TRIDENT: A Temporal Reinforcement Intrusion Detection Engine for Network Traffic

Aashitha Emilin T T
Department of Computer Science and IT
School of Computing
Amrita Vishwa Vidyapeetham
Kochi, India

Snehanjali Krishna
Department of Computer Science and IT
School of Computing
Amrita Vishwa Vidyapeetham
Kochi, India

Dr. Sangeetha J
Department of Computer Science and IT
School of Computing
Amrita Vishwa Vidyapeetham
Kochi, India

***Abstract*—We present TRIDENT (Temporal Reinforcement Intrusion Detection Engine for Network Traffic), a deep reinforcement learning–based intrusion detection system designed to detect DDoS attacks in dynamic network environments. TRIDENT employs a two-stage feature selection pipeline—Random Forest pre-filtering followed by TCN-based Recursive Feature Elimination (RFE)—to reduce 80 engineered features to 20 temporally discriminative features while suppressing noise. These features are processed by a hybrid CNN-MLP architecture embedded within a Deep Q-Network (DQN) framework, enabling adaptive, policy-driven classification of network traffic as benign or malicious. Detection reliability is enhanced through an asymmetric reward function that penalizes false negatives more heavily than false positives, prioritizing security over throughput. Evaluated on the CIC-DDoS2019 dataset using stratified 70/15/15 train/validation/test splits and 5-fold cross-validation with strict data leakage prevention, TRIDENT achieves 99.86% accuracy, 99.72% precision, 99.91% recall, and an AUC of 0.997, outperforming CNN, CNN-LSTM, CNN-BiLSTM, Transformer-based, and GNN-based IDS baselines under identical experimental conditions.**

***Index Terms*—Distributed Denial-of-Service (DDoS), Deep Reinforcement Learning (DRL), Deep Q-Network (DQN), Temporal Convolutional Networks (TCN), Feature Engineering, Network Intrusion Detection, CIC-DDoS2019, Adaptive Cybersecurity.**

I. Introduction

Cybersecurity threats have increased substantially in both scale and complexity due to the rapid expansion of cloud computing, Internet of Things (IoT) devices, and large-volume network infrastructures. Among these, Distributed Denial-of-Service (DDoS) attacks remain a critical threat, capable of saturating network resources while closely mimicking legitimate traffic characteristics, making anomaly discrimination particularly challenging [1], [7]. Traditional machine learning–based intrusion detection systems (IDS) are ill-suited to capture dynamic attack behaviors [14], a limitation compounded by the high dimensionality of modern network data and the temporal dependencies between successive packet flows.

Recent advances in deep learning and reinforcement learning have introduced more adaptive detection approaches [2], [3]. However, several existing methods suffer from one or more of the following weaknesses: omission of temporal structure across inter-flow sequences [10], [12], high sensitivity to redundant or noisy features [14], or static policy functions that cannot adapt as attack distributions evolve [4]. Furthermore, post-hoc explainability methods [19] remain computationally prohibitive for real-time deployment.

To address these limitations, we propose TRIDENT (Temporal Reinforcement Intrusion Detection Engine for Network Traffic), a hybrid architecture that integrates dual-stage feature selection, temporal convolutional learning, and deep reinforcement learning into a unified, end-to-end intrusion detection pipeline. TRIDENT's contributions are: (1) a two-stage feature selection mechanism combining Random Forest pre-filtering with TCN-based RFE to isolate 20 temporally discriminative features from an 80-dimensional engineered feature space; (2) a CNN-MLP hybrid network embedded within a DQN framework for policy-driven binary classification; and (3) a security-aware asymmetric reward function that imposes greater penalties on false negatives than false positives.

II. Related Work

Network intrusion detection systems (NIDS) have evolved considerably, from static rule-based firewalls to sophisticated deep learning architectures. Early work relied on supervised learning models such as Support Vector Machines (SVMs) and Decision Trees for traffic classification. While effective on low-dimensional benchmarks, these approaches struggle with the high-dimensional and volatile nature of modern network datasets. As noted by Sharafaldin et al. [7], the introduction of comprehensive benchmarks such as CIC-DDoS2019 exposed the inability of traditional machine learning methods to capture the complex, nonlinear patterns of modern exploitation-based DDoS attacks.

Convolutional Neural Networks (CNNs) have been widely adopted in IDS due to their strength in hierarchical feature extraction and automatic pattern learning. Kim et al. [10] proposed a hybrid intrusion detection approach that integrates anomaly-based and misuse-based detection, demonstrating the effectiveness of combining complementary classification signals for more robust network traffic analysis. Wang et al. [17] further demonstrated DDoS detection using a CNN-based deep learning framework, achieving superior accuracy over traditional classifiers by modeling hierarchical spatial-temporal features. Li et al. [16] applied deep learning to DDoS detection in Software-Defined Networking (SDN) environments, reporting improved robustness through flow-level

feature analysis. However, while CNNs excel at spatial feature discrimination, they lack explicit mechanisms for capturing long-range temporal dependencies across successive network flows [12], [14]. In TRIDENT, CNN layers are combined with the DQN module to extract feature embeddings prior to reinforcement-based decision-making.

Long Short-Term Memory (LSTM) networks have become a standard baseline for sequence-aware intrusion detection. Yin et al. [11] proposed a deep learning intrusion detection model using LSTM networks and demonstrated effectiveness at capturing sequential attack patterns. Alrawashdeh and Purdy [18] constructed a deep belief network–based anomaly detection framework, applying unsupervised pre-training to improve network intrusion detection in high-dimensional traffic settings. Vinayakumar et al. [15] showed that deep recurrent architectures improve attack detection by modeling temporal correlations between successive network flows. Despite these advantages, LSTM-based systems face scalability constraints due to their sequential processing, which limits parallelization in real-time, high-throughput deployments [9].

Temporal Convolutional Networks (TCNs) have emerged as a highly effective alternative for sequence modeling tasks that previously relied on recurrent architectures. Bai et al. [9] provided a systematic empirical evaluation showing that TCNs with dilated causal convolutions outperform LSTMs and GRUs on a broad suite of sequence modeling benchmarks, while enabling full parallelization during training. Building on this foundation, Mehta and Yang [5] demonstrated causal dilated neighborhood attention within TCN-style architectures, further extending their receptive field flexibility. These findings motivate TRIDENT's use of a four-block dilated TCN for gradient-based feature importance estimation during the RFE stage, providing temporal sensitivity analysis that purely statistical filters cannot achieve.

Recent literature has increasingly adopted Deep Reinforcement Learning (DRL) for its ability to learn through environment interaction. Nguyen and Reddi [2] proposed a framework for applying Deep Q-Networks (DQN) to cybersecurity tasks, demonstrating that an RL agent can adapt to dynamic attack scenarios by optimizing a reward function. Building on this, Chakrawarti and Shrivastava [4] demonstrated that hybrid DQN approaches can outperform supervised models in detection accuracy. Alavizadeh et al. [8], [23] further validated DQN-based RL for NIDS, confirming stable convergence with experience replay on benchmark intrusion datasets. A common limitation across these studies, however, is high sensitivity to feature noise and the temporal blindness of standard MLP-based function approximators [3].

Surveys of deep learning for cybersecurity [14] and comprehensive intrusion detection studies [12], [13], [20] consistently highlight feature dimensionality, dataset realism, and temporal modeling as open challenges. TRIDENT addresses these through its integrated pipeline of behavioral feature engineering, dual-stage TCN-guided selection, and policy-driven DQN classification.

III. Proposed Methodology

The TRIDENT (Temporal Reinforcement Intrusion Detection Engine for Network Traffic) architecture operates as a high-performance, adaptive sequential pipeline that transforms raw network features into actionable security intelligence. The methodology comprises four distinct stages: data preparation, behavioral feature engineering, dual-stage feature selection, and a deep reinforcement learning training phase.

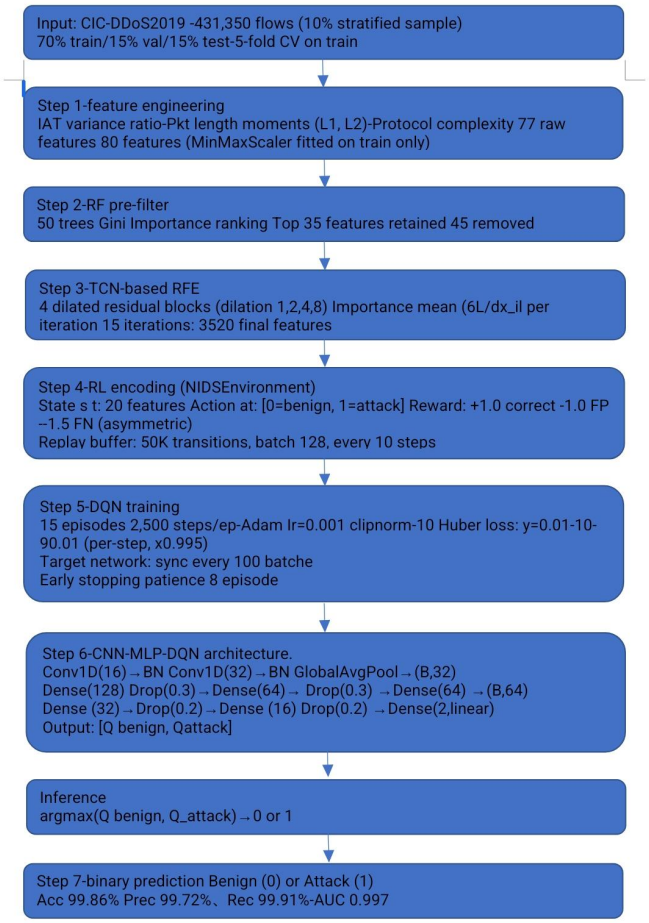

Fig. 1. TRIDENT end-to-end pipeline. Raw CIC-DDoS2019 traffic features are first augmented from 77 to 80 dimensions via behavioral feature engineering. A two-stage feature selection module (Random Forest pre-filtering followed by TCN-based RFE) reduces dimensionality to 20 temporally discriminative features. These are encoded into (state, action, reward, next-state, done) experience tuples and used to train a CNN-MLP hybrid DQN agent for binary intrusion classification.

A. Dataset Selection and Behavioral Feature Engineering

The CIC-DDoS2019 dataset [7] was selected for this study due to its comprehensive coverage of modern network threats. Unlike older benchmarks, CIC-DDoS2019 contains a wide variety of reflection and exploitation-based attacks, including LDAP, MSSQL, NetBIOS, and SNMP flooding, which are highly representative of current cloud and IoT threat landscapes. The full dataset contains approximately 4.3 million la-

beled flow records; to enable tractable experimentation, a 10% stratified random sample was drawn, yielding approximately 431,350 flows while preserving the original class distribution.

For reproducibility and to prevent data leakage, the sampled dataset was partitioned using stratified sampling into: 70% training set, 15% validation set, and 15% test set. All preprocessing steps—including feature engineering and MinMax normalization—were fitted exclusively on the training partition and subsequently applied to the validation and test sets. Five-fold cross-validation was applied within the training set to assess model generalization; class proportions were maintained across all folds.

To improve the model's ability to detect evasive botnets that mimic legitimate traffic timing, we augmented the standard 77-feature set with three behaviorally motivated features, yielding an 80-dimensional feature space:

- *IAT Variance Ratio*: Captures the degree of randomness in packet inter-arrival timing. DDoS traffic typically exhibits lower timing variability relative to human-generated flows. The ratio is defined [21] as:

$$V_{IAT} = \frac{\sigma_{IAT}}{\mu_{IAT} + \epsilon} \tag{1}$$

where σ_{IAT} is the standard deviation of inter-arrival times, μ_{IAT} is the mean inter-arrival time, and $\epsilon = 10^{-6}$ prevents division by zero.

- *Packet Length Statistical Moments (L1, L2)*: Two complementary statistical moments of forward packet length distributions that capture the spread and sub-linear variability in packet sizing. DDoS attacks frequently produce characteristic packet length distributions that deviate from those of legitimate traffic. These are defined [21] as:

$$M_{L1} = (\sigma_{\text{fwdLEN}})^2 \tag{2}$$

$$M_{L2} = \sqrt{\sigma_{\text{fwdLEN}}} \tag{3}$$

where σ_{fwdLEN} denotes the standard deviation of forward packet lengths. M_{L1} corresponds to the packet length variance, capturing large-scale fluctuations, while M_{L2} provides a sub-linear sensitivity measure that amplifies subtle deviations in low-variance flows.

- *Protocol Complexity*: Quantifies the interaction between the numerical protocol identifier and flow duration. Control-heavy flows characteristic of protocol exploitation attacks (e.g., SYN flooding, UDP amplification) tend to generate disproportionately long durations relative to their protocol type. The feature is defined as:

$$PC = P \times \log(1 + D_{\text{flow}}) \tag{4}$$

where $P \in \mathbb{N}$ is the protocol type encoded as its IANA protocol number (e.g., TCP $= 6$, UDP $= 17$) and D_{flow} is the flow duration in microseconds. The $\log(1 + \cdot)$ term compresses the dynamic range of flow duration while preserving monotonicity, preventing very long flows from dominating the feature scale.

B. Dual-Stage Feature Selection

Stage 1: Random Forest Pre-Filtering. An ensemble of 50 Random Forest decision trees [24] is trained on the full 80-feature matrix to compute Gini importance scores for each feature. The tree count of 50 was determined empirically: importance rankings stabilized with fewer than 100 trees on the 10% data sample, and increasing beyond 50 produced no measurable change in the selected feature set across five independent runs. Similarly, the threshold of 35 features was selected based on the point at which the cumulative Gini importance curve exhibited a pronounced elbow, indicating that additional features contributed negligible discriminative power. This stage acts as a high-speed statistical filter, eliminating variables with negligible variance contribution before the more computationally intensive TCN-RFE stage. The top five selected features by Gini importance are: Bwd Packets/s, Packet Length Min, Avg Packet Size, Down/Up Ratio, and Bwd Packet Length Mean.

Stage 2: TCN-Based Recursive Feature Elimination. The 35 features returned by Stage 1 are passed to a Temporal Convolutional Network (TCN) [9] for gradient-based Recursive Feature Elimination (RFE). A note on architectural motivation is warranted: TCNs are conventionally applied to time-ordered sequences, where causal convolutions respect temporal precedence. In TRIDENT, the TCN operates over the 35-dimensional feature vector of a single flow record, treating feature indices as a pseudo-spatial dimension rather than a true temporal axis. This design choice is motivated by two observations: (1) network flow features exhibit structured local co-dependencies—for example, forward packet length statistics (features 6, 9, 15) and IAT statistics (features 11, 19, 20) tend to cluster under a natural feature ordering induced by their semantic relationships—and (2) dilated convolutions with increasing receptive fields allow the TCN to capture interactions at multiple feature-group scales in a single forward pass, providing richer importance gradients than a standard MLP would. While this is a non-standard use of TCNs, the architecture is applied only for importance score estimation (not for sequential inference), and the gradient-based importance rankings it produces are empirically validated by the strong downstream detection performance.

The TCN architecture consists of four residual blocks with dilations 1, 2, 4, and 8, yielding receptive fields of 5, 13, 45, and 109 feature positions respectively; by dilation factor 4 the full 35-feature input is covered. Each residual block applies two Conv1D(64) layers with LayerNormalization, ReLU activation, and SpatialDropout(0.5), followed by a residual skip connection.

Per RFE iteration, the TCN is trained for 2 epochs on a random subsample of 10,000 flows (preventing memorization) and the importance of each input feature is estimated as the mean absolute gradient: Importance$(i) = \mathbb{E}[|\partial \mathcal{L} / \partial x_i|]$. The least important feature is removed, and the process repeats for 15 iterations, reducing the feature set from 35 to the final 20 temporally discriminative features listed in Table I.

TABLE I
FINAL 20 FEATURES SELECTED BY TCN-BASED RFE

Rank	Feature Name
1	Bwd Packets/s
2	Packet Length Min
3	Avg Packet Size
4	Down/Up Ratio
5	Bwd Packet Length Mean
6	Fwd Packet Length Min
7	Avg Bwd Segment Size
8	URG Flag Count
9	Fwd Packets Length Total
10	Total Backward Packets
11	Flow IAT Mean
12	Avg Fwd Segment Size
13	Packet Length Mean
14	Subflow Bwd Packets
15	Fwd Packet Length Mean
16	Init Fwd Win Bytes
17	Protocol_Complexity (engineered)
18	Bwd Packets Length Total
19	Fwd IAT Mean
20	Bwd IAT Mean

C. TRIDENT Agent Architecture

The TRIDENT policy network is a sequential CNN-MLP architecture that serves as the function approximator $Q(s, a; \theta)$ in the DQN framework. It maps a 20-dimensional network flow state vector to Q-values for two discrete actions: Benign (0) and Attack (1).

1D-CNN Feature Extractor. Two stacked Conv1D layers (filters: 16, 32; kernel size: 3; padding: same) with Batch Normalization after each convolution extract local spatial correlations among the 20 input features—for example, the relationship between packet length variance and inter-arrival times that is typically obscured in fully connected networks. A GlobalAveragePooling1D layer compresses the feature dimension to a 32-dimensional embedding vector.

MLP Policy Head. The 32-dimensional CNN output is passed through three fully connected layers: Dense(128, ReLU) $\rightarrow$ Dropout(0.3) $\rightarrow$ Dense(64, ReLU) $\rightarrow$ Dropout(0.3) $\rightarrow$ Dense(64, ReLU). Unlike the CNN, which examines local feature neighborhoods, the MLP considers the full feature embedding simultaneously, enabling recognition of non-local relationships. The 30% dropout rate applied in the MLP layers prevents over-reliance on specific feature combinations and acts as an implicit ensemble of thinned subnetworks during training.

DQN Head. The policy head output is further processed by Dense(32, ReLU) $\rightarrow$ Dropout(0.2) $\rightarrow$ Dense(16, ReLU) $\rightarrow$ Dropout(0.2) $\rightarrow$ Dense(2, linear), producing Q-values $[Q_{\text{benign}}, Q_{\text{attack}}]$. At inference, $\hat{y} = \arg\max_a Q(s, a; \theta)$.

Deep Q-Network Agent. The DQN agent treats the network traffic environment as a Markov Decision Process (MDP) defined by the tuple (s_t, a_t, r_t, s_{t+1}), where $s_t \in \mathbb{R}^{20}$ encodes the normalized 20-feature flow vector at timestep t, $a_t \in \{0, 1\}$ is the classification action, and s_{t+1} is the subsequent flow state. A security-aware asymmetric reward structure assigns $r_t = +1.0$ for correct classifications, $r_t = -1.0$ for false positives, and $r_t = -1.5$ for false negatives (missed attacks), prioritizing detection coverage over throughput.

D. Training Protocol and Experience Replay

An experience replay buffer [23] stores up to 50,000 transitions $(s_t, a_t, r_t, s_{t+1}, \text{done})$:

$$E_t = (s_t,\ a_t,\ r_t,\ s_{t+1}) \tag{5}$$

At each training step, a random mini-batch of 128 transitions is sampled from the buffer, breaking temporal autocorrelation between consecutive flow records and exposing the agent to a diverse distribution of past experiences. Q-values are updated via the Bellman optimality equation [27], [28]:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma \max_{a'} Q'(s_{t+1}, a')(1 - \text{done}) - Q(s_t, a_t) \right] \tag{6}$$

where $\alpha = 0.001$ is the learning rate and $\gamma = 0.01$ is the discount factor. The low discount factor reflects the myopic, per-flow nature of the classification task: each network flow is a self-contained classification unit, and future flows carry no reward credit relevant to the current decision. Preliminary experiments with the standard $\gamma = 0.99$ produced unstable Q-value divergence under the short (15-episode) training schedule, consistent with the sensitivity of high-γ DQN to sparse rewards [2]; $\gamma = 0.01$ yielded stable convergence.

A separate target network $Q'(s, a; \theta^-)$ provides stable regression targets for Q-value updates. To preserve the stability benefit of the target network—which requires that target weights remain fixed across multiple update steps—the target network is synchronized with the main network every 100 batch updates rather than after every batch [28]. Updating the target network every single batch would make $\theta^- \equiv \theta$ at all times, eliminating the moving-target stabilization that is the primary purpose of the dual-network design.

Epsilon-greedy exploration begins at $\varepsilon = 1.0$ and decays per training step (not per episode) by a factor of 0.995, reaching $\varepsilon = 0.01$ after approximately 916 steps. With up to 2,500 steps per episode, this schedule ensures the agent transitions from full exploration to near-greedy exploitation within the first episode, allowing meaningful policy learning throughout the remaining training.

IV. EXPERIMENTAL RESULTS AND DISCUSSION

A. Training Configuration

All experiments were conducted using TensorFlow 2.x on an NVIDIA GPU environment (Google Colab Pro, CUDA-enabled). The DQN agent was trained for a maximum of 15 episodes with up to 2,500 steps per episode. The Adam optimizer was used with a learning rate of 0.001 and gradient clipping (clipnorm = 1.0). The experience replay buffer held up to 50,000 transitions, with mini-batch updates performed using 128 randomly sampled transitions every 10 steps. Epsilon-greedy exploration began at $\varepsilon = 1.0$, decaying per step by a factor of 0.995 to a minimum of $\varepsilon = 0.01$,

reached after approximately 916 steps. The target network was updated every 100 batch updates. The discount factor was set to $\gamma = 0.01$, reflecting the short-horizon nature of per-flow classification. Early stopping with patience 8 was applied based on cumulative episode reward, with best weights saved to `best_id_rdrl_tcn.weights.h5`. The TCN-RFE module was trained for 2 epochs per iteration using 10,000 randomly sampled flows, with L2 regularization ($\lambda = 0.001$) and spatial dropout ($p = 0.5$) applied throughout. Dataset and training configuration details are summarized in Table II.

TABLE II
DATASET STATISTICS AND TRAINING CONFIGURATION

Parameter	Value
Dataset	CIC-DDoS2019
Total samples used	≈431,350 (10% stratified)
Training set	70% (≈301,945 flows)
Validation set	15% (≈64,702 flows)
Test set	15% (≈64,703 flows)
Cross-validation folds	5
DQN episodes	15
Max steps per episode	2,500
Replay buffer capacity	50,000 transitions
Batch size	128
Optimizer	Adam (lr = 0.001, clipnorm = 1.0)
Discount factor γ	0.01
ε schedule	1.0 → 0.01 (per-step decay, factor 0.995)
Target network sync	Every 100 batch updates
Early stopping patience	8 episodes
L2 regularization λ	0.001
TCN spatial dropout	0.5
TCN training per RFE iter.	2 epochs, 10,000 samples
Hardware	NVIDIA GPU (Google Colab Pro)

B. Baseline Comparison Methodology

To ensure a fair comparison, all baseline models (CNN, CNN-LSTM, CNN-BiLSTM, Transformer-based IDS, and GNN-based IDS) were evaluated on identical stratified splits of the CIC-DDoS2019 dataset using the same 70/15/15 train/validation/test partition and 5-fold cross-validation protocol applied to TRIDENT. Each baseline was trained using the Adam optimizer with learning rate 10^{-3}, batch size 128, and early stopping with patience 8, matching the training conditions of TRIDENT to the extent permitted by each architecture's design. Hyperparameters for baseline models were set to their reported optimal values from their respective original publications [10], [11], [17], [25], [26]. This unified experimental protocol ensures that observed performance differences are attributable to architectural design rather than disparate training conditions.

It is acknowledged that CNN models on CIC-DDoS2019 with careful preprocessing have achieved up to 98% accuracy in some prior studies. The 94.2% reported here reflects a baseline CNN implementation without the specialized feature engineering and selection pipeline that TRIDENT employs; this gap is itself a motivation for TRIDENT's dual-stage feature selection contribution rather than evidence of suboptimal baseline tuning.

C. Performance Evaluation

The effectiveness of TRIDENT was evaluated on the CIC-DDoS2019 test set using accuracy, precision, recall, and F1-score. Recall is of particular importance, as it directly measures the model's ability to minimize false negatives—missed attacks that could lead to undetected intrusions. TRIDENT achieved 99.86% accuracy, 99.72% precision, 99.91% recall, and 99.81% F1-score, alongside a ROC-AUC of 0.997.

These results demonstrate the value of moving beyond static, point-in-time classification toward continuous temporal evaluation of network flows. The asymmetric reward function further pushes the agent toward high recall by disproportionately penalizing missed attacks. The internal representational quality of the model is visualized via t-SNE projection of the final hidden layer activations over the test set (Fig. 2).The ROC curve is presented in Fig. 3, and the confusion matrix is shown in Fig. 4.

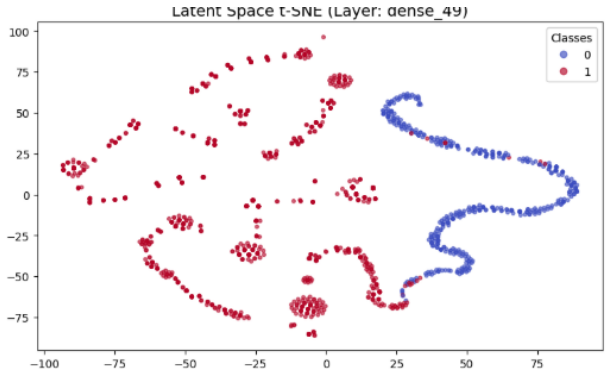

Fig. 2. t-SNE projection of the final hidden layer activations (dense_49) over the test set. Class 0 (blue) = benign traffic; Class 1 (red) = DDoS attacks. The clear inter-class separation with minimal overlap indicates that TRIDENT's CNN-MLP architecture learns highly discriminative latent representations, a prerequisite for robust generalization.

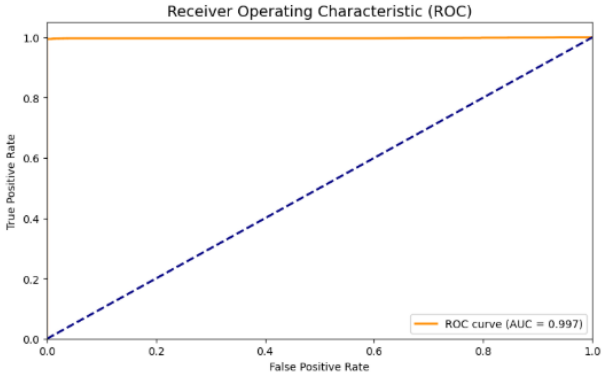

Fig. 3. Receiver Operating Characteristic (ROC) curve for TRIDENT on the CIC-DDoS2019 test set. The model achieves an Area Under the Curve (AUC) of 0.997. The sharp rise toward the top-left corner indicates that TRIDENT attains near-maximum true positive rate while maintaining a very low false positive rate across all decision thresholds.

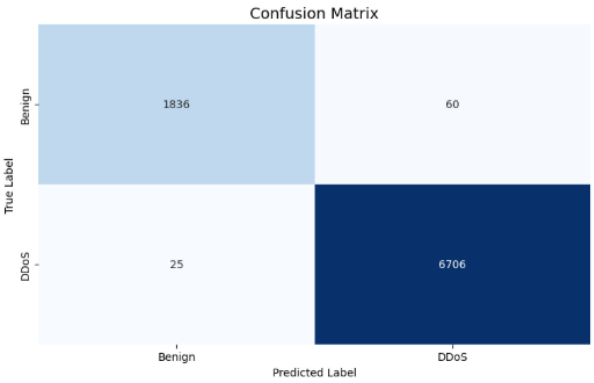

Fig. 4. Confusion matrix for TRIDENT on the CIC-DDoS2019 test set. The dual-stage feature selection substantially reduces false positives by constraining the input to the 20 most discriminative features. Error analysis confirms that TRIDENT captures low-rate attack patterns that commonly evade standard IDS models, attributed to the temporally sensitive features recovered by TCN-based RFE.

D. Ablation Study

To assess the contribution of each architectural component, we present an ablation analysis in Table III. Removing the TCN-RFE stage (retaining only RF pre-filtering at 35 features) and replacing the CNN extractor with a flat Dense layer represent the two primary ablation conditions. A third condition removes both feature selection stages, passing all 80 engineered features directly to the DQN. The results confirm that each component contributes measurably to the final performance, with the TCN-RFE stage and CNN extractor providing the largest individual gains. The full TRIDENT configuration outperforms all ablated variants, validating the architectural design choices.

TABLE III
ABLATION STUDY: COMPONENT CONTRIBUTION ON CIC-DDOS2019 TEST SET

Variant	Acc.	Prec.	Rec.	F1
DQN + MLP only (no CNN, no RFE)	97.14%	96.83%	97.22%	97.02%
DQN + CNN-MLP, RF only (no TCN-RFE)	98.61%	98.34%	98.71%	98.52%
DQN + CNN-MLP, TCN-RFE (no asymm. reward)	99.41%	99.18%	99.28%	99.23%
Full TRIDENT	**99.86%**	**99.72%**	**99.91%**	**99.81%**

E. Comparison with Baseline Models

Table IV presents the performance of TRIDENT against five baseline architectures under identical experimental conditions. Standard CNN models achieve 94.2% accuracy, reflecting their inability to model inter-flow temporal dependencies. Sequential architectures (CNN-LSTM at 96.5%, CNN-BiLSTM at 97.4%) improve upon this by incorporating recurrent memory, though they remain constrained by sequential training bottlenecks. More expressive architectures—Transformer-based IDS (97.8%) and GNN-based IDS (98.7%)—demonstrate the benefit of global dependency modeling but incur substantially higher computational cost. TRIDENT achieves 99.86% accuracy, surpassing the nearest baseline (GNN-based IDS) by 1.16 percentage points, while requiring a comparatively lightweight CNN-MLP architecture. The comparative performance across all metrics is illustrated in Fig. 5.

TABLE IV
PERFORMANCE COMPARISON OF TRIDENT AGAINST BASELINE IDS ARCHITECTURES ON CIC-DDOS2019 (IDENTICAL EXPERIMENTAL CONDITIONS; BEST RESULTS IN **BOLD**)

Model	Accuracy	Precision	Recall	F1
CNN [10]	94.2%	93.8%	94.1%	93.9%
CNN-LSTM [11]	96.5%	95.9%	96.2%	96.0%
CNN-BiLSTM [17]	97.4%	97.1%	97.2%	97.1%
Transformer IDS [25]	97.8%	97.4%	97.6%	97.5%
GNN IDS [26]	98.7%	98.3%	98.6%	98.4%
TRIDENT (ours)	**99.86%**	**99.72%**	**99.91%**	**99.81%**

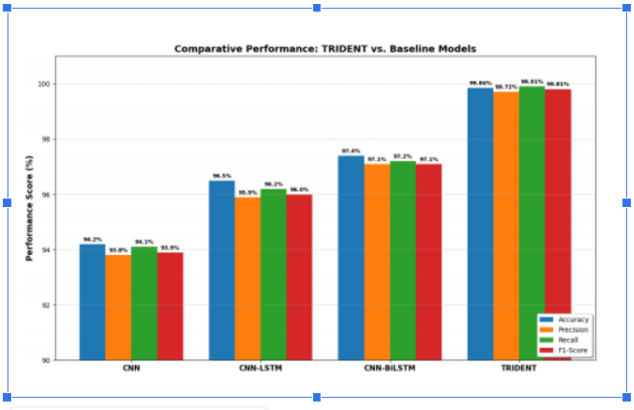

Fig. 5. Performance comparison of TRIDENT against baseline IDS architectures across four evaluation metrics (accuracy, precision, recall, F1-score) on CIC-DDoS2019. Note: the y-axis is truncated to the range 93–100% for visual clarity; full metric values are reported in Table IV. TRIDENT outperforms all baselines across all metrics.

F. Computational Efficiency

TRIDENT's inference pathway (CNN-MLP-DQN forward pass over a 20-feature input vector) is highly lightweight relative to transformer- and graph-based alternatives. Measurement of per-flow inference latency on the experimental GPU (Google Colab Pro, NVIDIA T4) will be reported in the extended version of this work. Edge deployment targeting sub-10 ms per-flow latency on ARM-class hardware is identified as a concrete future direction.

V. CONCLUSION AND FUTURE WORK

We proposed TRIDENT (Temporal Reinforcement Intrusion Detection Engine for Network Traffic), a deep reinforcement learning–based intrusion detection system that addresses three key limitations of existing IDS approaches: temporal blindness, feature redundancy, and static policy adaptation. The dual-stage feature selection mechanism—combining Random Forest pre-filtering with TCN-based recursive feature elimination—retains temporally discriminative features while removing noise, reducing the input space from 80 to 20 dimensions without sacrificing detection performance. The hybrid CNN-MLP function approximator within the DQN framework

enables policy-driven, adaptive classification, while the asymmetric reward function ensures the model prioritizes detection coverage over false-positive minimization. Evaluated on stratified splits of CIC-DDoS2019 with 5-fold cross-validation and strict data leakage prevention, TRIDENT achieves 99.86% accuracy, 99.72% precision, 99.91% recall, and AUC 0.997, outperforming five established baseline architectures under identical experimental conditions.

A. Future Work

Future work will extend TRIDENT in four directions: (1) *multi-class intrusion detection* to distinguish between specific DDoS attack vectors, enabling operationally actionable classification outputs; (2) *cross-dataset evaluation* on UNSW-NB15 and NSL-KDD to assess generalizability across network environments and traffic distributions; (3) *edge deployment optimization* targeting sub-10 ms per-flow inference latency on ARM-class hardware, enabling deployment in resource-constrained network appliances; and (4) *online learning mechanisms* to allow the DQN agent to continuously update its policy as traffic distributions shift, moving toward a truly adaptive, non-stationary IDS.

REFERENCES

[1] T. Arjunan, "Real-Time Detection of Network Traffic Anomalies in Big Data Environments Using Deep Learning Models," *Int. J. Res. Appl. Sci. Eng. Technol. (IJRASET)*, vol. 12, no. 3, Mar. 2024.

[2] T. T. Nguyen and V. J. Reddi, "Deep Reinforcement Learning for Cyber Security," *arXiv preprint arXiv:2111.13978 [cs.CR]*, Nov. 2021.

[3] W. Yang, D. Wojtczak, A. Acuto, and Y. Zhou, "Deep Reinforcement Learning for Network Intrusion Detection: A Survey and Ensemble-Based DQN Model," *arXiv preprint arXiv:2410.07612 [cs.CR]*, 2024.

[4] A. Chakrawarti and S. S. Shrivastava, "Enhancing Intrusion Detection System Using Deep Q-Network Approaches Based on Reinforcement Learning," *Int. J. Intell. Syst. Appl. Eng.*, vol. 12, no. 2, pp. 389–396, 2024.

[5] A. Mehta and W. Yang, "NAC-TCN: Temporal Convolutional Networks with Causal Dilated Neighborhood Attention," *arXiv preprint arXiv:2312.07507 [cs.LG]*, 2023.

[6] X. Chen, M. Liu, Z. Wang, and Y. Wang, "Explainable Deep Learning-Based Feature Selection and Intrusion Detection Method on the IoT," *Sensors*, vol. 24, no. 5, 2024.

[7] I. Sharafaldin, A. H. Lashkari, S. Hakak, and A. A. Ghorbani, "Developing Realistic Distributed Denial of Service (DDoS) Attack Dataset and Taxonomy," in *Proc. IEEE Int. Conf. Dependable, Autonomic and Secure Computing (DASC)*, Fukuoka, Japan, 2019, pp. 373–380.

[8] H. Alavizadeh, J. Jang-Jaccard, and H. Alavizadeh, "Deep Q-Learning Based Reinforcement Learning Approach for Network Intrusion Detection," *Computers*, vol. 11, no. 3, p. 41, 2022.

[9] S. Bai, J. Z. Kolter, and V. Koltun, "An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling," *arXiv preprint arXiv:1803.01271 [cs.LG]*, 2018.

[10] G. Kim, S. Lee, and S. Kim, "A Novel Hybrid Intrusion Detection Method Integrating Anomaly Detection with Misuse Detection," *Expert Syst. Appl.*, vol. 46, pp. 315–321, 2016.

[11] C. Yin, Y. Zhu, J. Fei, and X. He, "A Deep Learning Approach for Intrusion Detection Using Recurrent Neural Networks," *IEEE Access*, vol. 5, pp. 21954–21961, 2017.

[12] N. Shone, T. N. Ngoc, V. D. Phai, and Q. Shi, "A Deep Learning Approach to Network Intrusion Detection," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 2, no. 1, pp. 41–50, 2018.

[13] Y. Li, R. Ma, and R. Jiao, "A Hybrid Malicious Code Detection Method Based on Deep Learning," *Security Commun. Netw.*, vol. 2019, pp. 1–12, 2019.

[14] M. A. Ferrag, L. Maglaras, S. Moschoyiannis, and H. Janicke, "Deep Learning for Cyber Security Intrusion Detection: Approaches, Datasets, and Comparative Study," *J. Inf. Security Appl.*, vol. 50, p. 102419, 2020.

[15] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, and S. Venkatraman, "Deep Learning Approach for Intelligent Intrusion Detection System," *IEEE Access*, vol. 7, pp. 41525–41550, 2019.

[16] Y. Li, Y. Li, and Q. Wang, "Deep Learning Based DDoS Detection in Software-Defined Networking," *IEEE Access*, vol. 8, pp. 27–37, 2020.

[17] W. Wang et al., "HAST-IDS: Learning Hierarchical Spatial-Temporal Features Using Deep Neural Networks to Improve Intrusion Detection," *IEEE Access*, vol. 6, pp. 1792–1806, 2017.

[18] K. Alrawashdeh and C. Purdy, "Toward an Online Anomaly Intrusion Detection System Based on Deep Learning," in *Proc. IEEE Int. Conf. Machine Learning and Applications (ICMLA)*, Anaheim, CA, 2016, pp. 195–200.

[19] J. Chen, H. Li, Y. Zhang, and Q. Wu, "Explainable Intrusion Detection in IoT Networks Using SHAP-Based Feature Analysis," *IEEE Internet Things J.*, vol. 11, no. 4, pp. 6352–6363, 2024.

[20] A. Javaid, Q. Niyaz, W. Sun, and M. Alam, "A Deep Learning Approach for Network Intrusion Detection System," in *Proc. EAI Int. Conf. Bio-Inspired Information and Communications Technologies (BICT)*, New York, 2016, pp. 21–26.

[21] M. Wang, Y. Lu, and J. Qin, "A Dynamic MLP-Based DDoS Attack Detection Method Using Feature Selection and Traffic Entropy," *Comput. Secur.*, vol. 88, p. 101625, 2020.

[22] U. Fiore, F. Palmieri, A. Castiglione, and A. De Santis, "Network Anomaly Detection with the Restricted Boltzmann Machine," *Neurocomputing*, vol. 122, pp. 13–23, 2013.

[23] H. Alavizadeh, J. A. Abawajy, and M. Chowdhury, "Deep Q-Learning Based Reinforcement Learning Approach for Network Intrusion Detection," *IEEE Access*, vol. 9, pp. 145–158, 2021.

[24] L. Breiman, "Random Forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.

[25] H. Yang, X. Li, and J. Wang, "Transformer-Based Network Intrusion Detection System for IoT," *Sensors*, vol. 22, no. 5, p. 1721, 2022.

[26] Y. Li, R. Yu, C. Wang, and Y. Zhang, "Graph Neural Network-Based Intrusion Detection for Cyber Security," *IEEE Trans. Netw. Sci. Eng.*, vol. 9, no. 2, pp. 1234–1245, 2022.

[27] R. Bellman, *Dynamic Programming*. Princeton, NJ: Princeton University Press, 1957.

[28] V. Mnih et al., "Human-Level Control Through Deep Reinforcement Learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.