Journal of Information & Optimization Sciences
ISSN 0252-2667 (Print), ISSN 2169-0103 (Online)
DOI : 10.47974/JIOS-2431

BECT : A Block-Embedded ConvNeXt Transformer for robust intrusion detection using adversarial self-paced learning

Majid A. Jahsh [†]
Mina Z. Lighvan *
Seyed A. Mortazavi [§]
Department of Computer Engineering
Faculty of Electrical & Computer Engineering
University of Tabriz
Tabriz
Iran

Abstract

Block-Embedded ConvNeXt Transformer (BECT) addresses the growing challenge of detecting sophisticated cyber-attacks in dynamic network environments by jointly modeling fine-grained traffic content and high-level flow context. In this work, raw network packets in pcap format are chronologically ordered, grouped into flows, normalized to a fixed length, and reshaped into image-like matrices that preserve both byte sequences and structural dependencies. These flow images are then processed by a hybrid backbone that embeds ConvNeXt-style convolutional blocks and window-based transformer attention within each block to capture local spatial patterns and long-range temporal relationships in traffic data. Robustness and training stability are further enhanced by an adversarial self-paced learning regime that combines projected gradient descent–generated adversarial examples with a curriculum-like schedule that gradually shifts focus from easy to harder samples. Extensive experiments on chronologically split traffic demonstrate that BECT achieves strong detection performance under benign conditions and maintains high resilience against adversarial evasion, while ablation studies confirm the complementary benefits of window-based attention and adversarial self-paced training over purely convolutional, purely transformer, or standard training baselines. Overall, the proposed framework provides an effective and

[†] *ORCID-ID:* **0009-0005-9774-7681**
* *ORCID-ID:* **0000-0002-4555-597X**
[§] *ORCID-ID:* **0000-0002-3391-3290**

[†] *E-mail:* **majid.abduljabar@phd.tabrizu.ac.ir**
* *E-mail:* **mzolfy@tabrizu.ac.ir (Corresponding Author)**
[§] *E-mail:* **sa.mortezavi@tabrizu.ac.ir**

© Taru Publications

principled solution for robust intrusion detection in complex and adversarial network settings.

Subject Classification: *68T07, 68M25, 68T05.*

Keywords: *Adversarial training, ConvNeXt, Intrusion detection, Transformer, Window-based self-attention.*

1. Introduction

Traditional statistical and signature-based intrusion detection systems (IDS) are unable to combat sophisticated, zero-day cyber attacks because of an exponential increase in traffic complexity brought about by the fast growth of network infrastructures, cloud services, and the Internet of Things (IoT) [1, 34]. Deep Learning (DL) has emerged as a key component of modern-day cybersecurity, employing architectures such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) to independently extract intricate patterns and surpass traditional models on benchmarks [1–3, 6, 10]. The proliferation of specialised threats like Sinkhole and Distributed Denial of Service (DDoS) attacks in resource-constrained IoT and 5G systems has led to an increased focus on sophisticated hybrid and ensemble frameworks in recent literature [4, 5, 16, 18, 31, 33]. Models have been shown to prioritise important features and reduce detection latency more effectively when they combine CNNs for spatial extraction with Long Short-Term Memory (LSTM) or Gated Recurrent Unit (GRU) networks for temporal sequence processing, in addition to attention mechanisms, autoencoders, and generative adversarial networks (GANs) [7–9, 11–15, 17, 19–24]. Despite their empirical effectiveness, these hybrid designs frequently ignore the structural variance of network flows and remain highly vulnerable to adversarial perturbations. Additionally, in dynamic network conditions, the lack of curriculum learning mechanisms in current approaches often results in unstable training phases and decreased generalisation. This paper presents the Block-Embedded ConvNeXt Transformer (BECT), a novel architecture that simultaneously captures long-range temporal dependencies using Transformer-based attention and local spatial features via convolutional blocks by converting raw network traffic into fixed, image-like representations. Using a curriculum learning methodology that gradually moves from benign to complicated adversarial samples, BECT integrates an adversarial self-paced learning framework based on a Projected Gradient Descent (PGD) engine to strengthen the model's resistance to evasion strategies. Ultimately, the contributions of this work can be summarized as follows: (i) a new architecture named BECT that captures local and global traffic patterns; (ii) an adversarial self-paced training scheme, improving robustness and stabilising learning; and (iii) a

thorough evaluation demonstrating the effectiveness of the proposed framework in both benign and adversarial settings, together with ablation studies confirming the contribution of each component.

2. Materials and Methods

In this section, we introduce the Block-Embedded ConvNeXt Transformer (BECT) architecture for robust and efficient intrusion detection. This approach tackles two major problems in network security: the variability of raw traffic and flow sizes, and the vulnerability of deep learning models to adversarial attacks. To address these issues, we adopt a three-step process. First, a padding policy ensures that every network-flow sequence has the same length, and the sequences are reorganized into two-dimensional images that preserve the native traffic structure. Second, these flow images are processed by the BECT architecture, which uses ConvNeXt-based convolutional blocks and transformer-based attention to capture local and global patterns in the traffic. Third, we use an adversarial self-paced training process that combines PGD-generated adversarial examples, which provide resistance to evasion attacks, with a sample-weighting scheme that emphasizes easy samples early in training and gradually shifts toward harder samples, enabling robust and progressive learning. Together, these components form a principled and effective intrusion detection methodology that balances discriminative capacity, robustness, and versatility. Each stage is described in detail in the following subsections.

2.1 *Data Preprocessing and Representation*

Intrusion detection systems based on deep learning are sensitive to the choice, accuracy, and reliability of their input representation. The input data in this study are raw network packets in pcap format. Unlike techniques that use pre-computed statistical features (e.g., the 80 CICIDS flow statistics), the proposed framework operates on raw packet bytes aggregated per flow. No crafted statistical features are used as inputs into the model. Rather, the model is trained on the raw traffic bytes, which allows it to exploit both low-level payload representations and higher-level structural patterns implicit in network traffic flows. To simulate real deployment and avoid temporal information leakage, the dataset is chronologically sorted by packet timestamp. A time-based partitioning scheme is then applied, sequentially distributing 70%, 10%, and 20% of the flows into the training, validation, and testing subsets, respectively. This approach better reflects real-world deployment conditions, in which a model is trained on historical traffic and tested on future, unseen traffic. All preprocessing steps, which encompass truncation, padding, resizing, and normalization, are executed after the chronological split described above. Any statistical parameters needed during preprocessing are calculated on

the training set only and then applied unchanged to the validation and test sets. This protocol prevents data leakage and ensures that the evaluation is fair and reproducible. Preprocessing begins by parsing the pcap files to extract packet attributes, including timestamps, source and destination IP addresses, port numbers, transport-layer protocols, and payload contents. Packets are then grouped into flows, where each flow is a two-way communication session between two endpoints. A flow F_i can be defined mathematically as follows:

$$F_i = \{p_1, p_2, \cdots, p_N\}, \tag{1}$$

Here, p_j denotes the jth packet of the flow, and N is the number of packets in the flow. Both the sequential ordering and the temporal dependencies of the packets are preserved within each flow, and they play a crucial role in distinguishing between benign and malicious patterns. Since the flow lengths are heterogeneous, sequence-length normalization is required before model training. Let L be the original length of a flow and L_f be the fixed length used for truncation or padding. L_f is set to 2500, which approximately equals the 90th percentile of the training flow lengths and is determined from the training set alone. This choice offers a trade-off between information retention and computational tractability, and is also free of test-set effects. Flow sizes above L_f are truncated to the first 2500 bytes, and smaller flows are padded. A two-step approach is used to counter the excessive zero-padding, which may bias the feature distribution of short flows. First, a short flow of length $L < L_f$ is padded to the nearest square number $L_p = \left\lceil \sqrt{L} \right\rceil^2$, where L_p is the padded length. The resulting sequence is reshaped into a two-dimensional array of size $\sqrt{L_p} \times \sqrt{L_p}$. Second, to unify the input dimensions across all samples, this array is resized to 50×50, resulting in a fixed-size grayscale image representation [25]. Letting I_i denote the processed representation of the ith flow, we obtain:

$$I_i \in \mathbb{R}^{50\times 50},\ \forall i \in 1,2,\cdots,n, \tag{2}$$

where n denotes the total number of flows. This conversion enables both convolutional and transformer-based operations to be applied without losing the sequential nature of the original raw byte stream. Lastly, all numerical attributes are normalised to a common scale through min-max normalisation:

$$x' = \frac{x - x_{\min}}{x_{\max} - x_{\min}}, \tag{3}$$

where x is the original value, $x_{\min}$ and $x_{\max}$ are the lowest and highest values of the feature respectively. This normalisation not only stabilises training but also accelerates convergence by preventing features with disproportionately large scales from dominating. All data partitioning is

performed before any normalization or statistical transformation to avoid leakage of statistical information. Accordingly, the dataset is first divided into training, validation, and test sets. The normalization parameters $x_{\min}$ and $x_{\max}$ are computed exclusively from the training set; both are then applied without modification to the validation and test sets. Consequently, the validation and test data provide no statistical information during preprocessing or training, and the evaluation is therefore leakage-free, closely resembling a deployment scenario where the future traffic statistics are unavailable during training. The resulting representations are then passed to the proposed Block-Embedded ConvNeXt Transformer (BECT) model.

2.2 *Block-Embedded ConvNeXt Transformer Architecture*

The proposed Block-Embedded ConvNeXt Transformer (BECT) is designed to exploit both the local spatial patterns characteristic of convolutional neural networks and the global contextual relationships that are effectively captured by transformers. Traditional convolution-based intrusion detection models rely on local receptive fields, which extract fine-grained features but often fail to capture long-range dependencies across traffic streams. Pure transformer-based models, on the other hand, rely on global attention mechanisms but do not perform well numerically in capturing detailed local structures and are generally computationally costly. BECT combines the strengths of both paradigms in a hybrid architecture, which makes it especially suitable for flow-image-based intrusion detection. At a high level, the BECT pipeline consists of (i) patching flow images into a tokenized representation, (ii) hierarchical feature learning through stacked BECT blocks, and (iii) global pooling and a classification head. Each BECT block is a composite module comprising window-based attention, a ConvNeXt convolutional refinement branch, and an MLP expansion–projection pathway, connected by residual connections. Given a flow image $\mathbf{X} \in \mathbb{R}^{B\times C_{in}\times H\times W}$ where B is the batch size, C_{in} is the number of input channels (here $C_{in} = 1$ for grayscale), and H, W denote spatial dimensions, the first step is to partition the image into non-overlapping patches. Instead of explicitly reshaping the image, BECT uses a convolutional projection, as in vision transformers [26], with a kernel size and a stride both equal to a patch size p. This operation not only splits the image but projects each patch into a d-dimensional embedding space:

$$\mathbf{Z}_0 = \text{PatchEmbed}(\mathbf{X}) = \text{Conv}_{PS}(\mathbf{X}), \tag{4}$$

where $\mathbf{Z}_0 \in \mathbb{R}^{B\times N\times d}$, d is the embedding dimension, and $N = \frac{H}{p} \cdot \frac{W}{p}$ is the number of patches. Each patch token thus serves as an atomic unit of information for subsequent processing using BECT blocks. The BECT block is the fundamental computational unit of the architecture. It receives a sequence of patch tokens $\mathbf{Z} \in \mathbb{R}^{B\times N\times d}$ and refines it through three

successive pathways, each preceded by normalization and followed by a residual addition:

$$\mathbf{Z}_1 = \mathbf{Z} + \text{W-MSA}\big(\text{LN}(\mathbf{Z})\big), \tag{5}$$

$$\mathbf{Z}_2 = \mathbf{Z}_1 + \text{ConvNeXt}\big(\text{LN}(\mathbf{Z}_1)\big), \tag{6}$$

$$\mathbf{Z}_3 = \mathbf{Z}_2 + \text{MLP}\big(\text{LN}(\mathbf{Z}_2)\big), \tag{7}$$

where LN(·) denotes Layer Normalization, W-MSA is the window-based multi-head self-attention, ConvNeXt is a depthwise-separable convolutional refinement unit [27], and MLP is a two-layer feed-forward network. Although powerful, global self-attention is quadratic in the number of tokens N, which is impractical for high-resolution flow images. To address this, BECT employs window-based attention [28], which restricts attention to $M \times M$ sized non-overlapping windows (specifically 4×4). This design balances representational power and efficiency: tokens interact densely within each window, while information is exchanged across window boundaries by the 7 × 7 depthwise convolution of the subsequent ConvNeXt branch, so window shifting is not used. The multi-head self-attention is calculated as follows for each window token set $\mathbf{X}_w \in \mathbb{R}^{M^2 \times d}$:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_h}}\right)\mathbf{V}, \tag{8}$$

with projections

$$\mathbf{Q} = \mathbf{X}_w\mathbf{W}_Q,\ \mathbf{K} = \mathbf{X}_w\mathbf{W}_K,\ \mathbf{V} = \mathbf{X}_w\mathbf{W}_V, \tag{9}$$

where $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V \in \mathbb{R}^{d \times d_h}$ are learnable matrices, $d_h = d/h$, and h is the number of heads. The outputs of all windows are then merged back into their original spatial order. This mechanism lets BECT capture intra-window dependencies efficiently while avoiding the computational overhead of full global attention ($O(N^2)$ vs. $O(NM^2)$). Transformers capture long-range dependencies well but often lack sensitivity to fine-grained local details. To address this, BECT places a ConvNeXt block after the attention module to emphasize local features. This block uses a depthwise convolutional kernel of size 7×7 to capture a wider local context, followed by group normalization and two pointwise linear layers with a GELU non-linearity:

$$\mathbf{Y} = \text{DWConv}_{7\times7}(\mathbf{X}), \tag{10}$$

$$\widehat{\mathbf{Y}} = \text{GN}(\mathbf{Y}), \tag{11}$$

$$\mathbf{Z} = \mathbf{W}_2\,\sigma\big(\mathbf{W}_1\widehat{\mathbf{Y}}\big), \tag{12}$$

Here $\mathbf{W}_1 \in \mathbb{R}^{d \times 4d}$, $\mathbf{W}_2 \in \mathbb{R}^{4d \times d}$ and $\sigma(\cdot)$ is the GELU activation. This convolutional refinement makes the model less sensitive to small, local changes. It also complements the attention module, allowing it to focus on

the global context. Finally, the MLP module enriches the token representation by projecting each embedding into a higher-dimensional space and then projecting it back to the original dimension:

$$\mathbf{Z}' = \mathbf{W}_4\,\sigma(\mathbf{W}_3\mathbf{X}), \tag{13}$$

where $\mathbf{W}_3 \in \mathbb{R}^{d\times d_m}$, $\mathbf{W}_4 \in \mathbb{R}^{d_m\times d}$, $d_m = \alpha d$ is the hidden dimension, and $\alpha = 4$ is the expansion ratio. This module provides both non-linear feature mixing and channel-level expressivity. BECT arranges its blocks into multiple stages to learn hierarchical features at multiple depths. If the architecture specifies depths $[L_1, L_2, \cdots, L_S]$ across S stages, and BECTStage(·) denotes the sequential application of all BECT blocks of a stage (each block being defined by Eqs. (5)–(7)), then the representation is updated as:

$$\mathbf{Z}^{(s)} = \text{BECTStage}\left(\mathbf{Z}^{(s-1)}\right), \quad s = 1, \ldots, S, \tag{14}$$

with each stage composed of L_s consecutive BECT blocks. By progressively refining representations, the network develops both shallow and deep abstractions of flow-image features. After the final stage, Layer Normalization is applied, and global average pooling across tokens yields a compact embedding:

$$\mathbf{z} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{Z}_i^{(S)}. \tag{15}$$

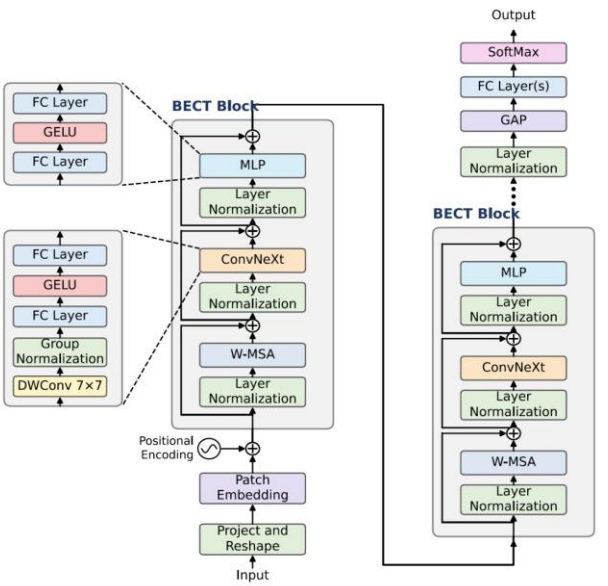

Figure 1
The architecture of the proposed BECT

This pooled feature vector is then passed to a fully connected layer for prediction:

$$\hat{\mathbf{y}} = \mathbf{W}_c\mathbf{z} + \mathbf{b}_c, \quad \hat{\mathbf{y}} \in \mathbb{R}^C, \tag{16}$$

where C is the number of output classes, and W_c and b_c are the weight matrix and bias vector of the classification layer (the subscript c stands for "classifier" and is distinct from the number of classes C). To enable deep representation learning, the BECT architecture integrates patch-level embedding, windowed self-attention, convolutional refinement, and MLP-based feature expansion within a single block and stacks these blocks hierarchically. Through the interaction of local and global modelling, BECT captures the complex spatial–temporal signatures of flow images, which makes it an efficient platform for intrusion detection. The proposed network design is depicted in Figure 1.

2.3 *Adversarial–Self-Paced Training Strategy*

Training deep intrusion detection models under adversarial conditions requires robustness to perturbations and resistance to premature overfitting to overly noisy or challenging examples. To address this, we propose a hybrid adversarial self-paced training method, which combines curriculum-based learning through self-paced weighting with adversarial robustness through PGD. This hybrid method progressively exposes the training process to more challenging samples, which, in addition to increasing resistance to evasion attacks, ensures stable convergence. We adopt PGD-based adversarial training [29] to generate perturbed flows that mimic adversarial manipulation. Given an input flow x and its label y, adversarial examples x^{adv} are generated iteratively as:

$$x_{adv}^{t+1} = \Pi_{B_\epsilon(x)}\left(x_{adv}^{t} + \alpha \cdot \mathrm{sign}\left(\nabla_x \mathcal{L}(f_\theta(x_{adv}^{t}), y)\right)\right), \tag{17}$$

where $\Pi_{B_\epsilon(x)}(\cdot)$ denotes the projection onto the ℓ_∞ ball of radius ϵ centered at the clean input x, α is the step size, and $\mathcal{L}$ is the task loss (weighted cross-entropy). Network intrusion detection datasets are inherently imbalanced. To avoid distorting the original data distribution and creating an unrealistic evaluation environment, no undersampling, oversampling, or synthetic data generation is performed. Instead, the imbalance is handled during training through a class-weighted cross-entropy loss. The class weights decrease with class frequency, thus assigning a higher penalty to the misclassification of minority classes. The model therefore learns from naturally imbalanced data, reduces bias toward majority classes, and preserves the original data characteristics. Accordingly, the adversarial training objective is defined as:

$$\mathcal{L}_{adv}(\theta) = \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\max_{\|x^{adv}-x\|_\infty \le \epsilon} \mathcal{L}(f_\theta(x^{adv}), y)\right]. \tag{18}$$

Here, $\mathcal{D}$ denotes the distribution of the training data. Not all training samples are equally informative, since some may be noisy, mislabeled, or genuinely difficult. To address this, we introduce self-paced weights [30],

[31] that dynamically adjust the contribution of each sample according to its difficulty. Let $\hat{y} = f_\theta(x)$ be the predicted label distribution and $c(x) = \hat{y}_y$ denote the confidence in the ground-truth class. A sample is considered easy if it is predicted correctly with high confidence ($c(x) > \tau$), and hard otherwise. We define the adaptive sample weights as follows, where arg max $\hat{y}$ denotes the index of the largest component of $\hat{y}$ (i.e., the predicted class):

$$w(x, y; t) = \begin{cases} w_{\text{easy}}(t) & \text{if } \operatorname{argmax}\hat{y} = y \text{ and } c(x) > \tau, \\ w_{\text{hard}}(t) & \text{otherwise,} \end{cases} \tag{19}$$

where t is the training epoch, $w_{\text{easy}}(t)$ decreases with training progression to reduce over-reliance on trivial samples, and $w_{\text{hard}}(t)$ increases to emphasize difficult or adversarial cases progressively. Specifically, we define:

$$w_{\text{easy}}(t) = \max\left(1 - \frac{t}{T},\, 0.1\right), \quad w_{\text{hard}}(t) = \min\left(\frac{t}{T},\, 1.0\right), \tag{20}$$

with T denoting the total number of training epochs. Because the weights depend both on the epoch and on each sample's current confidence, the resulting schedule adapts to the model's learning state rather than to time alone; we therefore refer to it as the adaptive schedule. Finally, the clean and adversarial losses are integrated into a single objective:

$$\begin{aligned} \mathcal{L}_{\text{total}}(\theta) = (1 - \lambda)\, \mathbb{E}[w(x, y; t) \cdot \mathcal{L}(f_\theta(x), y)] \\ + \lambda\, \mathbb{E}[w(x, y; t) \cdot \mathcal{L}(f_\theta(x^{adv}), y)], \end{aligned} \tag{21}$$

where $\lambda \in [0,1]$ controls the trade-off between clean accuracy and adversarial robustness. Combined with the self-paced weights, training starts with examples that the model classifies with high confidence and progressively moves to more difficult ones. This procedure improves the model's ability to identify normal activity and strengthens its protection against adversarial attacks, both of which are important in real-world intrusion detection.

2.4 *Dataset*

The data used in this study are the raw PCAP files of the Canadian Institute for Cybersecurity Intrusion Detection Evaluation Dataset (CICIDS2017). CICIDS2017 [32] is a widely used benchmark owing to its realistic traffic generation and its exhaustive coverage of attack types. Unlike existing studies that use pre-computed statistical flow features (e.g., the 80 CICFlowMeter features), the proposed framework does not use any handcrafted features; the raw packet-level traffic is processed and reassembled into bidirectional flows. Each flow is defined as a two-way communication session between a pair of endpoints identified by source IP, destination IP, source port, destination port, and transport protocol. To ensure a realistic deployment scenario and prevent temporal information leakage, all flows are chronologically sorted according to packet timestamps before any preprocessing operation. A time-based split is then

performed, where the first 70%, the next 10%, and the final 20% of the flows are allocated for training, validation, and testing, respectively. This strategy simulates real-world operational conditions in which a model is trained on historical traffic and evaluated on future unseen traffic. No random shuffling is applied prior to dataset splitting. Because the attacks in CICIDS2017 were generated on specific days, the test period may contain attack types that are rare or absent in the training period; since the task is binary (benign vs. attack), the reported results therefore also reflect the model's ability to generalise to attack traffic not seen during training. Furthermore, all preprocessing steps, including flow construction, truncation, padding, resizing, and normalization, are performed strictly after the chronological split. Any statistical parameters required during preprocessing are computed exclusively on the training set and subsequently applied to the validation and test sets to avoid data leakage. For computational tractability and architectural compatibility, each flow is truncated or zero-padded to a maximum length of 2500 bytes and reshaped into a two-dimensional grayscale representation. This representation enables the use of vision-based deep architectures, including the proposed BECT, and maintains the structural features of the raw network traffic.

2.5 *Evaluation Metrics*

To rigorously analyze the performance of BECT, several measures are used. In this study, intrusion detection is formulated as a binary classification task (benign vs. attack). Because the classes are imbalanced, accuracy alone is not sufficient; we therefore report accuracy, precision, recall, and F1-score to offer a broad evaluation, together with the confusion matrix, which provides a summary of prediction outcomes.

2.6 *Experimental Settings*

All experiments were conducted within a homogeneous parameter regime to ensure reproducibility and stable training of the proposed BECT-based intrusion detection system. The model was trained for 80 epochs with a batch size of 64 using the AdamW optimiser with a learning rate of 5.0×10^{-4}, a five-epoch warm-up, and cosine decay over the remaining epochs. The network was regularised using a weight decay of 0.05 to reduce overfitting, and the contributions of easy and hard samples were balanced by setting the confidence threshold τ in Eq. (19) to 0.5. To achieve adversarial robustness, projected gradient descent (PGD) was used to generate perturbed flow inputs during training. The size of the perturbation, ϵ, was kept between 0.002 and 0.008 so that the adversarial manipulations remained imperceptible, and a PGD step size of 0.00625 was used for the iterative updates. Because each iterate is projected back onto the $\ell\infty$ ball of radius ϵ (Eq. (17)), a step size larger than ϵ drives the perturbation to the boundary of the feasible set in the first iteration, and the

remaining iterations refine its direction; the perturbation magnitude never exceeds ϵ. This configuration allowed the model to train stably on both benign and adversarial examples, learning appropriate parameters and improving its detection capability. The resulting experimental design provides a sound benchmark for training the proposed BECT framework under a wide variety of network traffic conditions. Each experiment was repeated five times with different random seeds to reduce bias.

Table 1 summarizes the BECT configuration used in all experiments. The 4 × 4 convolutional patch embedding with stride 4 maps each 50 × 50 flow image to a 12 × 12 grid of N = 144 tokens (the last two rows and columns of pixels are not covered), and the 4 × 4 attention windows partition this grid into nine non-overlapping windows of 16 tokens each. The network has two stages with four BECT blocks in total, and the embedding dimension d = 64 is kept constant throughout, giving 347,458 trainable parameters.

Table 1

Architecture configuration of BECT

Parameter	Value
Input image size	50 × 50, 1 channel (grayscale)
Patch size p (convolution stride)	4 × 4 (4)
Token grid / number of tokens N	12 × 12 / 144
Embedding dimension d	64
Stages / total BECT blocks	2 / 4
Attention heads h per block	4
Head dimension d_h	16
Window size $M \times M$	4 × 4 (16 tokens; 9 windows per image)
Window shifting	Not used
ConvNeXt depthwise kernel	7 × 7
MLP expansion ratio α / dimensions	4 / 64 → 256 → 64
Activation	GELU
Residual connections per block	3
Pooling	Global average pooling
Classification layer	Linear, 64 → 2
Output classes C	2 (normal, attack)
Trainable parameters	347,458

3. Simulation Results

In this section, we report the simulation outcomes of the proposed intrusion detection system, examining its performance under multiple scenarios. The analysis focuses on the efficacy of the BECT model and on the role that adversarial self-paced training plays in improving detection

accuracy. The evaluation covers the influence of the λ parameter, detection performance under normal attack conditions, robustness against adversarial perturbations, and an ablation study to validate the contribution of the proposed training strategy. All simulations were performed in Python on a system with an Intel Core i7-13650HX CPU, 16 GB RAM, and an NVIDIA RTX 4060 GPU with 8 GB GDDR6 memory. On this hardware, training took about 21 hours, and the mean inference time was 8 ms (standard deviation 1.58 ms).

3.1 *Parameter Adjustment*

This subsection examines how various hyperparameters affect the proposed approach. Initially, the PGD iterations and perturbation bound ϵ were set at 3 and 0.005, respectively. Training, testing, and adversarial datasets were used to assess the impact of the trade-off parameter λ with λ values of [0, 0.25, 0.5, 0.75]. Figure 2 shows the corresponding classification accuracy. The training accuracy was high at all λ values, peaking at $\lambda = 0.5$ (99.84% ± 0.0503). However, $\lambda = 0.25$ (99.64% ± 0.0989) yielded the best accuracy on testing data, whereas adversarial resilience improved as λ grew, reaching 99.85% ± 0.0979 at $\lambda = 0.75$. At this setting, the adversarial accuracy slightly exceeds the clean test accuracy, most likely because a large λ fits the model mainly to perturbed inputs, which then resemble the adversarial evaluation set more closely than clean traffic. Since the clean test accuracy is the primary criterion and $\lambda = 0.25$ still retains strong adversarial robustness, it offers the best compromise between the two objectives; as a result, it was chosen for further testing.

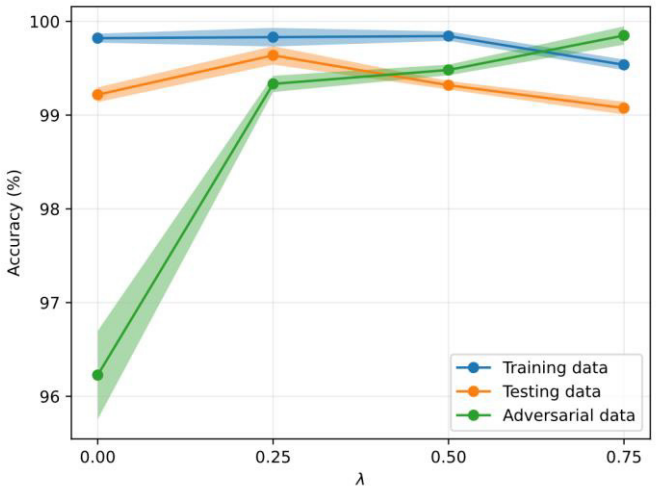

Figure 2
Evaluation of the performance for different λ values

The effect of the perturbation bound ϵ on model performance is summarized in Table 2. As ϵ increased, adversarial resilience improved, while training and testing accuracies declined slightly. The maximum adversarial resilience was attained at $\epsilon = 0.008$ (99.60% ± 0.0690), whereas the highest clean testing accuracy was attained at $\epsilon = 0.002$ (99.69% ± 0.0750). In subsequent experiments, $\epsilon = 0.005$ was chosen because it offered the optimum trade-off between adversarial resistance and clean performance.

Table 2

The effect of the perturbation bound on model performance

	ε = 0.002		**ε = 0.005**		**ε = 0.008**	
	Mean	**Std**	**Mean**	**Std**	**Mean**	**Std**
Training data	99.85	0.0681	99.83	0.0994	99.71	0.0885
Testing data	99.69	0.0750	99.64	0.0989	99.44	0.1011
Adversarial data	98.93	0.1216	99.33	0.0870	99.60	0.0690

3.2 *Normal Attack Detection Results*

This subsection assesses the proposed BECT-based intrusion detection model in normal testing scenarios. The run closest to the mean accuracy across five independent runs is reported. The confusion matrix was examined to assess the model's discriminative capacity. As Figure 3 shows, the model correctly detected 24,806 attack flows and 64,884 normal flows, with only 147 false positives and 163 false negatives. These results confirm strong class separation, a low false-alarm rate, and efficient attack detection, with a good trade-off between sensitivity and specificity.

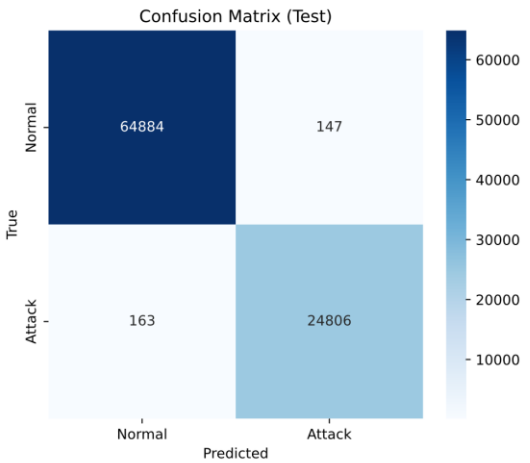

Figure 3

Confusion matrix on the clean test set

3.3 *Adversarial Attack Detection Results*

In this subsection, the proposed BECT model is evaluated on adversarially perturbed traffic flows that are intended to deceive intrusion detection systems with minute disturbances. As before, the confusion matrix is reported for the run closest to the mean accuracy of five independent runs. The model correctly detected 64,740 out of 65,031 normal samples and 24,654 out of 24,969 attack samples under adversarial settings, with only 291 false positives and 315 false negatives, as Figure 4 illustrates. With low false-positive and false-negative rates, these findings show strong resilience to adversarial perturbations. By mixing clean and adversarial samples throughout training, the adversarial self-paced learning technique enhanced resilience, producing robust decision boundaries and sustaining good detection performance under adversarial conditions.

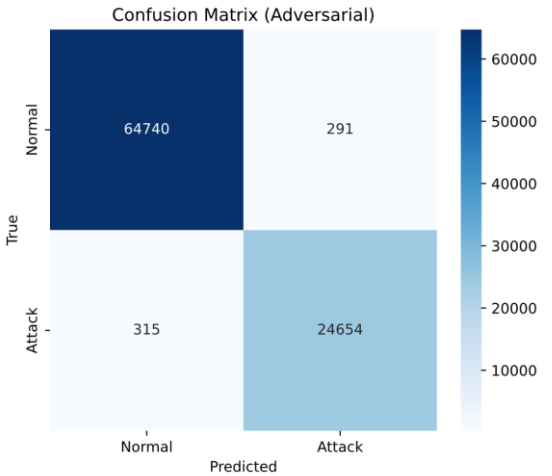

Figure 4
Confusion matrix on the adversarial test set

3.4 *Ablation Study*

Validation was conducted through an ablation study comparing four training paradigms: (i) adversarial self-paced training (the proposed one), (ii) adversarial training, (iii) self-paced training, and (iv) normal training without either enhancement. Adversarially perturbed inputs were used to assess model robustness. The results of five independent runs are summarized in Table 3. The proposed method achieved an accuracy of 99.33% ± 0.0593, a precision of 99.33% ± 0.0871, a recall of 99.27% ± 0.0847, and an F1-score of 99.29% ± 0.0355. By comparison, adversarial training alone achieved an accuracy of 99.11% ± 0.1127, a precision of 99.14% ± 0.0903, a recall of 99.05% ± 0.0260, and an F1-score of 99.07% ±

0.0241. Although adversarial training alone is already robust, the added self-paced mechanism provides more orderly and stable learning, further refining the decision boundaries and reducing misclassifications.

A second ablation study, carried out under the same experimental conditions, tested the contribution of the individual architectural components of the BECT framework. Four backbone architectures were considered: (i) convolution-based (convolutional feature extractor), (ii) transformer-based (global self-attention), (iii) convolution and global self-attention, and (iv) the proposed BECT architecture that combines convolution with window-based self-attention. For a fair comparison, all models used the same optimization parameters, perturbation limits and class-weighted loss. Both clean and adversarial test sets were evaluated, and Table 4 reports the mean and standard deviation over five independent runs.

Three pacing strategies were also compared under the same adversarial training conditions to validate the choice of schedule used in the self-paced learning mechanism: (i) the linear schedule, (ii) the exponential schedule, and (iii) the proposed adaptive schedule of Eqs. (19)–(20). Each strategy introduces increasingly difficult samples but differs in how quickly their weight grows. Table 5 reports the mean and standard deviation over five runs. These results show that the adaptive pacing scheme successfully balances adversarial robustness and curriculum learning, allowing the model to be exposed to hard samples without upsetting the optimization procedure. The findings confirm that the pacing function is an important ingredient for optimal generalization and robustness.

Table 3
The ablation study results for different training approaches

Training approach	Accuracy		Precision		Recall		F1-score	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std
Adversarial self-paced training	99.33	0.0593	99.33	0.0871	99.27	0.0847	99.29	0.0355
Adversarial training	99.11	0.1127	99.14	0.0903	99.05	0.0260	99.07	0.0241
Self-paced training	96.23	0.0679	96.23	0.0841	96.20	0.0542	96.21	0.0386
Normal training	95.73	0.0422	95.73	0.0503	95.66	0.0499	95.68	0.0487

Table 4
The architectural ablation study results

Model	Clean accuracy		Adversarial accuracy	
	Mean	Std	Mean	Std
ConvNeXt-only	99.42	0.08	98.72	0.11
Transformers-only (Global Attention)	99.16	0.12	98.97	0.14
ConvNeXt + Global Self-Attention	99.60	0.07	99.26	0.09
Proposed BECT (Window-Based Attention)	99.64	0.06	99.33	0.06

Table 5
Ablation study of different training schedules

Schedule type	Clean accuracy		Adversarial accuracy	
	Mean	Std	Mean	Std
Linear	99.52	0.09	99.11	0.12
Exponential	99.48	0.14	99.19	0.15
Proposed adaptive (Eqs. (19)–(20))	99.64	0.06	99.33	0.06

4. Discussion

The proposed Block-Embedded ConvNeXt Transformer (BECT) architecture, coupled with adversarial self-paced training, demonstrates that jointly modeling local byte-level patterns and long-range contextual dependencies can substantially improve both detection accuracy and robustness in intrusion detection systems. In contrast to prior CNN-based approaches that mainly focus on local spatial features, and LSTM or transformer-based methods that emphasize temporal or global context, our results indicate that tightly integrating ConvNeXt-style convolutional refinement with window-based self-attention yields consistent gains in clean and adversarial settings. This aligns with earlier reports that hybrid deep learning frameworks (e.g., CNN–LSTM, CNN–GRU, attention-augmented models) often outperform single-architecture baselines, but extends them by showing that a carefully designed block-level fusion with localized attention can further reduce vulnerability to adversarial perturbations. Moreover, the adversarial self-paced training procedure, which combines PGD-generated examples with a curriculum-style progression from easy to harder samples, addresses a limitation of many robust IDS methods that either rely solely on adversarial training or on scheduling without explicitly accounting for crafted perturbations. The observed improvements in stability and robustness support the working hypothesis that curriculum-inspired weighting can regularize adversarial training and mitigate overfitting to specific attack patterns. Taken together,

these findings suggest that robust IDS design should consider both architectural hybridity and training dynamics. This study has three main limitations. First, the evaluation is restricted to a single dataset (CICIDS2017). Second, the comparison is limited to internal ablation baselines, because most published IDS results are reported on pre-computed CICFlowMeter features with random train–test splits and are therefore not directly comparable with the raw-byte, chronologically split protocol used here. Third, packet headers, including IP addresses and port numbers, were retained in the raw bytes; because some hosts in CICIDS2017 consistently act as attackers or victims, the model may partly exploit host identifiers rather than traffic behaviour. Future work will therefore mask header fields before image construction, re-evaluate published IDS methods under the same protocol, and explore multi-class attack categorization, cross-dataset generalization, and deployment-oriented issues such as latency, resource constraints, and online adaptation in large-scale, real-time network environments.

5. Conclusions

This paper proposes a hybrid architecture, the Block-Embedded ConvNeXt Transformer (BECT), which combines convolutional refinement with transformer-based attention for accurate and effective intrusion detection. To further increase robustness to adversarial perturbations and stabilise training, an adversarial self-paced training policy was used that combines projected gradient descent (PGD) adversarial training with curriculum-based weighting. The proposed framework was validated through extensive simulation experiments. The preprocessing pipeline has been effectively used to convert heterogeneous raw network traffic into image representations suitable for feature extraction. The BECT backbone proved effective at capturing both local and global dependencies, and the adversarial self-paced training had a significant beneficial effect on meeting the challenge of crafted perturbations. The results show that the proposed solution achieves near-perfect detection rates in nominal settings and is highly reliable when faced with adversarial conditions. Moreover, the ablation studies supported the idea that the synergistic interaction of adversarial and self-paced learning yields consistent performance gains compared with either method alone or with standard training. Overall, the BECT framework provides a principled, high-performance solution that achieves a balance of discriminative accuracy, robustness, and adaptability under adversarial conditions. Future research will extend the approach to multi-class attack classification, cross-dataset validation and real-time implementation across large-scale network infrastructure.

Declarations

All authors declare that they have no conflicts of interest.

References

[1] P. Wasnik and N. Chavhan, "Designing intelligent intrusion detection system using deep learning," in *Proc. 14th Int. Conf. Comput. Commun. Netw. Technol. (ICCCNT)*, pp. 1–9 (2023).

[2] S. Al-Emadi, A. Al-Mohannadi, and F. Al-Senaid, "Using deep learning techniques for network intrusion detection," in *Proc. IEEE Int. Conf. Informat., IoT, Enabling Technol. (ICIoT)*, pp. 171–176 (2020).

[3] P. Chellammal, S. K. Malarchelvi, K. Reka, and G. Raja, "Fast and effective intrusion detection using multi-layered deep learning networks," *Int. J. Web Services Res.*, vol. 19, no. 1, pp. 1–16 (2022).

[4] A. R. Khan, M. Kashif, R. H. Jhaveri, R. Raut, T. Saba, and S. A. Bahaj, "Deep learning for intrusion detection and security of Internet of Things (IoT): Current analysis, challenges, and possible solutions," *Secur. Commun. Netw.*, vol. 2022, pp. 1–23 (2022).

[5] A. Awajan, "A novel deep learning-based intrusion detection system for IoT networks," *Computers*, vol. 12, no. 2, Art. no. 34 (2023).

[6] M. A. Agalit and Y. Idrissi Khamlichi, "Optimization of intrusion detection with deep learning: A study based on the KDD Cup 99 database," *Int. J. Saf. Secur. Eng.*, vol. 14, no. 4, pp. 1029–1038 (2024), doi: 10.18280/ijsse.140402.

[7] J. Jose and D. V. Jose, "Deep learning algorithms for intrusion detection systems in Internet of Things using CIC-IDS2017 dataset," *Int. J. Electr. Comput. Eng.*, vol. 13, no. 1, pp. 1134–1141 (2023).

[8] P. Ananthi, K. Nirmaladevi, and N. Kumar, "Intrusion detection mechanism using deep learning," in *Proc. Int. Conf. IoT Based Control Netw. Intell. Syst. (ICICNIS)*, pp. 188–194 (2024).

[9] A. Odeh and A. Abu Taleb, "Ensemble-based deep learning models for enhancing IoT intrusion detection," *Appl. Sci.*, vol. 13, no. 21, Art. no. 11985 (2023).

[10] P. Naresh, P. Srinath, K. Akshit, M. S. S. Raju, and P. VenkataTeja, "Decoding network anomalies using supervised machine learning and deep learning approaches," in *Proc. 2nd Int. Conf. Autom., Comput. Renewable Syst. (ICACRS)*, pp. 1598–1603 (2023).

[11] R. Jablaoui and N. Liouane, "An effective deep CNN-LSTM based intrusion detection system for network security," in *Proc. Int. Conf. Control, Autom. Diagnosis (ICCAD)*, pp. 1–6 (2024).

[12] B. AlOmar, Z. Trabelsi, and F. Saidi, "Attention-based deep learning modelling for intrusion detection," in *Proc. Eur. Conf. Cyber Warfare Secur. (ECCWS)* (2023).

[13] P. Shrivastava and R. Yadav, "Deep learning approach for intelligent intrusion detection system," *SSRN Electron. J.* (2023).

[14] N. Yadav, S. Pande, A. Khamparia, and D. Gupta, "Intrusion detection system on IoT with 5G network using deep learning," *Wireless Commun. Mobile Comput.*, vol. 2022, pp. 1–11 (2022).

[15] V. Hnamte and J. Hussain, "Dependable intrusion detection system using deep convolutional neural network: A novel framework and performance evaluation approach," *Telematics Informat. Rep.*, vol. 11, Art. no. 100077 (2023).

[16] T. Pardeshi, D. Vekariya, and A. Gandhi, "Exploring the synergy of GAN and CNN models for robust intrusion detection in cyber security," in *Proc. 3rd Int. Conf. Innov. Mech. Ind. Appl. (ICIMIA)*, pp. 1470–1475 (2023).

[17] V. Takale, A. Patil, A. Lonikar, A. Shinde, and V. Rupnar, "SecureNet: Network intrusion detection using machine learning and deep learning techniques," *Int. J. Res. Appl. Sci. Eng. Technol.*, vol. 12, no. 4 (Apr. 2024).

[18] S. L. Jacob and P. S. Habibullah, "A systematic analysis and review on intrusion detection systems using machine learning and deep learning algorithms," *J. Comput. Cogn. Eng.*, vol. 4, no. 2, pp. 108–120 (2025).

[19] M. Bhavsar, K. Roy, J. Kelly, and O. Olusola, "Anomaly-based intrusion detection system for IoT application," *Discover Internet Things*, vol. 3, no. 1, Art. no. 5 (2023).

[20] B. Madhu, M. V. G. Chari, R. Vankdothu, A. K. Silivery, and V. Aerranagula, "Intrusion detection models for IoT networks via deep learning approaches," *Measurement: Sensors*, vol. 25, Art. no. 100641 (2023).

[21] P. B. Jadhav, "Intrusion detection using IoT with deep learning," *IJSAT—Int. J. Sci. Technol.*, vol. 16, no. 3 (2025).

[22] A. A. Yılmaz, "A novel deep learning-based framework with particle swarm optimisation for intrusion detection in computer networks," *PLoS ONE*, vol. 20, no. 2 (2025).

[23] Y. Zeng, "CSAGC-IDS: A dual-module deep learning network intrusion detection model for complex and imbalanced data," arXiv:2505.14027 (2025).

[24] A. A. A. Mohammed, "Improving intrusion detection systems by using deep learning methods on time series data," *Eng. Technol. Appl. Sci. Res.*, vol. 15, no. 1, pp. 19267–19272 (2025).

[25] W. Wang, M. Zhu, X. Zeng, X. Ye, and Y. Sheng, "Malware traffic classification using convolutional neural network for representation learning," in *Proc. Int. Conf. Inf. Netw. (ICOIN)*, pp. 712–717 (2017).

[26] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16 × 16 words: Transformers for image recognition at scale," in *Proc. Int. Conf. Learn. Represent*. (ICLR), (2021).

[27] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, "A ConvNet for the 2020s," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, pp. 11976–11986 (2022).

[28] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin Transformer: Hierarchical vision transformer using shifted windows," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, pp. 10012–10022 (2021).

[29] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, (2018).

[30] M. P. Kumar, B. Packer, and D. Koller, "Self-paced learning for latent variable models," in *Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 23, pp. 1189–1197 (2010).

[31] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proc. 26th Int. Conf. Mach. Learn. (ICML)*, pp. 41–48 (2009).

[32] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *Proc. 4th Int. Conf. Inf. Syst. Secur. Privacy (ICISSP)*, pp. 108–116 (2018).

[33] A. B. Kathole, K. Vhatkar, G. Dharmale, S. Chiwhane, V. V. Kimbahune, and A. Goyal, "A novel approach to IoT security for intrusion detection system using ensemble network and heuristic-assisted feature fusion," *J. Discrete Math. Sci. Cryptogr.*, vol. 27, no. 7, pp. 2207–2217 (2024), doi: 10.47974/JDMSC-2092.

[34] J. S. Nixon, S. K. Gorva, S. Solanki, D. K. J. B. Saini, S. N. Prajwalasimha, and P. Tiwari, "Privacy-preserving federated learning for cyber threat intelligence in industrial systems with blockchain security," *J. Inf. Optim. Sci.*, pp. 1–10 (2026), doi: 10.47974/JIOS-2203.

Received June, 2026