ijsigma.org

SIGMA International Journal of Security, Intelligence, Governance & Modern AI Computing

VOLUME 01 · ISSUE 01 · 2026 · ARTICLE 0002

RESEARCH ARTICLE OPEN ACCESS · ISSN: [ISSN to be assigned] · DOI: [DOI to be assigned]

A Leakage-Safe Evaluation Framework for Gated Autoencoder–Transformer Multi-Class Network Intrusion Detection

Azhar Javed1, Naik Nawaz2,* and Asher Mehfooz3

1 Information Security Department, National University of Sciences and Technology (NUST), Islamabad, Pakistan

2 MSCS, National University of Sciences and Technology (NUST), Pakistan

3 Department of Data Science, University of Kotli (UOK), Azad Jammu and Kashmir, Pakistan

Corresponding author: Naik Nawaz (naik.021nawaz@gmail.com) · ORCID: [0000-0000-0000-0000]

Author emails: Azhar Javed (ajaved.phdismcs@student.nust.edu.pk)

ABSTRACT

Expressive representation learning together with leakage-safe evaluation is required to achieve reliable multi-class network intrusion detection. In this paper, we introduce a leakage-safe evaluation protocol for a gated autoencoder–Transformer architecture applied to seven traffic groups in the CICIDS2017 dataset. We use the partitioning method as an experimental condition: S1 reuses the stratified reference partitioning, S2 places exact duplicates into one partition and audits near duplicates, and S3 uses whole capture sessions or time partitions when a seven-class scenario is feasible. No preprocessing (filtering, imputation, scaling, class balancing, autoencoder training) is allowed on any test data. The provided architectural baseline achieves 99.61 ± 0.05% macro-F1 on a balanced, stratified subset of the data, but we could not verify its bibliographic history or duplicate/group-disjoint proof; we retain those figures as reference results, not as the outcome of the current framework. Contribution: An audit-ready protocol for separating the effects of architecture from data leakage via reference baselines, ablation studies, per-class metrics, and a reproducibility report.

Keywords: Network intrusion detection; multi-class classification; CICIDS2017; autoencoder; Transformer; gated feature fusion; data leakage; duplicate-aware evaluation.

Highlights

- Expressive representation learning together with leakage-safe evaluation is…
- In this paper, we introduce a leakage-safe evaluation protocol for a gated…
- We use the partitioning method as an experimental condition: S1 reuses the…

IMPACT STATEMENT

Expressive representation learning together with leakage-safe evaluation is required to achieve reliable multi-class network intrusion detection. In this paper, we introduce a leakage-safe evaluation protocol for a gated autoencoder–Transformer architecture applied to seven traffic groups in the CICIDS2017 dataset. We use the partitioning method as an experimental condition: S1 reuses the stratified reference partitioning, S2 places exact duplicates into one partition and audits near duplicates, and S3 uses whole capture sessions or time partitions when a seven-class scenario is feasible. No preprocessing (filtering, imputation, scaling, class balancing, autoencoder training) is allowed on any test data. The provided architectural baseline achieves 99.61 ± 0.05% macro-F1 on a balanced, stratified subset of the data, but we could not verify its bibliographic history or duplicate/group-disjoint proof; we retain those figures as reference results, not as the outcome of the current framework. Contribution: An audit-ready protocol for separating the effects of architecture from data…

1 I. INTRODUCTION

2

Multi-class Intrusion Detection Systems (IDS) analyze flows and distinguish normal traffic from attack traffic. Multi-class IDS classify the attack type, so different measures are required to counter brute-force attacks, denial-of-service attacks, port scans, and web attacks. CICIDS2017 contains class-labeled data and can be used for research. However, strong results on this dataset make sense only with complete documentation of the data source, its classification, and the evaluation process [1]. Randomly splitting the records does not mean the IDS performs well on independent network traffic.

3

3.1 *A. Background*

New approaches to intrusion detection rely on representation learning along with attention-based classification. The autoencoder encodes numerical flow descriptors into latent descriptors. At the same time, a transformer learns interactions among the feature descriptors for classification. Although raw and latent descriptors may contain different information (the first describes traffic characteristics, while the second captures nonlinear relationships), a learned gate can adjust their relative importance for each flow. This approach seems promising, but complexity alone does not guarantee the architecture's superiority. Strong baseline architectures exist for tree-structured tabular data [2]. Hence, a reasonable comparison would include such an experiment.

4

4.1 *B. Problem Statement*

The intrusion-detection system's performance may seem too good if the training and test datasets are not truly independent. Any normalization, feature selection, or autoencoder fitting before splitting will influence the model's training because it will learn from the final distribution. Although preprocessing is conducted correctly, exact or almost exact copies of flows from the same capture may be located in different parts of the split sample. This makes model performance evaluation more difficult. Data leakage research identifies contamination of model development and testing as a common problem in obtaining reliable predictions [3]. This problem is especially relevant in multi-class detection, where good performance can hide mistakes on less frequent attacks.

5

5.1 *C. Research Gap*

The provided architectural citation employs the traffic features in a standard format, frozen autoencoder embeddings, gated raw-latent fusion, a Transformer classifier, and focal loss training. On a balanced subset of 13,992 records across seven classes from the CICIDS2017 dataset, it achieves 99.61 ± 0.05% macro-F1. However, the paper provides no verifiable bibliographic citation for this material, and it lacks duplicate counts and session- and time-disjoint evaluations. Research on experimental bias shows how data collection and partitioning can significantly affect interpretations of security model performance [4]. Thus, this paper treats the provided score as a

benchmark and asks whether the gated model retains its relative superiority when duplicate counts and session- and time-disjoint evaluations are considered.

Table 1: Existing evidence and the unresolved leakage-safe evaluation questions.

Evaluation issue	Evidence available	Unresolved question	Required analysis
Seven-class performance	Macro-F1 of 99.61 ± 0.05% Previously reported on the balanced stratified subset	Does the score persist with more independent test traffic?	Compare matched split protocols.
Preprocessing	Training-only fitting is described	Can the fitted objects and their input rows be independently audited?	Retain fit indices and preprocessing objects.
Duplicate flows	Cross-split counts are not supplied	Do identical or nearly identical flows cross partitions?	Exact-hash and near-duplicate audit
Capture relationships	Stratified random splitting is reported	Do related sessions or capture periods span splits?	Feasible group/time-disjoint holdouts
Minority attacks	Seven-class F1 scores are reported	How do minority-class errors change under stricter splits?	Per-class recall, PR analysis, confusion matrices
Model components	Reference-protocol ablations are reported	Do gating and focal-loss gains persist after the audit?	Matched ablations on audited partitions

Table 1 illustrates the separation between the evidence found regarding references and the unanswered questions. We have not yet done duplicate detection or time/partition-disjoint analyses for this paper. Figure 1 shows that the process starts with provenance and duplicate auditing, data partitioning, and fitting the transformation to the training data alone.

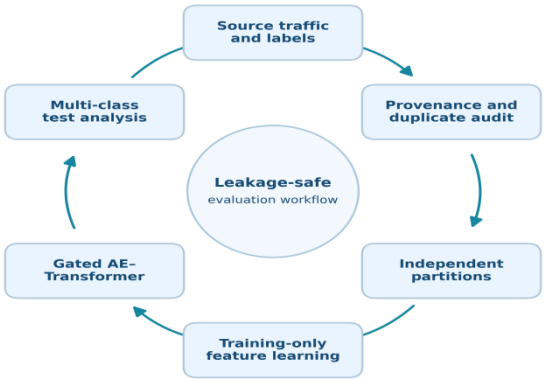


Figure 1. Evaluation methodology for a gated autoencoder-Transformer that is leakage-safe.

The six steps of the framework shown in Figure 1 are: source traffic and labeling, provenance and duplicate auditing, generation of disjoint partitions, learning transformations on training data alone, gated autoencoder-Transformer classification, and multi-class testing. This ordering matters because provenance and duplicates are checked before generating partitions, transformations are fit only on training data, validation is for model selection, and the last partition is not opened for evaluation until necessary.

5.2 D. Proposed Framework and Contributions

This article describes an auditable framework for evaluating gated autoencoder-Transformer on seven classes of CICIDS2017 in six steps: label and provenance checking, exact and near-duplicate checking, and comparing feasible session- or time-disjoint partitions with the stratified reference partition. In each partition, we train transformations only on training data, use validation data for model selection, and reserve test data for evaluation. This paper's contribution is methodological: the proposed design enables a fair comparison between baseline

classical and neural networks while examining the impact of latent representation learning, sample-dependent gating, Transformers, and focal loss across increasingly independent partitions.

5.3 E. Research Questions

This design framework centers on three testable research questions. RQ1 examines changes in total- and class-level macro-F1 and recall when moving from the less strict stratification S1 to the stricter S2 and feasible S3 partitioning. RQ2 asks whether the advantage of raw-latent gating and focal loss remains relevant as evaluation progresses from less strict to stricter partitions. RQ3 explores changes in minority-class behavior, false alerts, and efficiency as traffic independence increases.

6 II. LITERATURE REVIEW

Multi-class network intrusion detection has evolved from building classifiers based on single-flow features to latent representation learning, attention-based feature interactions, and imbalance-aware optimization. Recent studies on the role of artificial intelligence in cybersecurity have also highlighted the growing importance of machine and deep learning in automated security threat detection [5]. However, architectural complexity does not solve evaluation dependence on factors such as dataset version, class grouping and balancing, preprocessing sequence, and the relationship between training and test traffic.

6.1 A. Attention-Based Classification of Network Flows

The numeric properties of traffic flow data differ in importance across attack families. Attention captures interactions among these properties rather than weighting them equally. The self-attention mechanism was introduced in the Transformer model to establish relations between elements of the input sequence [6]. Zegarra Rodríguez et al. considered an attention-based approach to feature selection in IoT intrusion detection and obtained satisfactory results [7]. Tseng et al. investigated multi-class classification of the CIC-IoT-2023 dataset based on Transformer architecture [8]. Although these papers demonstrate the feasibility of attention-based attack-family classification, their performance measures cannot be directly compared with those obtained on the seven-group CICIDS2017 partition because of differences in traffic recording, attack definitions, class imbalance, and metrics. Therefore, this paper compares the Transformer model and other classifiers on the same CICIDS2017 partitions.

6.2 B. Minority Attacks and Representation Learning

Another limitation of multi-class intrusion detection is class imbalance, which means a classifier can be accurate when trained on the dataset but may behave less predictably on some minority attack families. Ullah et al. investigated Transfer Learning on Imbalanced Network Traffic using Transformers [9]. This line of research shows that building models under class imbalance is worth studying. However, class imbalance is a completely different problem from split leakage, so solving one does not prove the absence of the other. Thus, the proposed experiment requires testing both the model's performance on the minority classes and the integrity of the partition. Autoencoders can also enhance input data for the classifier by learning representations through reconstruction. These representations may reflect nonlinear interconnections among flow attributes. Researchers tested neural detectors based on such representations to detect and mitigate threats in enterprises [10]. Nevertheless, the reconstruction procedure may not capture all features that distinguish similar attacks. Sarıkaya et al. investigated the robust autoencoder-based intrusion-detection model in adversarial conditions [11]. The problem studied in the paper is different from that discussed in the present article.

6.3 C. Autoencoder-Transformer Combinations

Hybrid approaches leverage representation learning and attention to overcome each method's inherent deficiencies. Kamal and Mashaly proposed an autoencoder-dense-Transformer architecture for anomaly-based intrusion detection [12]. Salehiyan et al. examined a Transformer-GAN-autoencoder approach on edge and industrial IoT

datasets [13]. These examples show that learned representations can be used with further classification, but the architectural combinations differ. However, the reported results cannot isolate the effect of sample-dependent gating compared with concatenating raw data and latent representations in the current seven-class classification problem. It is necessary to have four models: raw-only, latent-only, fixed-fusion, and gated, trained and tested on the same data samples. Zhang et al. proposed a Transformer-CNN-BiLSTM combination for IoT intrusion detection [14]. Multi-component architectures can capture different feature dependencies, but their analysis is complicated because performance improvements may not reveal which component caused them when an experiment eliminates or substitutes components under equal conditions. The current article requires such an approach. The same applies to the paper's evaluation procedure: an improvement after a pre-processing step does not necessarily mean it improves the classifier.

6.4 D. Evaluation Integrity as the Remaining Gap

The reference provided is the closest architectural comparator, describing a gated autoencoder–Transformer, five stratified random seeds, tree and neural baselines, component ablations, and per-class metrics on a balanced CICIDS2017 seven-group subset. Thus, this protocol can serve as a reference protocol because its results are available. Nevertheless, it does not provide duplicate counts or the model's performance on session- or time-disjoint holdout datasets for the seven classes. This difference matters because fitting a scaler and autoencoder on the training dataset is enough to prevent learned-preprocessing leakage. Still, a separate audit is needed to determine whether duplicates or related traffic appear across partitions. Consequently, the remaining research question is not whether models need higher accuracy. It is necessary to identify whether the model advantage and recognition of minority classes reported by the authors remain the same after evaluating the same task on increasingly independent and well-documented partitions. To address this gap, three measures are possible: the amount of overlap between partitions; differences in macro-F1 and class-specific recall across split protocols; and the contribution of gating and other model components in each protocol separately. The table in Figure 2 separates these research questions. The limitation column contains information about the absence of this evidence for each specific study, not a statement about their methodological invalidity.

Table 2: Critical comparison of related work and the testable gap addressed by this article.

Research direction	Study	Dataset or task as relevant here	Main method	Evidence contributed	Unresolved issue for this article
Attentive features	Zegarra Rodríguez et al. [7]	Multiple CIC intrusion benchmarks	Attention and feature selection	Supports attentive learning on tabular traffic	No matched seven-group leakage audit
Multi-class Transformer	Tseng et al. [8]	CIC-IoT-2023	Transformer classifier	Tests IoT attack-family classification	Different dataset and class mapping
Imbalance-aware learning	Ullah et al. [9]	Imbalanced network traffic	Transformer transfer learning	Addresses minority-sensitive classification	Does not measure cross-split flow overlap
Autoencoder robustness	Sarıkaya et al. [11]	Intrusion detection under adversarial conditions	Robust autoencoder	Examines robustness of learned representations	Adversarial robustness differs from split independence
Hybrid classification	Kamal and Mashaly [12]	Anomaly-based intrusion detection	Autoencoder–dense–Transformer	Integrates latent learning and attention	Does not isolate this specific raw–latent gate on matched splits
Hybrid classification	Salehiyan et al. [13]	Edge and industrial IoT datasets	Transformer–GAN–autoencoder	Evaluates a different hybrid design	Cross-dataset scores are not a direct CICIDS2017 baseline

Multi-component attention	Zhang et al. [14]	IoT intrusion detection	Transformer and CNN–BiLSTM	Explores complementary model components	Component attribution requires matched ablations
Closest comparator	Supplied reference material (supplied; unverified bibliographic record)	Balanced CICIDS2017 seven-group subset	Gated AE–Transformer	Five-seed baseline, ablation, and per-class reference results	Duplicate and group/time-disjoint results not supplied
Present article	Proposed audited seven-group evaluation	CICIDS2017 with documented split units	Gated AE–Transformer plus leakage audit	Specifies a protocol that tests model and partition effects together	Empirical findings require the new runs

Attention, autoencoders, balance correction, and gating are each potential methods of improving intrusion detection. According to Table 2, the above-cited papers support evaluating these approaches, but their independent results do not show whether a detector generalizes to other flows in the specific capture. The current paper makes an important contribution by presenting a controlled experiment that tests detector generalizability: it maintains a seven-class problem setup, verifies duplicate and capture connections, adapts pre-processing learning with training captures, and tests the same baseline and ablated model versions across possible splits.

7 III. METHODOLOGY

This section outlines the study's evaluation methodology for a gated autoencoder–Transformer architecture for seven-class network intrusion detection. This research's key question is whether the model's superiority reported in the literature still holds when controlling for duplicate flow records, capture relationships, and preprocessing. Thus, the partition design becomes another experimental condition. In each valid condition, all baseline and ablation models share the same data partitions and seven-class label mappings, and the final test dataset is not accessible during feature fitting, hyperparameter tuning, or checkpoint selection.

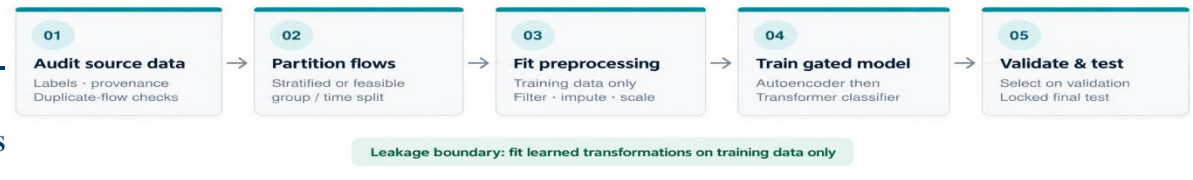
7.1 A. Research Design and Evaluation Conditions

Three evaluation conditions are described below. Condition S1 replicates the stratified random split used in the reference material. It can serve as the architectural baseline. Condition S2 leaves exact duplicates in one partition and reviews overlapping near-duplicates. Condition S3 leaves whole capture sessions or time intervals if the available metadata and seven-class labeling are feasible. Condition S3 cannot be called a seven-class evaluation if any class is missing in the held-out partition. For a valid comparison, each condition must preserve the seven-class label mapping, input feature design, architecture configuration, and main metric. The comparison should focus on performance changes due to evaluation independence, not on irrelevant published results.

Table 3: Evaluation conditions and the question answered by each condition.

Condition	Partition unit	Duplicate control	Main question
S1: Reference stratified	Individual labeled records	Measure overlap; reproduce reference setting	What does the reported style of split measure?
S2: Duplicate-group-disjoint	Groups of identical feature records	Exact duplicates confined to one partition; near duplicates audited	Does removing cross-split duplication change performance?
S3: Group/time-disjoint	Feasible capture session or period	Group and duplicate relationships confined to one partition	Does performance persist on more independent traffic?

Table 3 implements increasing independence. In S1, we use the same stratified reference split; in S2, no exact



duplicate sets cross partitions, and we audit for near duplicates; and in S3, we withhold entire capture sessions or time sets where all seven classes are present. We use the same labeling, models, and metrics to isolate partitioning's effect on performance.

Figure 2. A leakage-proofed experimental pipeline for the three partition conditions.

Figure 2 translates the framework into an operational pipeline with five steps: auditing provenance and duplicates, constructing the selected partition condition, preprocessing and fitting using training data alone, training the autoencoder and gated classifier, and validation-based selection followed by a single locked-test evaluation. The leakage barrier operates on all transformations that have been learned, namely feature selection, imputation, scaling, class balancing, and representation learning; each of these transformations is separately refit within the training segment of each split.

7.2 B. Dataset Audit and Seven-Class Mapping

CICIDS2017 is the main baseline, divided into the categories Benign, Bot/Infiltration, Brute Force, DDoS, DoS, PortScan, and Web Attack. Before modeling, the experiment documents the source release and file hashes, original labels, categorized labels, capture files, timestamps, and available session IDs. The generated manifest enables reproduction of the class mapping and partitions. The reference data includes 78 possible numeric attributes, with the baseline reference implementation selecting 49 attributes. The restricted data set includes 13,992 instances: 2,000 per category except 1,992 instances of the Bot/Infiltration category. The provided reference data does not explain why the Bot/Infiltration category has eight fewer instances than the other categories; it reports this as given, without assuming a reason. These are reference-study conditions, not automatic tallies for a fresh download release. Audit the release data and the extraction program before deciding the composition of the new study's table. Notably, the Web Attacks count in a provided source table is incompatible with its claim of having a 2,000-record controlled sample. We will create a canonical duplicate key based on the canonical representation of the candidate flow features. A secondary key will lack a label so we can find duplicates with inconsistent labels. We will measure the number of near duplicates using a similarity metric decided before examining final testing error rates.

7.3 C. Leakage-Resilient Preprocessing and Partitioning

We mark non-finite values using a documented cleaning protocol. For each partition, we learn all imputation values, the low-variance feature filter, the scaler, and the class weights from the training records alone. We then apply the fitted objects as is to the validation and test datasets. Validation determines hyperparameters and checkpointing; validation and test records are not used during autoencoder training. In an additional oversampling experiment, it applies only to the training records. The reference split conforms to the specified reference material: 35% for training, 35% for validation, and 30% for testing, using seeds 11, 22, 33, 44, and 55. For S2, the partitioning uses information on duplicate group membership. For S3, we know the selected capture/session/time and the number of independent groups per class. Temporal partitioning is infeasible for a seven-class problem because it excludes all seven classes from the test set.

Let D_{tr}, $D_{val,\ and\ Dte\ denote\ the\ training,\ validation,}$ and test partitions, and let T_{tr} denote the preprocessing transformation fitted using D_{tr} alone. The transformed inputs are

$$X_{tr}^{*} = T_{tr}(X_{tr}), \quad X_{val}^{*} = T_{tr}(X_{val}), \quad X_{te}^{*} = T_{tr}(X_{te}) \tag{1}$$

Equation (1) is the governing leakage constraint: no separate scaler, filter, or imputer is fitted to the validation or test distribution.

7.4 D. Autoencoder Feature Learning

The standardized features of the training flow i can be expressed as xi ∈ ℝd. The encoder learns the latent features zi ∈ ℝk through Eθ, and the decoder Dφ reconstructs the data:

$$z_i = E_\theta(x_i), \quad x^{*}_i = D_\varphi(z_i) \tag{2}$$

Parameters of the autoencoders are trained using mean squared reconstruction error minimized over all training flows:

$$L_{AE} = (1/N_{tr}) \Sigma_i \|x_i - \hat{x}_i\|_2^2 \quad (3)$$

The summation in (3) includes all training flows, and Ntr is the total number of training flows. We use the validation reconstruction loss to select checkpoints. For the classification task, we remove the decoder and freeze the best-performing encoder. In our baseline configuration, we set d = 49 and k = 64; because feature filtering is repeated in a new split, we must determine d for each run.

7.5 E. Sample-Dependent Gated Fusion

The vector x and the frozen autoencoder embedding z generated from (2) are concatenated as follows:

$$u_i = [x_i ; z_i] \in \mathbb{R}^{d+k} \quad (4)$$

Weights for each element of the fused vector are determined via a learned sigmoid gate:

$$g_i = \sigma(W_g u_i + b_g), \; v_i = g_i \odot u_i \quad (5)$$

In this case, $W_g \in \mathbb{R}^{(d+k)\times(d+k)}$ is the weight matrix, bg is the bias vector, σ is the element-wise sigmoid function, and ⊙ is element-wise multiplication. The gate can thus assign different weights to raw and latent data across flows. In the reference coordinates, the fused vector vi has 113 elements and is used in the Transformer classifier explained in the next section.

We evaluate the effect of gating through two ablation experiments: fixed concatenation and no-gate settings.

7.6 F. Transformer Classifier and Training Goal

The gated raw-latent embedding vi of (5) is fed into a small Transformer encoder classifier [6]. In the reference setup, the 113-dimensional gated embedding is initially linearly projected to the model width of 128 dimensions; it is unclear whether such projection takes place only once for the entire embedding or is done independently for a set of feature tokens – tokenization ambiguity as discussed below. The resulting embedding passes through two encoder layers with four attention heads and a 256-dimensional feed-forward network. The encoder extracts interactions between input features, and the subsequent classification layer predicts the traffic class the flow belongs to among seven possibilities. We first train the autoencoder, then freeze the encoder while training the gated classifier.

The classifier in the reference setup is trained using focal loss [15],

$$FL(p_t) = -\alpha_t (1 - p_t)^{\gamma} \log(p_t) \quad (6)$$

where pt stands for the probability of the predicted true class, αt is the weight of that class computed using training data, and γ is the focusing parameter. In the reference case, γ is set to 2 to pay more attention to hard examples; the class weights αt are computed only on the training-data subset; see [15]. We use the validation data to choose the checkpoint, and the test data only after selection. One aspect is not specified in the available documentation and should be mentioned in the experimental results: how exactly the combined vector is passed to the encoder as input

tokens. If the vector is treated as a single entity and used as one token, it cannot attend across different feature tokens. Therefore, the tokenization approach must be stated explicitly.

Table 4: Reference settings and planned component tests.

Item	Reference setting	Methodological check in this article
Optimizer	Adam; learning rate 0.001	Record exact optimizer configuration per run
Autoencoder	50 epochs; batch size 128	Fit on training flows only
Classifier	20 epochs; batch size 64	Select checkpoint using validation
Transformer	2 encoder blocks; 4 heads; feed-forward size 256	Record the implemented feature-token construction
Focal loss	$\gamma = 2$	Compare with gated model without focal loss
Raw features	49 retained in reference run	Refit filter within each new training partition
Latent features	64 in reference run	Compare raw-only and latent-only models
Gating	Sample-dependent sigmoid weights	Compare with fixed concatenation
Random variation	Seeds 11, 22, 33, 44, 55	Retain matched partitions and per-seed outputs

Table 4 contains the settings for the reference model and the testing plan for this study. For each data split, we train the autoencoder and feature filter on training data only, and select the classifier checkpoint based on validation data. We will use the raw-only, latent-only, and fixed-fusion models to evaluate the impact of the learned gate, and the gated model without focal loss to evaluate the impact of focal loss. We will use five seeds with matched splits to estimate run-to-run performance variation. We have yet to validate tokenization of features from the Transformer.

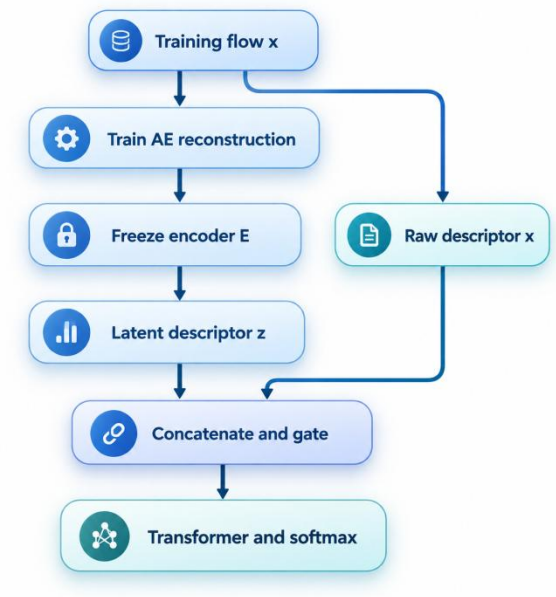


Figure 3. Raw-latent gated autoencoder-Transformer: Training and inference pipeline.

Figure 3 displays the pipeline utilized within each partition. The autoencoder is trained on the training partition. Its encoder is frozen to generate the latent feature z, and the standardized raw feature x is merged with z via the sample-specific gate before Transformer classification. This figure shows only the data flow, but the exact feature-to-token mapping construction is an implementation detail that needs documentation.

7.7 G. Performance Metrics and Statistical Analysis Plan

Macro-F1 score is the main performance metric because the main comparison treats each of the seven traffic classes equally. The secondary metrics include balanced accuracy, class-specific precision, recall, and F1, one-vs-rest precision-recall characteristics, benign false-positive rate, confusion matrices, and inference efficiency. All model comparisons will use the same partition indices and seed values. Results will include values by seed and mean ± standard deviation and, whenever possible, will compare paired models using confidence intervals estimated via

bootstrapping. Multiple ablation comparisons will involve a multiplicity correction procedure. If any class is absent in S3, a seven-class macro evaluation will not be reported.

7.8 H. Reproducibility and Leakage-Safe Acceptance Criteria

To be deemed suitable for a leakage-safety claim, four criteria need to be met: no identical group should appear across partitions S2 and S3; all transformation models should only be fit on the training set; only the validation set should determine the hyperparameter and checkpoint choice; and the test set should be sealed until testing time. In the reproducibility package, we need to maintain hashes of the source files, label map, partition files, seeds, chosen features, preprocessing objects, model configuration (including Transformer tokenization), checkpoints, prediction files, and the hardware/software timing protocol.

8 IV. DATASET AND EXPERIMENTAL SETUP

This model relies on CICIDS2017 to classify seven network intrusion detection classes. The source materials provide a balanced set of references, including the features used, split ratio, training parameters, and metrics reported earlier. Our proposed evaluation will use reference conditions (S1), duplicate-aware conditions (S2), and feasible group/time-disjointed conditions (S3). In this part, we separate the parameters described in the source materials from those required to be generated later.

8.1 A. Dataset Origin and Class Distribution

The CICIDS2017 dataset consists of labeled network flow records collected under varying traffic and attacks [1]. The provided reference information categorizes its labels as Benign, Bot/Infiltration, Brute Force, DDoS, DoS, PortScan, and Web Attacks. To enable reproducibility, the dataset record will include details on its origin and release, filenames, file hash values, extraction date, the mapping from original to grouped labels, and the number of records retained after each cleaning process. The capture date and any session identifiers will be recorded in an audit document, regardless of whether they are included in model training. The provided reference information shows a 13,992-record control dataset, whose distribution is given in Table 5 below.

Table 5: Seven-class controlled sample reported in the supplied reference material.

Grouped class	Records in controlled subset	Reported test support per seed
Benign	2,000	600
Bot/Infiltration	1,992	598
Brute Force	2,000	600
DDoS	2,000	600
DoS	2,000	600
PortScan	2,000	600
Web Attack	2,000	600
Total	13,992	4,198

According to Table 5, the reference comparison took place under conditions of approximately equal class support. This helps with component comparison in the model, but it leads to different prior classes in traffic than those used operationally. During dataset creation, one test is needed: another table in the provided information shows 218 Web Attack examples in the pre-balanced set, whereas the controlled set includes 2,000 Web Attack examples.

8.2 B. Features, Missing Values, and Categorical Variables

The provided reference material states there are 78 numerical flow descriptors as potential candidates, of which only 49 are kept using its feature selection technique. Include the names of the retained features in the reproducibility pack for the new experiment. Because the filter is trained independently for each training set, the

number of retained features is not fixed at 49; report it separately for each training-set split. The reference approach uses its 49 numerical features as an input to an autoencoder embedding model that produces 64-dimensional latent variables, resulting in a concatenated vector of 113 dimensions – raw and latent. Before training any model, validate and clean the dataset for missing fields, positive or negative infinities, invalid numeric values, constant columns, duplicated rows, and identification columns that describe the capture scenario. Each data-cleaning step should report the number of affected rows/columns. For imputing missing numeric values, compute them using the training records and apply them unchanged to the validation and test sets. The reference approach describes a dataset after cleaning its numerical features; it does not guarantee encoding of categorical predictors. If the new setup involves categorical input, build the vocabulary from the training data and define a pre-defined policy to handle unseen categories in the validation/testing datasets.

8.3 C. Splitting and Leakage Prevention

The reference materials follow a stratified split of 35% training, 35% validation, and 30% test, with seeds 11, 22, 33, 44, and 55. In each split, we create the data partition before applying the feature filter, imputer (if any), scaler, autoencoder, class weights, or classifier. We perform validation tuning and set aside test-set observations for evaluation only. Table 3 presents the three partition conditions considered in the leakage analysis. Perform exact feature hashing and label-independent duplicate detection before partitioning the data. Set the near-duplicate condition before evaluating the final test-set error. For S3, state the number of independent partitions and the class support in each partition. If S3's held-out test set does not contain all seven classes, S3 cannot be considered a seven-class comparison.

8.4 D. Scaling and Imbalance Handling Procedure

We estimate scaling parameters using only training samples. We then apply the same transformation to the validation and test sets. We train autoencoders on the training dataset; after selecting a checkpoint, we feed the encoder's frozen latent features into the classifier. Focal loss class weights are estimated using training samples, not validation and test class proportions. The reference subset has been intentionally balanced to allow controlled comparison. The new experiment should report counts for each class in the original dataset, in the clean dataset (after any sampling performed exclusively during training), and in the untouched test set. For oversampling evaluation, it will be done only within the training dataset. An unbiased test set, when supported by the source extraction, must be reported separately from the balanced diagnosis subset so that precision and false alarms are assessed under the correct class priors.

8.5 E. Hardware and Software Information

Table 4 provides information on the reference optimizer, number of epochs, Transformer configuration, focal loss application, feature dimensions, and seeds. Therefore, Table 6 focuses on reproducibility information not inferable from the provided results: the encoder and embedding dimensions used, the hardware (CPU/GPU and RAM), software and library versions, and the timing protocol (batch size, memory usage).

Table 6: Required Reproducibility and Execution Records

Component	Setting reported in the supplied reference material	Required run record
Latent representation	64 dimensions for the seven-class reference task	Encoder weights and embedding dimensions
Hardware and software	Not specified in the supplied result tables	Processor or GPU, RAM, framework and package versions
Efficiency	Per-sample inference time reported (ms/sample)	Batch size, timing method, memory, and throughput

As noted in Table 4, the implementation should also indicate the tokenization process for each feature, as the architecture diagram and description allow for multiple such processes.

9 V. RESULTS AND COMPARATIVE ANALYSIS

The following summarizes data from the provided reference literature to prepare the planned comparison. These values are not results obtained by the leakage-safe framework developed in Sections III and IV. We did not run conditions S2 and S3; therefore, we cannot present novel performance metrics or deployment parameters. Bibliographic information and restrictions regarding the provided literature are discussed explicitly in Section VI.

9.1 A. Primary Reported Results and Baselines

In the reference study, the proposed gated autoencoder-Transformer achieved 99.57 ± 0.04% accuracy and 99.61 ± 0.05% macro-F1, respectively, over five seeds. The macro-F1 value is 0.32 percentage points higher than XGBoost, but it requires about four times the per-sample inference time. Table 7 shows the three reported baselines.

Table 7: Previously reported results on balanced CICIDS2017 seven-class subset

Model	Accuracy (%)	Macro-F1 (%)	Inference (ms/sample)
Random Forest	99.22 ± 0.16	99.22 ± 0.16	0.034 ± 0.040
XGBoost	99.29 ± 0.16	99.29 ± 0.16	0.032 ± 0.037
Gated AE–Transformer	99.57 ± 0.04	99.61 ± 0.05	0.132 ± 0.009

The margin in Table 7 over tree-based baselines is small in absolute terms and comes with a higher inference cost. Because we evaluated all three models using a single stratified procedure, the table cannot reflect any change in that margin under duplicate-aware or group-disjoint splits.

9.2 B. Reported Ablation and Per-Class Performance

The ablation shows that combining raw and autoencoded features performed better than using each feature individually. From Figure 4, the raw-feature MLP scored close to 98.79% in macro-F1, while fixed raw–latent fusion scored around 99.01%; the provided document does not include a table with these two numbers, so we report them here approximately based on Figure 4. However, we can tabulate the gated variants precisely: gating without focal loss scored 99.21 ± 0.11% in macro-F1, while the full model scored 99.61 ± 0.05%.

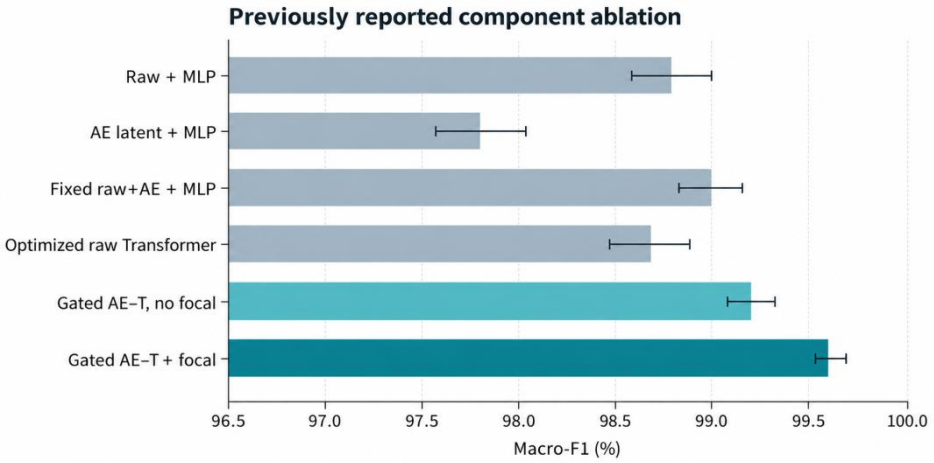


Figure 4: Component ablations found in the provided reference materials.

In Figure 4, error bars represent standard deviations based on five seeds, and the x-axis is limited to 96.5% to be able to see small variations. The first reading from the bottom up, on the weakest variation, is that of the autoencoder latent MLP, followed by the raw-only Transformer and the raw feature MLP; raw-latent concatenation outperforms all three, while the two gated variations outperform all four. The ranking is thus consistent with the assumption that the gate provides an advantage over simple concatenation. The difference between the performance of the complete model and its closest variation (without focal loss gate: 99.61% vs 99.21%) is several times larger than either variation's standard deviation. Still, the authors provide no per-seed comparison; thus, they describe the difference but do not show it is statistically significant. All seven classes achieve a reported F1 of at least 99.30%. Bot/Infiltration had the lowest recall (99.18 ± 0.15%) and the lowest F1 (99.30 ± 0.12%). Table 8 provides detailed information for each class, not just accuracy.

Table 8: Reported per-class recall and F1 of the gated AE-Transformer

Class	Recall (%)	F1 (%)
Benign	99.83 ± 0.05	99.79 ± 0.05
Bot/Infiltration	99.18 ± 0.15	99.30 ± 0.12
Brute Force	99.64 ± 0.07	99.68 ± 0.05
DDoS	99.55 ± 0.08	99.61 ± 0.06
DoS	99.86 ± 0.04	99.83 ± 0.04
PortScan	99.51 ± 0.09	99.57 ± 0.07
Web Attack	99.42 ± 0.12	99.50 ± 0.10

Across all seven traffic classes, recall and F1 scores were above 99%. The DoS class had the highest recall (99.86%). The Benign and DoS classes had the highest F1 scores of 99.83%, and the Bot/Infiltration class had the lowest recall, making it the hardest class to detect.

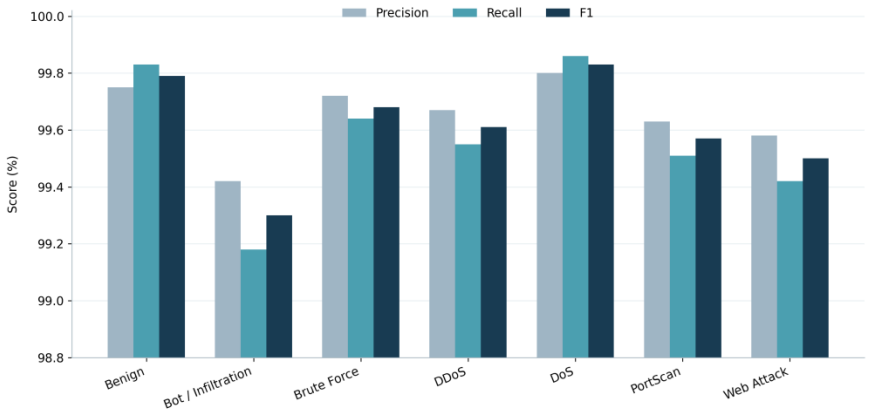


Figure 5. Precision, recall, and F1 for all seven CICIDS2017 classes in the provided reference material.

As shown in Figure 5, precision, recall, and F1 values are very high for all seven CICIDS2017 classes. However, although this is a balanced test set, the Bot/Infiltration class appears to be the weakest. That is why this class deserves attention in further research on the model's performance in duplicate-aware and group-disjoint splits. The vertical axis starts at 98.8%, so the differences look more significant. Table 8 contains recall and F1 values for each class, but not precision.

9.3 C. Requirements for Leakage-Safe Validation

Leakage-safe validation is deemed complete once the empirical package includes: (i) exact and near-duplicate audit counts for each split configuration; (ii) class support tables that check if each withheld partition supports all seven classes; (iii) baseline and ablated output at the seed level; (iv) full confusion matrices and precision-recall analysis based on probabilities from locked-test predictions; and (v) timing results from end-to-end runs on stated hardware. S2 and S3 are not intended to require performance improvements beyond S1; rather, they check whether the architectural claims hold up under greater traffic independence. Thus, a drop in performance under more restrictive splits is meaningful, not an experiment gone wrong.

9.4 D. Discussion

The reference reports a macro-F1 of 99.61% on a balanced seven-class CICIDS2017 subset. In their ablation sequence, the authors ranked the full gated autoencoder-Transformer above the raw features, latent features, fixed fusion, and no-focal-loss versions. This may be because sample-specific gating helps the classifier modulate its reliance on both raw and latent features. However, this is an interpretative conclusion without statistical significance, since we did not provide seed-level predictions. The reference results are encouraging because they

classify the seven attack groups in a controlled environment, but this does not imply equivalent performance under operational conditions. "Bot/Infiltration" has the lowest recall rate (99.18%) and should therefore be the focus of future rigorous experiments. In line with cyber threat-intelligence standards [16], a security operation center should also measure false positives under natural class-distribution conditions; the modern SOC and MITRE ATT&CK communities describe the relevant methodology [17]. More extensive research on AI in cybersecurity also highlights the distinction between benchmark performance and operational utility [18]. The reported inference time of 0.132 ms per sample is only a reference value because the authors do not document preprocessing steps, traffic collection, batching, hardware, memory, or latency.

10 VI. LIMITATIONS AND THREATS TO VALIDITY

First, the reference experiment is based on a balanced subset of 13,992 samples, which is unrepresentative of attack frequencies in a production network and, in addition, does not demonstrate performance under the duplicate-aware S2 or group/time-disjoint S3 settings because it uses a stratified train/validation/test split. Flows that may belong to each other, or even repeated records, may make random splitting simpler than with new traffic. Thus, in this paper, leakage-safe refers to the evaluation framework design rather than proven S2/S3 performance, which requires the specified audits and training-only transformations with independent holdouts. CICIDS2017 alone is not enough to show generalization to modern IoT devices, other networks, concept drift, or unseen advanced persistent threats, and grouping Bot/Infiltration labels is not a valid measurement. Another limitation related to source integrity is the following: the provided source has 99.61% macro-F1 in its tables but 99.64% in the concluding passage, and neither the author, conference/journal, year, volume, issue, page number, nor DOI could be found. The article thus uses 99.61% as a reference value, not as peer-reviewed evidence. The paper provides no figures on significance, ROC/AUC, or a seven-class confusion matrix because the predictions are not available.

11 VII. CONCLUSION AND FUTURE WORK

This work provides a leakage-safe evaluation protocol for gated autoencoder–Transformer-based multi-class network intrusion detection [19]. Rather than focusing only on high accuracy on benchmark datasets, the protocol accounts for duplicate contamination, partition independence, capture relations, and training-only preprocessing as necessary aspects of evaluation [20]. The S1-S3 protocol gradually moves from conventional stratified reference partitioning to duplicate-group-disjoint and increasingly independent group/time-disjoint evaluation while keeping class mappings, model architectures, and evaluation metrics intact. This protocol also includes raw and latent feature combinations using sample-specific gating and matches baselines, ablations, class-specific metrics, statistical methods, and reproducibility criteria [21]. The previously reported Macro-F1 score of 99.61 ± 0.05% is provided as a reference value, not as a leakage-safe result of this study. Future work will implement the full S1, S2, and S3 experimental protocol on the CICIDS2017 source data with verified partitions, paying special attention to exact and near-duplicate evaluation, class-support constraints, and the robustness of the raw-latent gated combination under increasing traffic independence. This implementation will also document the Transformer feature-to-token conversion process and test raw-only, latent-only, fixed-fusion, no-focal-loss, and complete-model variants with matched seeds and partitions. Future experiments should report class-specific precision, recall, and F1-scores; balanced accuracy; false positive rates; confusion matrices; precision-recall performance; confidence-interval comparisons; computational cost; memory consumption; and inference speed [22]. Validation on natural class distributions and external, independent IoT or network security datasets is also needed to assess robustness to distribution shifts, concept drift, or unseen attacks before claiming operational generalization [23].

REFERENCES:

[1] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, “Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization,” in *Proceedings of the 4th International Conference on Information Systems Security and Privacy*, 2018.

[2] L. Grinsztajn, E. Oyallon, and G. Varoquaux, “Why Do Tree-Based Models Still Outperform Deep Learning on Tabular Data?” *Advances in Neural Information Processing Systems*, vol. 35, 2022.

[3] S. Kaufman, S. Rosset, C. Perlich, and O. Stitelman, “Leakage in Data Mining: Formulation, Detection, and Avoidance,” *ACM Transactions on Knowledge Discovery from Data*, vol. 6, no. 4, 2012, doi: 10.1145/2382577.2382579.

[4] F. Pendlebury, F. Pierazzi, R. Jordaney, J. Kinder, and L. Cavallaro, “TESSERACT: Eliminating Experimental Bias in Malware Classification across Space and Time,” in *28th USENIX Security Symposium*, 2019.

[5] A. Ali, A. Razzaque, U. Munir, et al., “AI-Driven Approaches to Cybersecurity: The Impact of Machine and Deep Learning,” in *2024 2nd International Conference on Cyber Resilience (ICCR)*, IEEE, 2024.

[6] A. Vaswani, N. Shazeer, N. Parmar, et al., “Attention Is All You Need,” *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[7] D. Zegarra Rodríguez et al., “Attentive Transformer Deep Learning Algorithm for Intrusion Detection on IoT Systems,” *PLOS ONE*, vol. 18, no. 10, e0286652, 2023, doi: 10.1371/journal.pone.0286652.

[8] S.-M. Tseng et al., “Multi-Class Intrusion Detection Based on Transformer for Internet of Things,” *Future Internet*, vol. 16, no. 8, p. 284, 2024, doi: 10.3390/fi16080284.

[9] F. Ullah, S. Ullah, G. Srivastava, and J. C.-W. Lin, “IDS-INT: Intrusion Detection System Using Transformer-Based Transfer Learning for Imbalanced Network Traffic,” *Digital Communications and Networks*, vol. 10, no. 1, pp. 190–204, 2024, doi: 10.1016/j.dcan.2023.03.008.

[10] A. Ali, A. Zia, A. Razzaque, et al., “Enhancing Cybersecurity with Artificial Neural Networks: A Study on Threat Detection and Mitigation Strategies,” in *2024 2nd International Conference on Cyber Resilience (ICCR)*, IEEE, 2024.

[11] A. Sarıkaya, B. G. Kılıç, and M. Demirci, “RAIDS: Robust Autoencoder-Based Intrusion Detection System Model against Adversarial Attacks,” *Computers & Security*, vol. 135, p. 103483, 2023, doi: 10.1016/j.cose.2023.103483.

[12] H. Kamal and M. Mashaly, “AE-DTNN: Autoencoder–Dense–Transformer Neural Network Model for Efficient Anomaly-Based Intrusion Detection Systems,” *Machine Learning and Knowledge Extraction*, vol. 7, no. 3, p. 78, 2025, doi: 10.3390/make7030078.

[13] A. Salehiyan, P. S. Moghaddam, and M. Kaveh, “An Optimized Transformer–GAN–AE for Intrusion Detection in Edge and IIoT Systems: Experimental Insights from WUSTL-IIoT-2021, EdgeIIoTset, and TON_IoT Datasets,” *Future Internet*, vol. 17, no. 7, p. 279, 2025, doi: 10.3390/fi17070279.

[14] C. Zhang, J. Li, N. Wang, and D. Zhang, “Research on Intrusion Detection Method Based on Transformer and CNN-BiLSTM in Internet of Things,” *Sensors*, vol. 25, no. 9, p. 2725, 2025, doi: 10.3390/s25092725.

[15] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, “Focal Loss for Dense Object Detection,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2017.

[16] A. Ali and B. M. Bhatti, *Spies in the Bits and Bytes: The Art of Cyber Threat Intelligence*, 1st ed. CRC Press, 2024, doi: 10.1201/9781003504108.

[17] S. G. Virk, J. Iqbal, A. Ali, A. R. Mahmud, I. Rashid, and T. Hanif, “Advancing Security Operations Centers: Modern Use Cases, MITRE ATT&CK Integration, and Coverage Optimization in 2025,” *Journal of Computing & Biomedical Informatics*, vol. 9, no. 02, 2025.

[18] A. Ali, S. Ahmed, M. Hussain, et al., “The Synergy of Artificial Intelligence and Cybersecurity,” in *2024 2nd International Conference on Cyber Resilience (ICCR)*, IEEE, 2024.

[19] A. R. Mahmud, A. Ali, M. R. Ajmal, S. G. Virk, S. Sheikh, and M. T. Khan, "A Federated Intrusion Detection Approach for SOC-Centered APT Defense Systems," in *Proceedings of the 11th International Conference on Next Generation Computing (ICNGC)*, 2025, pp. 193–196.

[20] A. Ali, M. A. Khan, K. Farid, S. S. Akbar, A. Ilyas, T. M. Ghazal, and H. Al Hamadi, "The Effect of Artificial Intelligence on Cybersecurity," in *2023 International Conference on Business Analytics for Technology and Security (ICBATS)*, 2023, doi: 10.1109/ICBATS57792.2023.10111151.

[21] N. A. Al-Dmour, M. Kamrul Hasan, M. Ajmal, M. Ali, I. Naseer, A. Ali, H. A. Hamadi, and N. Ali, "An Automated Platform for Gathering and Managing Open-Source Cyber Threat Intelligence," in *2023 International Conference on Business Analytics for Technology and Security (ICBATS)*, 2023, doi: 10.1109/ICBATS57792.2023.10111470.

[22] S. A. Changazi, I. Shafi, K. Saleh, M. H. Islam, S. M. Hussainn, and A. Ali, "Performance Enhancement of Snort IDS Through Kernel Modification," in *2019 8th International Conference on Information and Communication Technologies (ICICT)*, pp. 155–161, 2019.

[23] S. M. Hussain, M. H. Islam, A. Ali, and M. U. Nazir, "Threat Modeling Framework for Security of Unified Storages in Private Data Centers," in *2020 IEEE 22nd Conference on Business Informatics (CBI)*, vol. 2, pp. 111–120, 2020.

Declarations

Funding. [Funder name and grant number, or a statement that no external funding was received.]

Conflicts of interest. [Relevant interests, or: The authors declare no conflicts of interest.]

Ethics and consent. [Approval reference / not applicable statement.]

Data availability. [Repository / access conditions, or not applicable.]

Code availability. [Repository / version, or not applicable.]

Author contributions. [CRediT roles and initials.]

Acknowledgments. [Optional. Delete if none.]

Use of generative AI. [Disclose AI tool use, or state that none was used.]

How to Cite This Article

SUGGESTED CITATION

A. Javed et al., "A Leakage-Safe Evaluation Framework for Gated Autoencoder–Transformer Multi-Class Network Intrusion Detection," International Journal of Security, Intelligence, Governance & Modern AI Computing (IJSIGMA), vol. 1, no. 1, pp. 34–49, 2026, doi: [DOI to be assigned].