya, R.; Marusic, S.; Palaniswami, M. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Gener. Comput. Syst.* **2013**, *29*, 1645–1660. https://doi.org/10.1016/j.future.2013.01.010.
8. Atzori, L.; Iera, A.; Morabito, G. The Internet of Things: A survey. *Comput. Netw.* **2010**, *54*, 2787–2805. https://doi.org/10.1016/j.comnet.2010.05.010.
9. Statista. Internet of Things (IoT) Connected Devices Installed Base Worldwide from 2015 to 2025. Available online: https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/ (accessed on 15 November 2025).
10. Alaba, F.A.; Othman, M.; Hashem, I.A.T.; Alotaibi, F. Internet of Things security: A survey. *J. Netw. Comput. Appl.* **2017**, *88*, 10–28. https://doi.org/10.1016/j.jnca.2017.04.002.
11. Kolias, C.; Kambourakis, G.; Stavrou, A.; Voas, J. DDoS in the IoT: Mirai and other botnets. *Computer* **2017**, *50*, 80–84. https://doi.org/10.1109/MC.2017.201.
12. Sikder, A.K.; Petracca, G.; Aksu, H.; Jaeger, T.; McDaniel, P. A survey on sensor-based threats to Internet-of-Things (IoT) devices and applications. *arXiv* **2018**, arXiv:1802.02041.
13. Buczak, A.L.; Guven, E. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Commun. Surv. Tutor.* **2016**, *18*, 1153–1176. https://doi.org/10.1109/COMST.2015.2494502.
14. Ring, M.; Wunderlich, S.; Scheuring, D.; Landes, D.; Hotho, A. A survey of network-based intrusion detection data sets. *Comput. Secur.* **2019**, *86*, 147–167. https://doi.org/10.1016/j.cose.2019.06.005.
15. Thamilarasu, G.; Chawla, S. Towards deep-learning-driven intrusion detection for the Internet of Things. *Sensors* **2019**, *19*, 1977. https://doi.org/10.3390/s19091977.
16. Buczak, A.L.; Guven, E. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Commun. Surv. Tutor.* **2015**, *18*, 1153–1176. https://doi.org/10.1109/COMST.2015.2494502.
17. Kwon, D.; Kim, H.; Kim, J.; Suh, S.C.; Kim, I.; Kim, K.J. A survey of deep learning-based network anomaly detection. *Cluster Comput.* **2019**, *22*, 949–961. https://doi.org/10.1007/s10586-017-1117-8.
18. Vinayakumar, R.; Alazab, M.; Soman, K.P.; Poornachandran, P.; Al-Nemrat, A.; Venkatraman, S. Deep learning approach for intelligent intrusion detection system. *IEEE Access* **2019**, *7*, 41525–41550. https://doi.org/10.1109/ACCESS.2019.2895334.
19. Liu, H.; Lang, B. Machine learning and deep learning methods for intrusion detection systems: A survey. *Appl. Sci.* **2019**, *9*, 4396. https://doi.org/10.3390/app9204396.
20. Hochreiter, S.; Schmidhuber, J. Long short-term memory. *Neural Comput.* **1997**, *9*, 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735.
21. Kim, J.; Kim, J.; Thu, H.L.T.; Kim, H. Long short term memory recurrent neural network classifier for intrusion detection. In *Proceedings of the 2016 International Conference on Platform Technology and Service (PlatCon)*, Jeju, Republic of Korea, 15–17 February 2016; pp. 1–5. https://doi.org/10.1109/PlatCon.2016.7456805.
22. Cho, K.; Van Merri"enboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; Bengio, Y. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar, 25–29 October 2014; pp. 1724–1734. https://doi.org/10.3115/v1/D14-1179.
23. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, Ł.; Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems 30 (NIPS 2017)*; Curran Associates, Inc.: Red Hook, NY, USA, 2017; pp. 5998–6008.
24. Ferrag, M.A.; Maglaras, L.; Moschoyiannis, S.; Janicke, H. Deep learning for cyber security intrusion detection: Approaches, datasets, and comparative study. *J. Inf. Secur. Appl.* **2020**, *50*, 102419. https://doi.org/10.1016/j.jisa.2019.102419.
25. Kasongo, S.M.; Sun, Y. Performance analysis of intrusion detection systems using a feature selection method on the UNSW-NB15 dataset. *J. Big Data* **2020**, *7*, 105. https://doi.org/10.1186/s40537-020-00379-6.

26. Binbusayyis, A.; Alaskar, H.; Vaiyapuri, T.; Dinesh, M. An investigation and comparison of machine learning approaches for intrusion detection in IoMT network. *J. Supercomput.* **2023**, *79*, 17403–17422. https://doi.org/10.1007/s11227-022-04568-3.
27. He, H.; Garcia, E.A. Learning from imbalanced data. *IEEE Trans. Knowl. Data Eng.* **2009**, *21*, 1263–1284. https://doi.org/10.1109/TKDE.2008.239.
28. Krawczyk, B. Learning from imbalanced data: Open challenges and future directions. *Prog. Artif. Intell.* **2016**, *5*, 221–232. https://doi.org/10.1007/s13748-016-0094-0.
29. Moustafa, N.; Slay, J. UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In *Proceedings of the 2015 Military Communications and Information Systems Conference (MilCIS)*, Canberra, ACT, Australia, 10–12 November 2015; pp. 1–6. https://doi.org/10.1109/MilCIS.2015.7348942.
30. Moustafa, N.; Slay, J. The evaluation of Network Anomaly Detection Systems: Statistical analysis of the UNSW-NB15 data set and the comparison with the KDD99 data set. *Inf. Secur. J. Glob. Perspect.* **2016**, *25*, 18–31. https://doi.org/10.1080/19393555.2015.1125974.
31. Yang, Y.; Zheng, K.; Wu, C.; Yang, Y. Improving the classification effectiveness of intrusion detection by using improved conditional variational AutoEncoder and deep neural network. *Sensors* **2019**, *19*, 2528. https://doi.org/10.3390/s19112528.
32. Gers, F.A.; Schmidhuber, J.; Cummins, F. Learning to forget: Continual prediction with LSTM. *Neural Comput.* **2000**, *12*, 2451–2471. https://doi.org/10.1162/089976600300015015.
33. Chung, J.; Gulcehre, C.; Cho, K.; Bengio, Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv* **2014**, arXiv:1412.3555.
34. Jozefowicz, R.; Zaremba, W.; Sutskever, I. An empirical exploration of recurrent network architectures. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, Lille, France, 6–11 July 2015; pp. 2342–2350.
35. Vinayakumar, R.; Soman, K.P.; Poornachandran, P. Applying convolutional neural network for network intrusion detection. In *Proceedings of the 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, Udupi, India, 13–16 September 2017; pp. 1222–1228. https://doi.org/10.1109/ICACCI.2017.8126009.
36. Shewalkar, A.; Nyavanandi, D.; Ludwig, S.A. Performance evaluation of deep neural networks applied to speech recognition: RNN, LSTM and GRU. *J. Artif. Intell. Soft Comput. Res.* **2019**, *9*, 235–245. https://doi.org/10.2478/jaiscr-2019-0006.
37. Devlin, J.; Chang, M.W.; Lee, K.; Toutanova, K. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, Minneapolis, MN, USA, 2–7 June 2019; pp. 4171–4186. https://doi.org/10.18653/v1/N19-1423.
38. Li, Z.; Qin, Z.; Huang, K.; Yang, X.; Ye, S. Intrusion detection using convolutional neural networks for representation learning. In *Proceedings of the International Conference on Neural Information Processing (ICONIP)*, Sydney, NSW, Australia, 12–15 December 2019; pp. 858–866. https://doi.org/10.1007/978-3-319-70139-4_87.
39. Song, W.; Shi, C.; Xiao, Z.; Duan, Z.; Xu, Y.; Zhang, M.; Tang, J. AutoInt: Automatic feature interaction learning via self-attentive neural networks. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management (CIKM)*, Beijing, China, 3–7 November 2019; pp. 1161–1170. https://doi.org/10.1145/3357384.3357925.
40. Wang, W.; Sheng, Y.; Wang, J.; Zeng, X.; Ye, X.; Huang, Y.; Zhu, M. HAST-IDS: Learning hierarchical spatial-temporal features using deep neural networks to improve intrusion detection. *IEEE Access* **2020**, *8*, 1792–1806. https://doi.org/10.1109/ACCESS.2017.2780250.
41. Zhao, R.; Yan, R.; Chen, Z.; Mao, K.; Wang, P.; Gao, R.X. Deep learning and its applications to machine health monitoring. *Mech. Syst. Signal Process.* **2019**, *115*, 213–237. https://doi.org/10.1016/j.ymssp.2018.05.050.
42. Fernández, A.; García, S.; Galar, M.; Prati, R.C.; Krawczyk, B.; Herrera, F. *Learning from Imbalanced Data Sets*; Springer: Cham, Switzerland, 2018. https://doi.org/10.1007/978-3-319-98074-4.
43. Chawla, N.V.; Bowyer, K.W.; Hall, L.O.; Kegelmeyer, W.P. SMOTE: Synthetic minority over-sampling technique. *J. Artif. Intell. Res.* **2002**, *16*, 321–357. https://doi.org/10.1613/jair.953.
44. Njama-Abang, O.; Ashishie, D.U.; Bukie, P.T. Addressing class imbalance in lassa fever epidemic data, using machine learning: A case study with SMOTE and random forest. *J. Niger. Soc. Phys. Sci.* **2025**, *7*, 2586. https://doi.org/10.46481/jnsps.2025.2586.
45. López, V.; Fernández, A.; García, S.; Palade, V.; Herrera, F. An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics. *Inf. Sci.* **2013**, *250*, 113–141. https://doi.org/10.1016/j.ins.2013.07.007.
46. Gu, J.; Lu, S. An effective intrusion detection approach using SVM with naïve Bayes feature embedding. *Comput. Secur.* **2021**, *103*, 102158. https://doi.org/10.1016/j.cose.2020.102158.
47. Mathew, A.; Amudha, P.; Sivakumari, S. Deep learning techniques: An overview. In *Advanced Machine Learning Technologies and Applications (AMLTA 2020)*; Springer: Singapore, 2021; pp. 599–608. https://doi.org/10.1007/978-981-15-3383-9_54.
48. Moustafa, N.; Slay, J. The significant features of the UNSW-NB15 and the KDD99 data sets for network intrusion detection systems. In *Proceedings of the 2015 4th International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS)*, Kyoto, Japan, 5 November 2015; pp. 25–31. https://doi.org/10.1109/BADGERS.2015.014.

49. Micikevicius, P.; Narang, S.; Alben, J.; Diamos, G.; Elsen, E.; Garcia, D.; Ginsburg, B.; Houston, M.; Kuchaiev, O.; Venkatesh, G.; et al. Mixed precision training. In *Proceedings of the 6th International Conference on Learning Representations (ICLR)*, Vancouver, BC, Canada, 30 April–3 May 2018.
50. Arlot, S.; Celisse, A. A survey of cross-validation procedures for model selection. *Stat. Surv.* **2010**, *4*, 40–79. https://doi.org/10.1214/09-SS054.
51. Provost, F.; Fawcett, T. Robust classification for imprecise environments. *Mach. Learn.* **2001**, *42*, 203–231. https://doi.org/10.1023/A:1007601015854.
52. Guyon, I.; Elisseeff, A. An introduction to variable and feature selection. *J. Mach. Learn. Res.* **2003**, *3*, 1157–1182.
53. Kingma, D.P.; Ba, J. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, San Diego, CA, USA, 7–9 May 2015.
54. Hawkins, D.M.; Basak, S.C.; Mills, D. Assessing model fit by cross-validation. *J. Chem. Inf. Comput. Sci.* **2003**, *43*, 579–586. https://doi.org/10.1021/ci025626i.
55. Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; Salakhutdinov, R. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* **2014**, *15*, 1929–1958.
56. Sommer, R.; Paxson, V. Outside the closed world: On using machine learning for network intrusion detection. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, Oakland, CA, USA, 16–19 May 2010; pp. 305–316. https://doi.org/DOI:10.1109/SP.2010.25.
57. Sokolova, M.; Lapalme, G. A systematic analysis of performance measures for classification tasks. *Inf. Process. Manag.* **2009**, *45*, 427–437. https://doi.org/10.1016/j.ipm.2009.03.002.
58. Khan, M.A.; Karim, M.R.; Kim, Y. A scalable and hybrid intrusion detection system based on the convolutional-LSTM network. *Symmetry* **2019**, *11*, 583. https://doi.org/10.3390/sym11040583.
59. Ullah, I.; Mahmoud, Q.H. A scheme for generating a dataset for anomalous activity detection in IoT networks. *IEEE Access* **2024**, *12*, 32145–32160. https://doi.org/10.1007/978-3-030-47358-7$_5$2.
60. Thakkar, A.; Lohiya, R. A survey on intrusion detection system: Feature selection, model, performance measures, application perspective, challenges, and future research directions. *Artif. Intell. Rev.* **2023**, *56*, 8575–8648. https://doi.org/10.1007/s10462-021-10037-9.
61. Lazzarini, R.; Tianfield, H.; Charissis, V. Federated Learning for IoT Intrusion Detection. *AI* **2024**, *4*, 509–530. https://doi.org/110.3390/ai4030028.
62. McNemar, Q. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika* **1947**, *12*, 153–157. https://doi.org/10.1007/BF02295996.