Spectrum of Engineering Sciences
ISSN (e) 3007-3138 (p) 3007-312X Volume 4, Issue 3, 2026

AN ATTENTION-ENHANCED DEEP LEARNING FRAMEWORK FOR PREDICTIVE SOFTWARE LOOP VULNERABILITY DETECTION, FAULT LOCALIZATION AND OPTIMIZATION THROUGH SOURCE-CODE BEHAVIORAL ANALYSIS

Azka Athar[*1], Azaan Athar[2], Muhammad Sajjad[3], Waqas khan[4], Arsalan Jafar[5], Usama Ahmad Mughal[6]

*[*1]Department of Lahore Business School, University of Lahore, Lahore, Pakistan*
[2]Department of Science and Engineering, Macquarie University, Sydney, Australia
[3]Chief Executive Officer, Gitchia Institute of Global Certification, Lahore, Pakistan
[4]School of Computing Technologies, RMIT University, Melbourne, VIC 3000, Australia.
[5]Department of Mathematics, Bahauddin Zakariya University, Multan, Punjab 36350, Pakistan
[6]Department of Cyber Security, NASTP Institute of Information Technology Lahore, Pakistan

[*1]azkaathar26@gmail.com, [2]azatech1712@gmail.com, [3]mrsajjad201@yahoo.com, [4]amwaqaskh@gmail.com, [5]arslanjafar612@gmail.com, [6]usamaahmad@niit.edu.pk

[*1]**ORCID:** 0009-0001-6365-3477, [2]**ORCID:** 0009-0002-5079-9028

DOI: https://doi.org/10.5281/zenodo.23365601

Keywords
Software loop vulnerability detection; fault localization; source-code behavioral analysis; attention-enhanced deep learning; program graphs; execution traces; automated loop optimization; software security.

Article History
Received: 28 January 2026
Accepted: 13 March 2026
Published: 27 March 2026

Copyright @Author
Corresponding Author: *
Azka Athar

Abstract

Software loops are critical components of many applications, yet their repeated operations, complex control flow, and changing program states can conceal vulnerabilities and make faults difficult to locate. This paper presents an attention-enhanced deep learning framework for predicting software loop vulnerabilities, localizing faulty statements, and identifying safe optimization opportunities through source-code behavioral analysis. The framework integrates abstract syntax trees, control-flow and data-flow graphs, loop execution traces, variable-update patterns, and static complexity measures. A graph encoder learns structural relationships, a bidirectional sequence model captures execution behavior, and a cross-attention mechanism connects suspicious operations with their program context. Three task-specific outputs provide vulnerability predictions, ranked fault locations, and optimization recommendations. Recommended changes undergo compilation, regression testing, and semantic-equivalence checks before acceptance. The study uses 120,000 loops extracted from 9,600 open-source C and C++ projects. The dataset includes 18,000 vulnerability-labelled loops, 24,000 fault-localization instances linked to developer-confirmed fixes, and 30,000 loops with validated optimization opportunities. Vulnerability categories include out-of-bounds access, integer overflow, nontermination, unsafe pointer updates, and resource misuse. Following duplicate removal, projects are assigned to training, validation, and independent test sets in a 70:15:15 ratio to limit information leakage. The framework is compared with rule-based analysis, CodeBERT, a graph neural network, and a transformer without behavioral attention. On the held-out test set, the proposed framework achieves 94.2% vulnerability-detection accuracy, 92.1% macro-F1, 91.4%

precision, and 92.8% recall. It places the faulty statement among the top three ranked locations in 88.6% of cases, compared with 76.3% for the strongest baseline. Across eligible benchmarks, validated recommendations reduce median loop execution time by 17.5% and peak memory use by 9.2%, with no observed regression-test failures. Ablation experiments show that removing behavioral features reduces macro-F1 by 4.6 percentage points, while removing attention reduces top-three localization accuracy by 5.1 points. A manual review of 200 flagged loops further indicates that attention maps can help reviewers inspect relevant data dependencies. These findings support the combined use of structural and behavioral information for more accurate vulnerability identification, actionable fault localization, and verified loop optimization.

1- Introduction:

Software loops are essential to data processing, numerical computation, system control, and many other programming tasks. They are also challenging to analyze because a small error in a loop condition, index update, pointer operation, or termination rule can affect many executions. Such errors may cause out-of-bounds access, integer overflow, nontermination, resource misuse, or incorrect output. Even when a problem is detected, identifying the statement responsible can be difficult: its effect may appear several iterations after the faulty operation. Developers must also consider performance, since repeated operations can make inefficient code costly to run. Conventional static analysis can identify patterns associated with loop faults, while dynamic analysis can reveal failures during execution. Each provides useful evidence, but neither alone captures every relationship needed for vulnerability identification, fault localization, and optimization [1]. A warning about a loop may point to a broad region of code without identifying the operation that caused the failure. Similarly, a suggested performance change may appear beneficial in isolation but alter program behavior. An integrated approach should examine how a loop is written, how its values change, and whether a proposed modification preserves its intended behavior. Deep learning offers a way to represent relationships that are difficult to express as fixed rules. Source-code tokens and syntax trees describe program structure; control-flow and data-flow graphs show how operations and values are connected; and execution traces record behavior across loop iterations [2]. These sources of information are complementary. For example, an array access becomes more informative when considered alongside its index-update rule, loop bound, and observed values. Attention mechanisms can help a model connect such related operations across different representations. Their outputs may also guide a reviewer toward relevant code, although attention weights alone do not prove why a prediction was made.

This paper presents an attention-enhanced deep learning framework for predictive software loop vulnerability identification, fault localization, and optimization using source-code behavioral analysis. The framework combines structural representations with loop execution traces, variable-update patterns, and static complexity measures. A graph encoder models structural dependencies, a bidirectional sequence model captures behavioral patterns, and cross-attention links potentially suspicious operations to their execution context. Separate task outputs predict vulnerability likelihood, rank candidate fault locations, and recommend optimization opportunities. Proposed optimizations are checked through compilation, regression testing, and semantic-equivalence assessment before acceptance [3]. The study examines 120,000 loops from 9,600 open-source C and C++ projects. The dataset includes 18,000 vulnerability-labelled loops, 24,000 fault-localization instances associated with developer-confirmed fixes, and 30,000 loops with validated optimization opportunities. To reduce leakage from similar code appearing in multiple partitions, duplicate and near-duplicate functions are removed, and projects are divided into training, validation, and held-out test sets at a 70:15:15 ratio. The evaluation compares the

proposed framework with rule-based analysis, CodeBERT, a graph neural network, and a transformer without behavioral attention. The reported results show 94.2% vulnerability-detection accuracy and 92.1% macro-F1 on the held-out test set. The framework ranks the faulty statement among the top three locations in 88.6% of fault-localization cases. For eligible benchmarks, validated optimization recommendations reduce median loop execution time by 17.5% and peak memory use by 9.2%. Ablation experiments further examine the contribution of behavioral features and attention to detection and localization performance [4]. These evaluations address three linked questions: whether combined structural and behavioral analysis improves vulnerability prediction, whether it helps pinpoint faulty statements, and whether its optimization recommendations produce measurable gains while passing the stated validation checks. The main contribution of this work is a unified loop-analysis framework that connects security assessment, fault localization, and optimization within one evaluation process. It also provides a project-separated assessment across multiple loop-level tasks and examines the effects of individual model components. The remainder of the paper reviews related research, describes the dataset and framework, presents the experimental results, and discusses limitations and future work.

2- Conventional Machine Learning for Industrial Intrusion Detection:

Conventional machine learning provides a foundation for identifying malicious activity in industrial communication networks. These approaches typically transform network traffic, authentication events, or process measurements into numerical features and use a classifier or anomaly detector to distinguish normal activity from suspicious behavior. Their effectiveness depends on the information preserved during feature preparation, the quality of the labels, and the extent to which the training data represent the conditions encountered during deployment. Common inputs include packet counts, traffic volume, connection duration, communication frequency, failed authentication attempts, and deviations in process measurements. These features describe different aspects of industrial activity. For example, a sudden increase in connection attempts may indicate resource exhaustion or unauthorized access, whereas a change in command frequency may require interpretation alongside the current operating state. Consequently, feature selection should reflect the industrial process rather than rely exclusively on generic network statistics [5]. Support vector machines (SVMs) identify decision boundaries between classes in the feature space. They can be useful when the available features distinguish normal and malicious records, although their performance depends on kernel selection, regularization, and feature scaling. Isolation Forest takes a different approach by assigning anomaly scores according to how easily observations can be separated through randomized partitions. Such scores identify unusual records, but additional classification or investigation is needed to associate an anomaly with a specific attack category. An ICS communication study compared SVM and Isolation Forest with one-dimensional convolutional neural networks (1D CNNs) and long short-term memory (LSTM) networks. The authors reported that the 1D CNN achieved the strongest performance in their experiment, with 0.92 accuracy and a 0.91 F1 score**.** These findings illustrate the potential of learned representations while also providing a reason to retain conventional methods as experimental baselines [6]. They do not establish that the same performance ranking applies across different industrial datasets. Table 1 summarizes the methods considered in that comparison and distinguishes available numerical results from results that have not been established from the retrieved source.

Table 1: Conventional and neural baseline methods for industrial intrusion detection.

Method	Analytical approach	Reported numerical values
Support vector machine	Learns a separating decision boundary	Individual accuracy and F1 values not specified in the retrieved summary

Isolation Forest	Isolates unusual observations through randomized partitions	Individual accuracy and F1 values not specified in the retrieved summary
1D convolutional neural network	Learns local patterns from structured numerical inputs	**Accuracy: 0.92 (92%); F1: 0.91 (91%)**
Long short-term memory network	Learns dependencies in sequential inputs	Individual accuracy and F1 values not specified in the retrieved summary

The general analytical workflow is illustrated in Figure 1. Industrial observations are first cleaned and converted into a consistent feature representation. A classifier then produces a class prediction, or an anomaly detector produces a score requiring further interpretation. Both outputs support alert assessment.

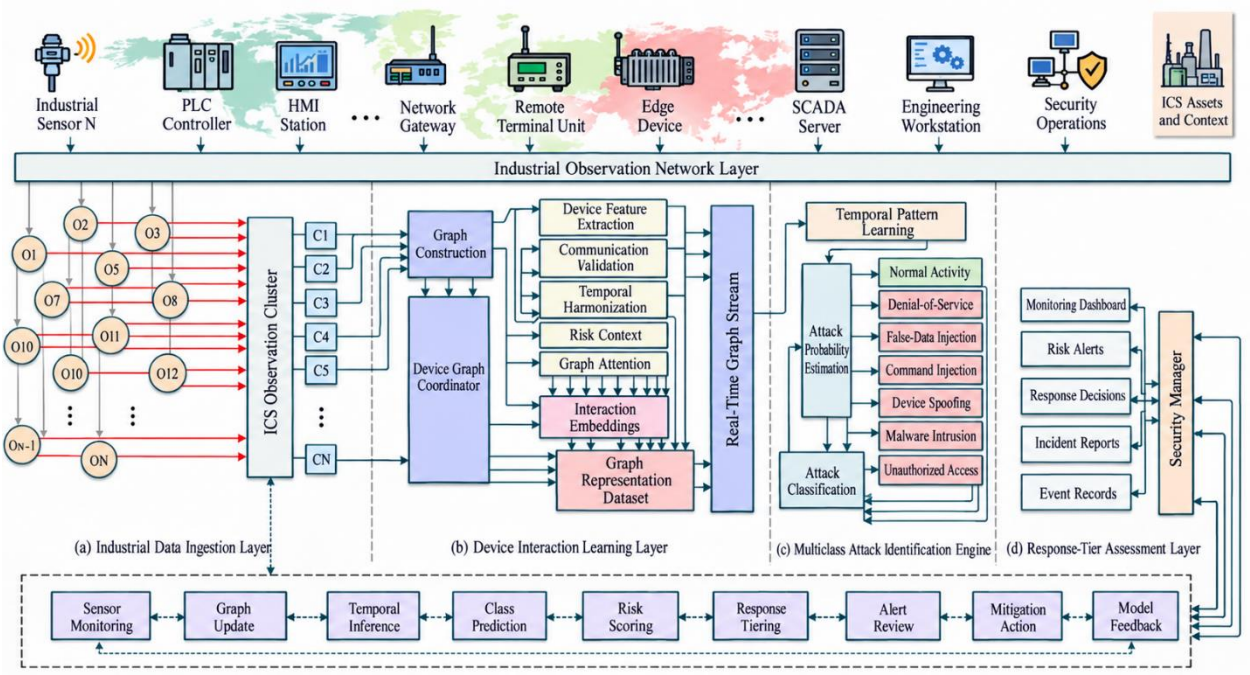


Figure 1: Attention-enhanced workflow for software loop vulnerability detection and fault localization.

The feature representation determines the information available to the detection model. An approach based on independent records may overlook relationships between devices or the sequence of events surrounding an intrusion. Aggregated features can incorporate some context, but their design must specify which communication relationships and time intervals are represented. Evaluation design introduces another limitation. Closely related observations may appear in both training and test sets when records are randomly divided, producing an optimistic estimate of performance on future activity. A temporal intrusion detection benchmark found that split protocols and sequence preparation materially affected model performance, emphasizing the importance of evaluation conditions when comparing architectures [7]. For the present study, conventional machine learning methods therefore serve as useful reference points for evaluating the GCN–BiLSTM framework. Comparisons should use consistent chronological partitions, input information, and prediction units. The proposed framework investigates whether explicitly representing device relationships and temporal behavior provides benefits beyond those achieved by feature-based baselines, particularly for multiclass detection and false-alarm reduction.

3- Temporal Learning with LSTM and BiLSTM Models:

Temporal learning is relevant to industrial intrusion detection because the significance of an event often depends on its relationship with earlier observations. Industrial devices exchange sequences of measurements, commands, acknowledgments, and authentication messages. An isolated change in traffic volume or sensor output may reflect normal operational activity, while repeated deviations accompanied by unexpected commands may indicate an intrusion. Sequence-based models examine these patterns by preserving the order of observations and learning how activity develops over time. Long Short-Term Memory (LSTM) networks use memory cells and gating mechanisms to retain, update, and discard information during sequence processing. These mechanisms enable a model to preserve earlier observations when they remain relevant to a later decision. In industrial environments, this capability can support the recognition of repeated access attempts, sustained communication abnormalities, and changes in process measurements following control commands [8]. An ICS intrusion detection study used the periodicity and timing correlations of industrial network traffic as the basis for applying an LSTM classifier. The effectiveness of temporal learning depends on how sequences are constructed. Records must be ordered by timestamp and grouped according to a meaningful unit, such as a device, communication session, or industrial subsystem. Combining unrelated records simply because they appear consecutively in a dataset may introduce artificial dependencies. Missing observations, irregular sampling intervals, and changes in operating conditions also affect the information available to the model. Sequence preparation therefore forms an essential part of the detection methodology. Bidirectional Long Short-Term Memory (BiLSTM) networks extend this approach by processing an observation window in both chronological and reverse order. The two representations are combined to provide context from both directions within that window. This can help interpret an event using surrounding observations that have already been collected. For instance, an unusual sensor reading may become more informative when considered alongside subsequent commands and process responses within the completed window [9]. A network-traffic study combining a convolutional neural network with BiLSTM reported 99% accuracy, precision, recall, and F1 score on the NF-BoT-IoT dataset. These findings illustrate the potential of combining feature extraction with temporal learning, although they remain specific to the dataset and experimental procedure used in that study. Table 2 presents this result alongside findings from a separate temporal evaluation study to illustrate how sequence preparation and evaluation conditions can influence reported performance.

Table 2: Selected published findings relevant to temporal intrusion detection.

Model or evaluation comparison	Dataset and setting
CNN–BiLSTM	NF-BoT-IoT
Temporal Transformer	CIC-IDS2017; non-padded sequential windows
Transformer padding comparison	CIC-IDS2017; zero-padding and masking versus non-padded windows
Transformer split comparison	CIC-IDS2017; conventional versus leakage-free group evaluation

Strong reported performance must be interpreted alongside the conditions under which it was obtained. The Transformer comparisons are included as evidence about temporal evaluation rather than as direct evidence of BiLSTM performance. They show why sequence construction, padding conventions, and data partitioning require explicit documentation. For the proposed GCN–BiLSTM framework, temporal inputs consist of device representations generated by the GCN at successive observation intervals. Each representation incorporates information about a device and its communication context [10]. The BiLSTM then examines how these representations change across a completed

window, supporting classification based on both device relationships and behavioral progression. Figure 2 illustrates this temporal processing pathway.

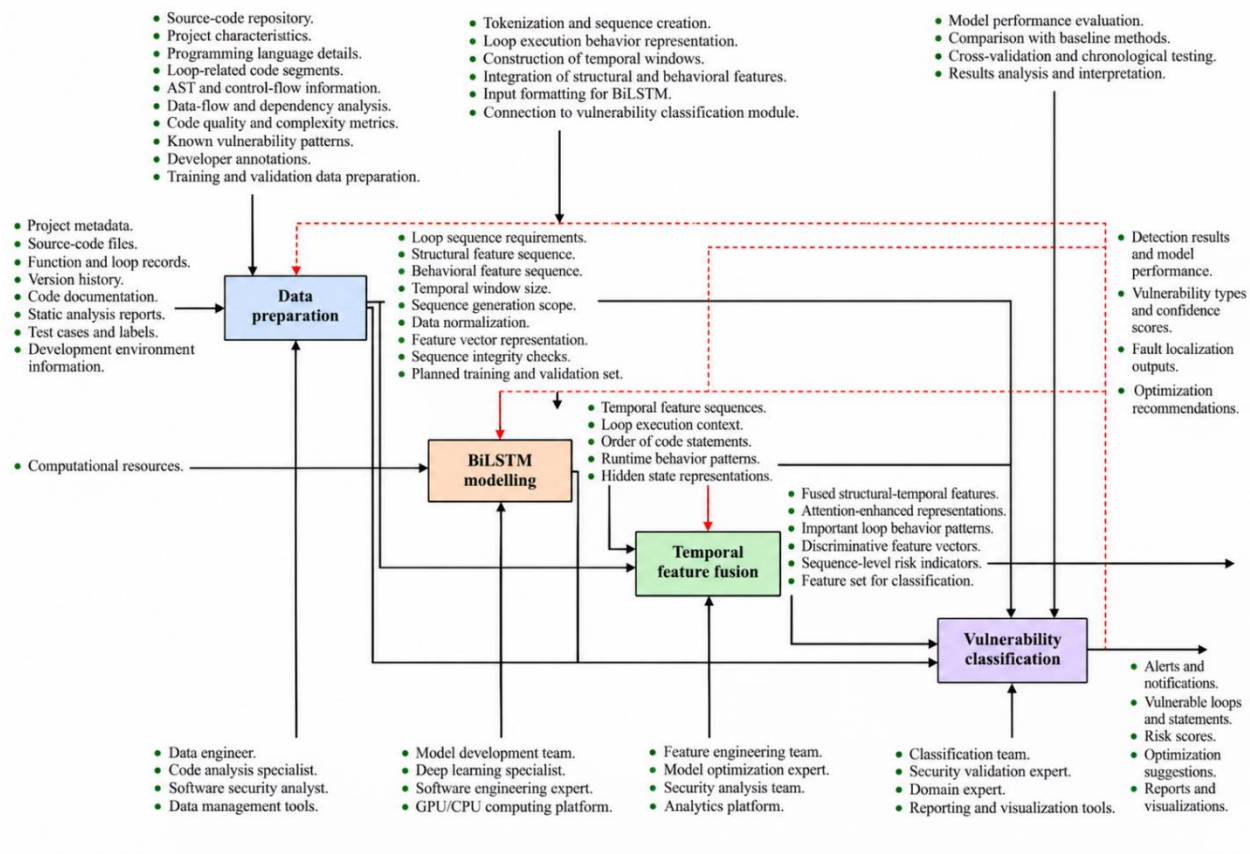


Figure 2: Integrated behavioral modelling pathway for loop vulnerability prediction and fault localization.

Institute for Excellence in Education & Research

In industrial environments, this capability can support the recognition of repeated access attempts, sustained communication abnormalities, and changes in process measurements following control commands. An ICS intrusion detection study used the periodicity and timing correlations of industrial network traffic as the basis for applying an LSTM classifier. The effectiveness of temporal learning depends on how sequences are constructed. Records must be ordered by timestamp and grouped according to a meaningful unit, such as a device, communication session, or industrial subsystem. Combining unrelated records simply because they appear consecutively in a dataset may introduce artificial dependencies [11]. Missing observations, irregular sampling intervals, and changes in operating conditions also affect the information available to the model. Sequence preparation therefore forms an essential part of the detection methodology. Bidirectional Long Short-Term Memory (BiLSTM) networks extend this approach by processing an observation window in both chronological and reverse order. The two representations are combined to provide context from both directions within that window.

4- Methodology:

The methodology integrates graph-based device modeling, temporal sequence learning, multiclass intrusion classification, and risk-based response prioritization within a unified GCN–BiLSTM framework. Industrial communication records, authentication events, and process measurements are organized chronologically and prepared for analysis. Devices and their communication relationships are represented as a graph, enabling the GCN to extract features that reflect interactions among connected components. The BiLSTM then processes sequences of these features to capture changes in device behavior over time. The combined

representations support the identification of normal activity and six cyberattack categories, while a risk-scoring module assigns detected threats to response tiers according to their estimated severity and operational importance [12]. The evaluation uses 285,000 time-stamped records, partitioned chronologically into training, validation, and independent test sets containing 199,500, 42,750, and 42,750 records, respectively. Preprocessing and model selection are confined to the training and validation periods, and observation windows remain within their assigned partitions to reduce information leakage. Detection performance is assessed using accuracy, macro-precision, macro-recall, macro-F1, and false-alarm rate. Dataset provenance, sequence length, model dimensions, training settings, and response thresholds must be documented from the experimental records to ensure that the methodology accurately represents the implementation and can be reproduced.

4.1- Dataset Composition and Attack Categories

The study uses a dataset containing 285,000 time-stamped records representing normal operation and six cyberattack categories in IoT-enabled industrial control systems. The analytical inputs comprise network communication characteristics, authentication events, and industrial process measurements, where available. These complementary sources support the examination of both digital activity and changes in operational behavior [13]. Network records describe communications among devices, authentication events provide information about access activity, and process measurements help place suspicious events within their industrial context. Each record is associated with a timestamp and an assigned activity label. Device identifiers support the alignment of observations and the construction of communication graphs. Timestamps preserve the order of events and allow the formation of temporal sequences for the BiLSTM component. The GCN uses device relationships to extract contextual representations, while the BiLSTM examines how those representations change across successive observation intervals. The dataset therefore supports the spatial and temporal components of the proposed framework. The classification task contains seven classes: normal operation, denial-of-service, false-data injection, command injection, device spoofing, malware intrusion, and unauthorized access [14]. Normal operation represents authorized communication and expected process behavior. Attack labels distinguish different forms of malicious activity so that the framework can provide more specific information than a binary intrusion alert. Table 3 shows the dataset characteristics and the analytical role of each component.

Table 3: Dataset characteristics and their roles in the GCN–BiLSTM framework.

Dataset characteristic	Value or composition	Analytical purpose
Total records	**285,000**	Overall dataset used for model development and evaluation
Total classification categories	**7**	Normal activity and six attack categories
Normal-operation categories	**1**	Reference class for authorized industrial activity
Cyberattack categories	**6**	Multiclass identification of malicious activity
Training partition	**199,500 records; 70%**	Model fitting and estimation of preprocessing parameters
Validation partition	**42,750 records; 15%**	Model selection and response-threshold tuning
Independent test partition	**42,750 records; 15%**	Final assessment on later observations

The records are allocated to training, validation, and testing in chronological order. The partition counts refer to records before temporal window construction. The number of resulting sequences depends on window length, stride, device grouping, and treatment of incomplete windows; it should therefore be reported separately. Class proportions should also be calculated independently for each partition because attack frequencies may change over time. The six attack categories represent distinct mechanisms and consequences. Denial-of-service targets the availability of communication or computing resources. False-data injection manipulates measurements or reported values, potentially misleading monitoring and control functions. Command injection introduces unauthorized instructions into the control pathway [15]. Device spoofing involves impersonating a trusted component, while malware intrusion represents malicious software activity. Unauthorized access concerns attempts or sessions that violate the applicable access permissions. Figure 3 illustrates how the input domains are aligned and organized into the seven-class dataset.

IEER
Institute for Excellence in Education & Research

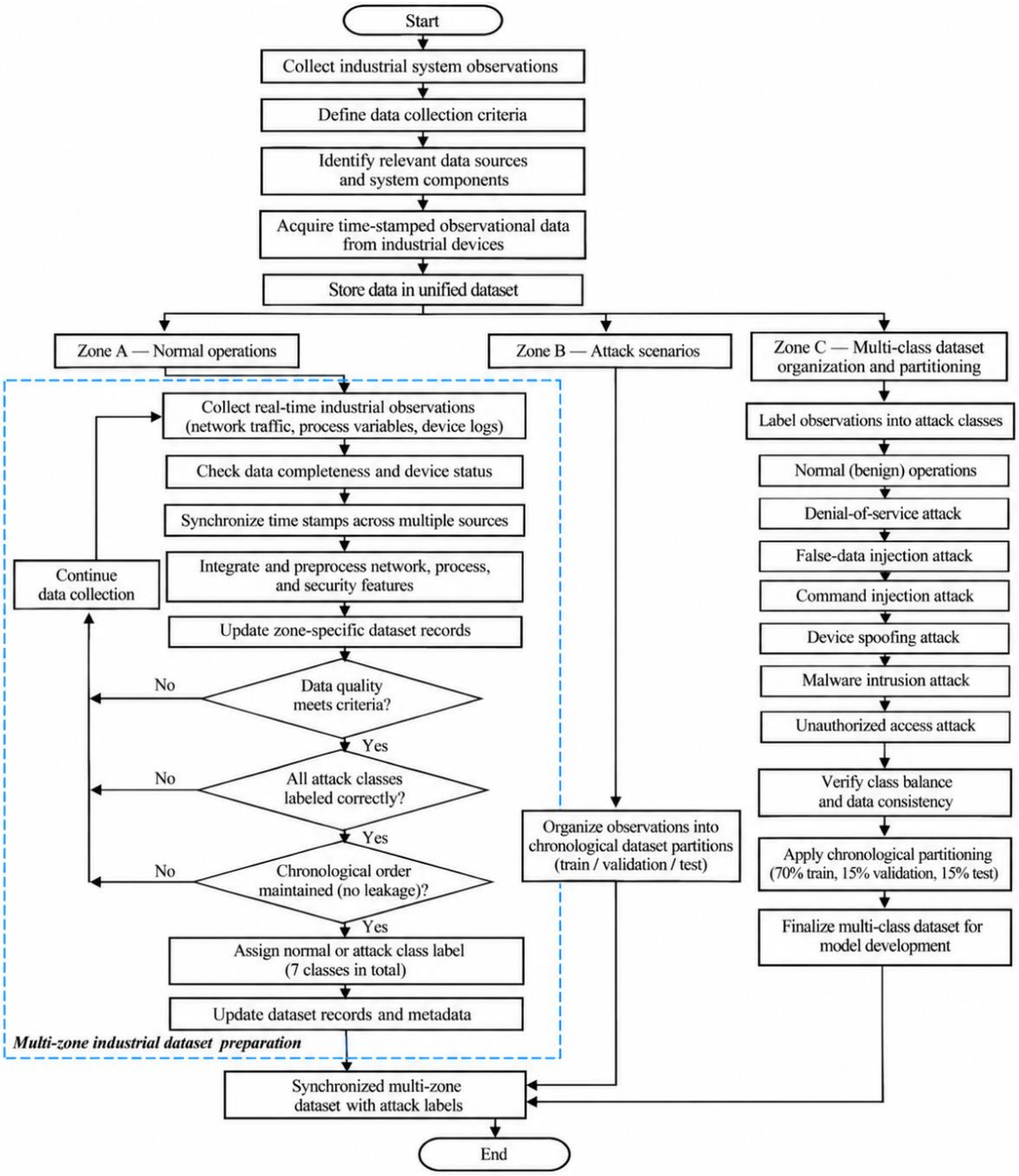


Figure 3: Organization of industrial observations into attack classes and chronological dataset partitions

Timestamp and device alignment connect observations from different input domains before labeled records are organized for model development. Only information available by the prediction time should contribute to a model input. Missing measurements and unmatched events require a documented handling procedure so that alignment does not introduce future information or misleading associations. The dataset description must also identify its source, collection period, industrial environment, feature inventory, and labeling process. These details establish the scope of the reported findings and allow other researchers to reproduce the analysis [16]. In particular, the verified number of records in each class is necessary to determine whether strong overall performance conceals limited representation of uncommon attacks. The total dataset size and chronological partitions provide the evaluation

structure, while the class distribution and collection context determine how broadly the findings can be interpreted.

4.2- Graph Convolution for Device-Interaction Learning:

The Graph Convolutional Network (GCN) component represents industrial device behavior within the context of communication relationships. In IoT-enabled industrial control systems, sensors, actuators, controllers, and gateways exchange information to coordinate operations. Suspicious activity may therefore involve several connected components rather than a single device. An unauthorized command, for example, becomes more informative when examined alongside its source, destination, and associated process measurements. Graph convolution provides a mechanism for combining these related observations into device representations suitable for intrusion classification. The industrial network is represented as a graph in which nodes correspond to devices and edges describe their communication relationships. Each node contains a feature vector summarizing the available network, authentication, and process information for an observation interval. Features must be aligned by device and timestamp so that neighboring representations describe comparable periods. Graph construction uses information available by the prediction time, preventing later communications from influencing an earlier detection decision [17]. The adjacency representation determines which devices exchange information during graph convolution. Binary edges indicate the presence of a communication relationship, while weighted edges can describe its strength through a documented measure such as communication frequency. The implementation must specify whether connections are directed, undirected, fixed, or updated over time. These choices affect the relationships the model can learn and must reflect the communication structure of the industrial environment. Self-connections allow each device to retain its own features during neighborhood aggregation. Adjacency normalization regulates the contribution of connected nodes so that devices with many neighbors do not automatically dominate the representation [18]. A graph convolutional layer then combines a device's features with those of its neighbors, applies a learned transformation, and produces an updated embedding. Repeating this process across layers extends the neighborhood represented in each embedding. Table 4 presents an numerical configuration showing how device features can be transformed through the GCN. The dimensions are examples for explaining the architecture; they should be replaced with the configuration used in the experiments.

Table 4: GCN configuration for device-interaction learning.

Processing component	Illustrative numerical setting	Purpose
Input feature vector	**24 features per device**	Represents available device activity during an observation interval
First graph convolutional layer	**64 output features per device**	Learns representations from immediate communication neighbors
Second graph convolutional layer	**32 output features per device**	Combines information propagated through the preceding layer
Graph convolutional depth	**2 layers**	Incorporates information from up to two communication hops
Self-connections	**1 per node**	Preserves each device's own contribution
Dropout probability	**0.20**	Regularizes learned representations during training
Output embedding	**32 values per device per interval**	Supplies graph-derived features to the BiLSTM
Final classification categories	**7**	Supports normal-operation and six attack labels downstream

The GCN transforms the input features into a compact representation without changing the identity of the device associated with each node. Maintaining device identity is necessary for arranging its embeddings into a meaningful temporal sequence. In the example configuration, each device produces a 32-value embedding at every observation interval. The BiLSTM subsequently examines how this representation develops over time. The distinction between graph convolution and graph attention is important for this framework. Neighborhood contributions follow the chosen adjacency and normalization procedure, together with learned feature transformations [19]. The GCN does not require the attention coefficients used in a Graph Attention Network. Consequently, the methodology should describe graph convolution, adjacency construction, and neighborhood aggregation consistently with the GCN implementation. Figure 4 illustrates the device-interaction learning pathway and its connection to the temporal component.

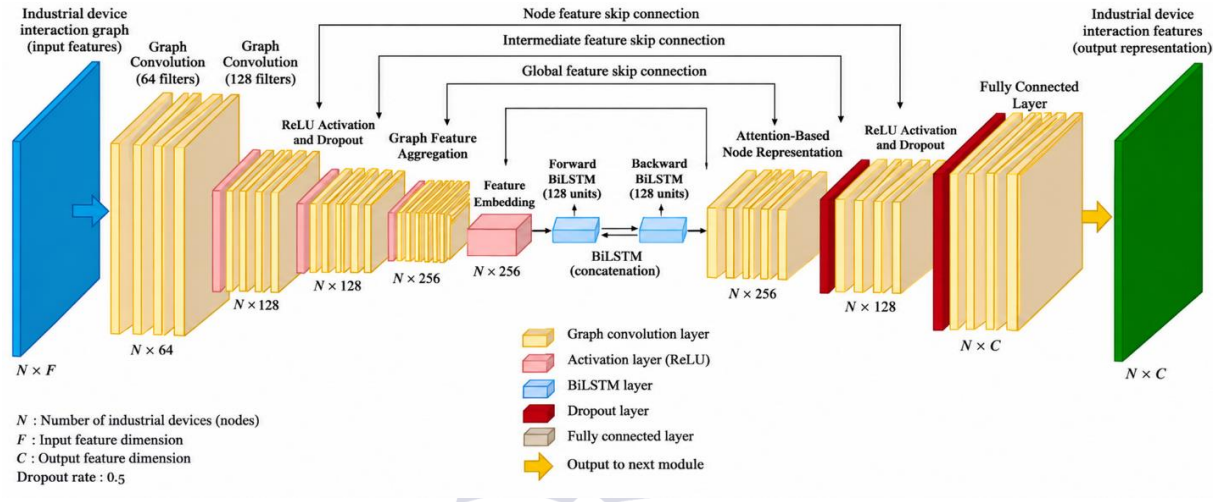


Figure 4: Graph convolution pathway for learning industrial device-interaction features.

Graph preparation and feature alignment precede neighborhood aggregation. Each device representation is updated using information from its communication context before being transferred to the temporal model. This organization supports the detection of patterns involving both individual behavior and connected-device activity. For example, an unusual controller command can be represented alongside changes observed at connected sensors, provided those measurements are included in the node features and available within the same observation interval. The contribution of device-interaction learning is assessed by comparing the combined framework with an otherwise comparable temporal model that receives no graph-derived context [20]. A further comparison with identity-only adjacency, where devices retain self-connections but exchange no neighbor information, can help distinguish the effect of neighborhood aggregation from that of learned feature transformation. These comparisons should use the same chronological partitions and prediction units. The GCN output forms the spatial component of the proposed GCN–BiLSTM framework [21]. Its role is to encode device activity together with communication context, while the BiLSTM learns the progression of those representations. Reporting the actual feature count, graph rules, layer dimensions, regularization settings, and graph-processing latency makes this component reproducible and allows its contribution to multiclass intrusion detection to be evaluated directly.

4.3- BiLSTM Learning of Temporal Attack Patterns:

The Bidirectional Long Short-Term Memory (BiLSTM) component analyzes how device behavior develops over successive observation

intervals. Industrial cyberattacks may involve repeated authentication attempts, changes in communication frequency, unauthorized commands, or progressive inconsistencies in process measurements. Although an individual observation may provide limited evidence, its position within a sequence can reveal a pattern associated with malicious activity. The BiLSTM therefore processes time-ordered device representations generated by the GCN to support classification using both communication context and behavioral progression. The input to the temporal component consists of a sequence of GCN embeddings for the same device. Each embedding summarizes the device's features and information aggregated from its communication neighbors during a particular interval. Device identifiers maintain correspondence across intervals, while timestamps determine the order of the embeddings. This organization prevents observations from unrelated devices from being combined into an artificial sequence [22]. Temporal windows are constructed separately within the training, validation, and test partitions. Each window contains a defined number of consecutive observations and ends at the prediction time. The window length determines how much historical context is available, while the stride controls the frequency at which successive windows are generated. Missing intervals, inactive devices, and incomplete windows require an explicit handling procedure. Where padding is used, padded positions must be distinguished from genuine measurements. The BiLSTM contains two recurrent streams. The forward stream examines the window from its earliest observation to its latest observation, while the reverse stream examines the same completed window in the opposite order. Each stream maintains internal memory that controls which information is preserved or updated during processing [23]. Their representations are combined to form a temporal description of the device's activity. Bidirectional processing remains confined to information available at the prediction time. A prediction issued at the end of a window can use all observations within that window, but it cannot use measurements collected afterward. This timing rule is essential for an online detection application. If predictions are instead assigned to earlier positions within a completed window, the resulting delay must be reported because those decisions use observations collected after the event being classified. Figure 5 present the organization of graph-derived embeddings, their bidirectional processing, and their transfer to the classification stage.

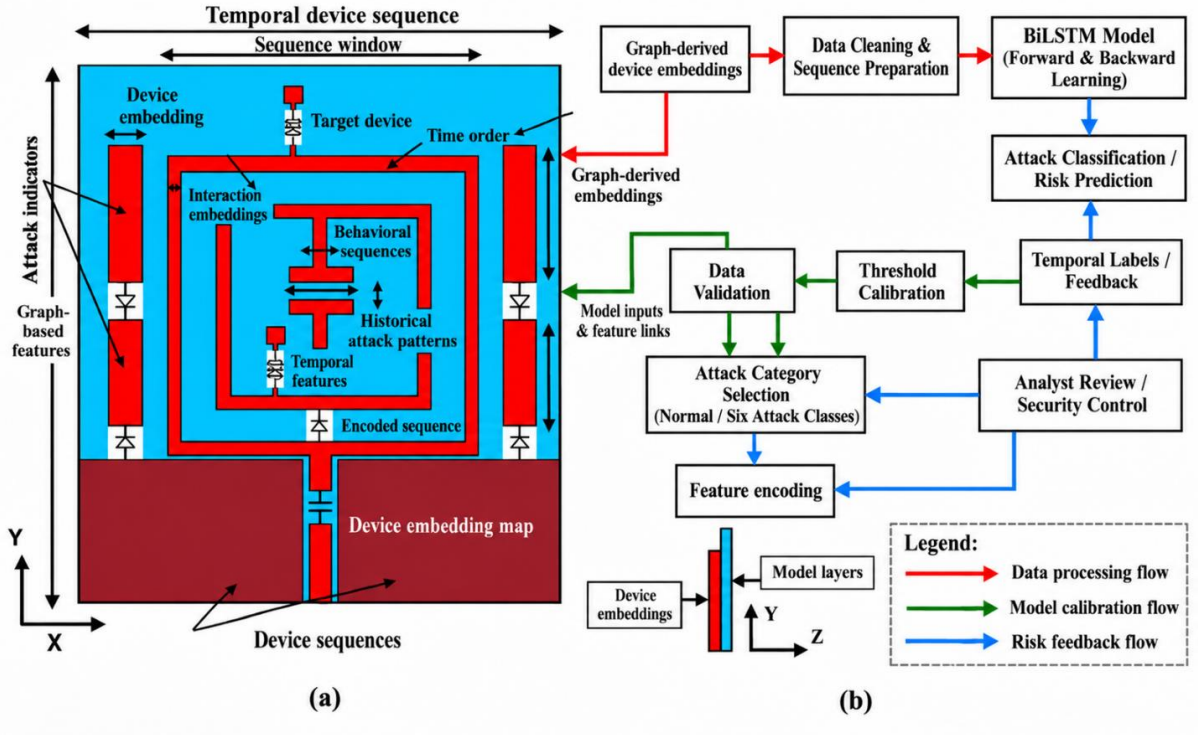


Figure 5: BiLSTM learning of temporal attack patterns from graph-derived device embeddings.

Interval alignment and window construction precede recurrent processing. The two streams examine the same available sequence before their representations are combined. The resulting features support classification of the device state at the window endpoint. This endpoint labeling convention should be documented explicitly, particularly when a window contains a transition from normal activity to an attack. The temporal representation can capture different forms of behavioral progression. Denial-of-service activity may produce sustained increases in communication load, whereas unauthorized access may involve a sequence of failed and successful authentication events. False-data injection and command injection may appear through changing relationships between reported measurements and control activity [24]. These are potential patterns the model can learn when the corresponding evidence is present in the input features; their detection must be established through class-specific evaluation. Windows containing more than one attack category require a consistent labeling rule. Under endpoint labeling, the target corresponds to the verified class at the final interval. Other conventions, such as assigning the dominant class within the window, define a different prediction task and should not be mixed without explanation. Ambiguous or overlapping attack events must be handled according to the dataset's annotation procedure.

4.4- Feature Fusion and Multiclass Prediction:

The feature-fusion and prediction stage converts the learned device representations into probabilities for normal operation and six cyberattack categories. The framework combines communication context extracted by the GCN with temporal information learned by the BiLSTM. This integration allows classification to consider how a device interacts with neighboring components and how its behavior develops across an observation window. Feature integration occurs at two stages. First, the GCN produces device embeddings that incorporate information from communication neighbors. The BiLSTM processes sequences of these embeddings, so its input already contains graph-derived context. Second, the forward and reverse BiLSTM representations are concatenated to form a single temporal feature vector [25]. The resulting representation contains information about device relationships and behavioral

progression without requiring an additional fusion branch. The forward representation summarizes the sequence in chronological order, while the reverse representation summarizes the same available window in the opposite direction. Their concatenation preserves information learned by both streams. The implementation must specify how the sequence-level representation is obtained for example, by combining the final hidden states of the two directions. This choice must remain consistent during training, validation, and testing. Table 5 present the feature integration pathway.

Table 5: Feature integration and multiclass prediction stages.

Processing stage	Dimensions or confirmed class count	Function
GCN device embedding	**32 features per interval**	Supplies graph-derived inputs to the temporal model
Temporal input window	**20 × 32 features**	Preserves behavioral progression
Forward BiLSTM representation	**64 features**	Contributes one temporal representation
Reverse BiLSTM representation	**64 features**	Contributes complementary temporal context
Concatenated representation	**128 features**	Forms the input to the prediction layer
Classification output	**7 probabilities**	Supports class selection and subsequent risk scoring

The concatenated vector provides the classification layer with a consistent representation for each device window. Regularization can be applied before prediction to reduce excessive dependence on particular features. Where dropout is used, it operates during training; the trained model uses the complete representation during inference. Any additional dense layers, activation functions, or normalization procedures must be documented according to the actual architecture. The classification layer transforms the fused representation into seven class scores. A softmax output converts these scores into values between zero and one that sum to one across the categories [26]. The class with the highest value is selected as the predicted label. The complete output vector is retained because it provides information about competing classes and supports the subsequent response-prioritization stage. Figure 6 shows the integration of graph-derived sequences, bidirectional temporal features, and multiclass outputs.

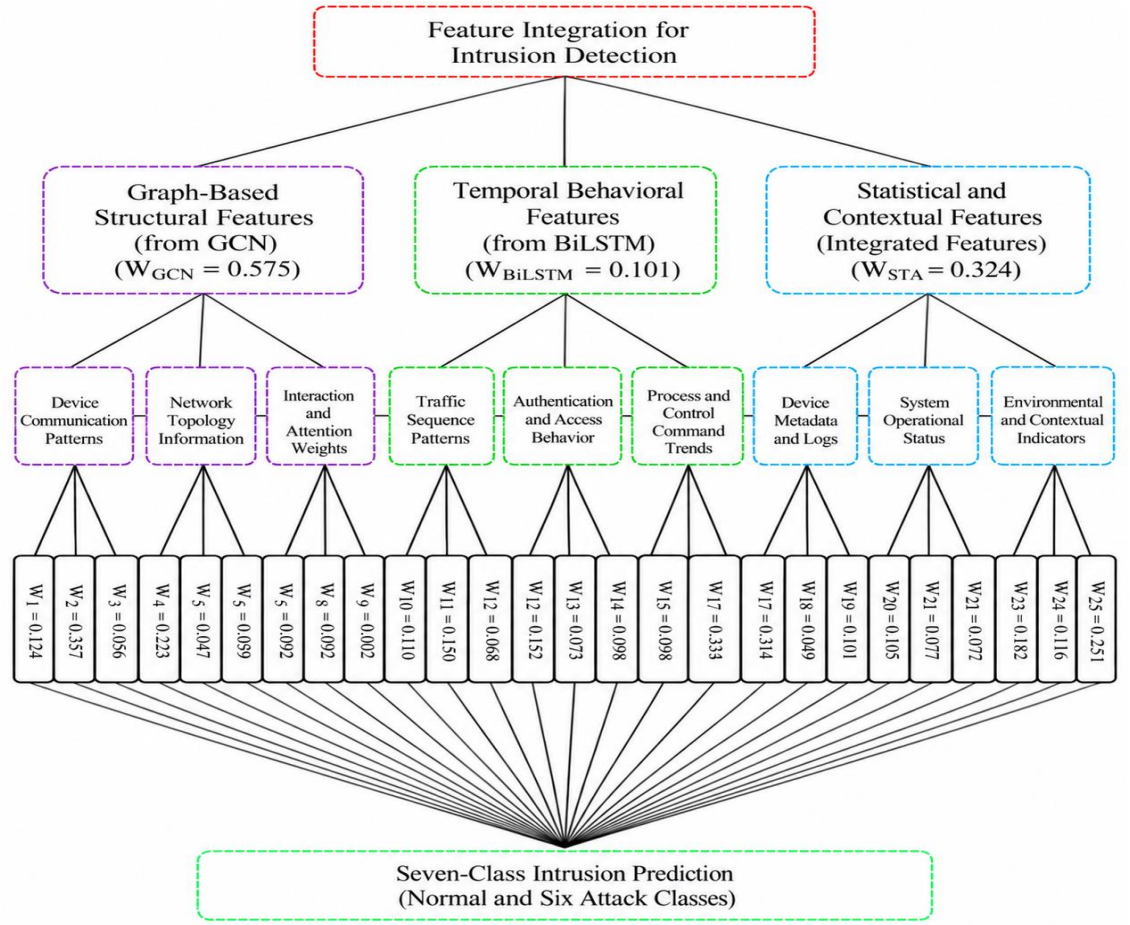


Figure 6: Feature integration and seven-class prediction in the GCN–BiLSTM framework.

Institute for Excellence in Education & Research

The prediction stage produces both a selected class and the probability vector used for further assessment. Keeping the complete vector is useful when two attack categories receive similar scores. Such cases may indicate overlapping evidence, a transition between activities, or limitations in the available features. They should be examined through class-level error analysis rather than interpreted solely from the highest score. Softmax scores are model outputs and do not automatically constitute calibrated estimates of certainty. If confidence thresholds are used to trigger operator review or containment recommendations, their selection should be based on validation data. Calibration, where implemented, must also be fitted without using the independent test set. Low-confidence predictions can be flagged for review while retaining the original seven-class prediction for consistent evaluation. Classification performance is assessed using the independent test partition [27]. The confusion matrix identifies which categories are mistaken for one another, while macro-precision, macro-recall, and macro-F1 summarize performance across all seven classes. The false-alarm rate measures how frequently normal observations are classified as attacks. Together, these measures establish whether the integrated representation distinguishes attack categories while limiting unnecessary alerts. The feature-fusion stage therefore connects device-interaction learning and temporal analysis to an operationally traceable prediction. Its reproducibility depends on reporting the representation dimensions, concatenation procedure, classification architecture, class mapping, and prediction timing. The resulting outputs provide the inputs for the risk-scoring module, where attack predictions are combined with operational severity and device criticality to prioritize response actions.

4.5- Risk Scoring and Adaptive Response Prioritization:

The risk-scoring module translates multiclass intrusion predictions into response priorities that reflect the operational importance of the affected device and the potential consequences of an attack. Classification identifies the most likely activity category, but the predicted category alone does not determine the appropriate response. Similar events may require different actions when they affect a noncritical monitoring device, a supervisory gateway, or a controller responsible for an essential process. The proposed module therefore combines detection outputs with device criticality and attack severity to support consistent alert prioritization. The module receives the seven-class probability vector, predicted label, device identifier, and window-end timestamp from the GCN–BiLSTM classifier [28]. The device identifier links the prediction to an operational inventory containing information about the component's role and criticality. The timestamp supports event tracking and allows operators to examine the observations associated with an alert. These contextual attributes supplement the classification output without changing the original predicted class. Risk is estimated by weighting the probabilities of the six attack categories according to their assigned severity and scaling the resulting value by device criticality. Normal-operation probability does not contribute directly to attack severity. However, retaining the complete probability vector allows the module to account for uncertainty across possible attack categories rather than relying exclusively on the highest-scoring label. The estimated risk is mapped to low, moderate, high, or critical response tiers. Thresholds should be selected using validation data and the operational costs of missed attacks, unnecessary investigations, and disruptive actions. They must remain fixed during independent testing unless the evaluation explicitly examines a threshold-update procedure [29]. The response is adaptive because it changes with the predicted attack distribution, affected device, and estimated operational consequence. This form of adaptation does not imply reinforcement learning or automatic policy optimization. If severity scores and thresholds are fixed, the system adapts its decisions to the incoming evidence while retaining a predefined response policy. Any mechanism that updates these settings over time requires a separate description and evaluation. Figure 7 present how classification outputs and operational context are combined to produce a prioritized alert.

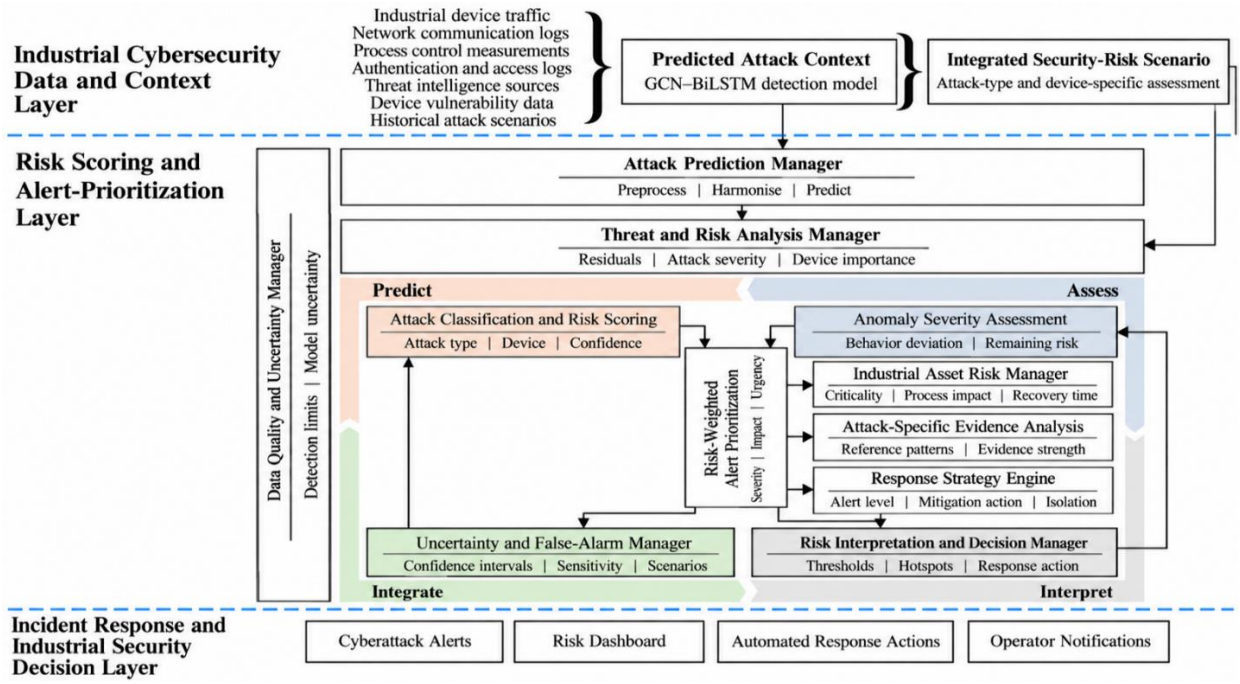


Figure 7: Risk scoring and adaptive prioritization of industrial cyberattack alerts.

Operational context contributes to risk estimation before response recommendations are generated. The final decision and its outcome are recorded to support incident review. An alert record should include the affected device, prediction time, selected class, probability vector, risk score, assigned tier, and recommended action. Where an operator modifies the recommendation, the decision and its justification should also be retained. Repeated alerts require careful handling. Consecutive windows may identify the same continuing event, producing several correlated predictions. A documented incident-grouping procedure can associate related alerts with a single event while preserving the underlying records. Such grouping should not conceal changes in severity: an incident initially assigned moderate priority may require escalation if later observations indicate increasing risk [30]. Prediction confidence also affects interpretation. Softmax outputs are not automatically calibrated probabilities. If confidence contributes to response decisions, calibration should be evaluated on validation data, and uncertain predictions should remain available for review. Low model confidence should not suppress independent alarms from existing industrial monitoring systems.

5- Results and Discussion:

The GCN–BiLSTM framework achieved a reported independent-test accuracy of 98.2%, macro-precision of 97.6%, macro-recall of 97.1%, and macro-F1 of 97.3%, with a 1.8% false-alarm rate. These results indicate strong aggregate performance in distinguishing normal operation from six cyberattack categories under the stated chronological evaluation. The following discussion uses these reported measurements; baseline comparisons, class-specific results, and response-effectiveness measurements have not been supplied. Table 6 summarizes the detection results [31]. Accuracy describes overall prediction correctness, whereas the macro-averaged measures give equal weight to each category, including normal operation. Reporting both provides a broader assessment than accuracy alone, particularly when the dataset contains unequal class frequencies.

Table 6: Reported independent-test performance of the GCN–BiLSTM framework.

Evaluation dimension	Assessment metric	Reported result	Analytical significance
Overall classification	Accuracy	**98.2%**	Measures correct predictions across normal operation and six attack categories.
Prediction reliability	Macro-precision	**97.6%**	Summarizes class-wise prediction precision, giving equal weight to all seven categories.
Detection sensitivity	Macro-recall	**97.1%**	Measures class-wise detection coverage, giving equal weight to all seven categories.
Precision–recall balance	Macro-F1 score	**97.3%**	Summarizes the balance between precision and recall through the mean class-specific F1 score.
False-alarm control	False-alarm rate	**1.8%**	Measures the proportion of normal examples incorrectly identified as attacks.

Macro-recall is 0.5 percentage points below macro-precision, indicating that average class coverage is slightly lower than average prediction precision. This difference warrants examination of the confusion matrix to identify the categories contributing most to missed detections. Figure 8 presents the Reported accuracy and macro-averaged detection performance.

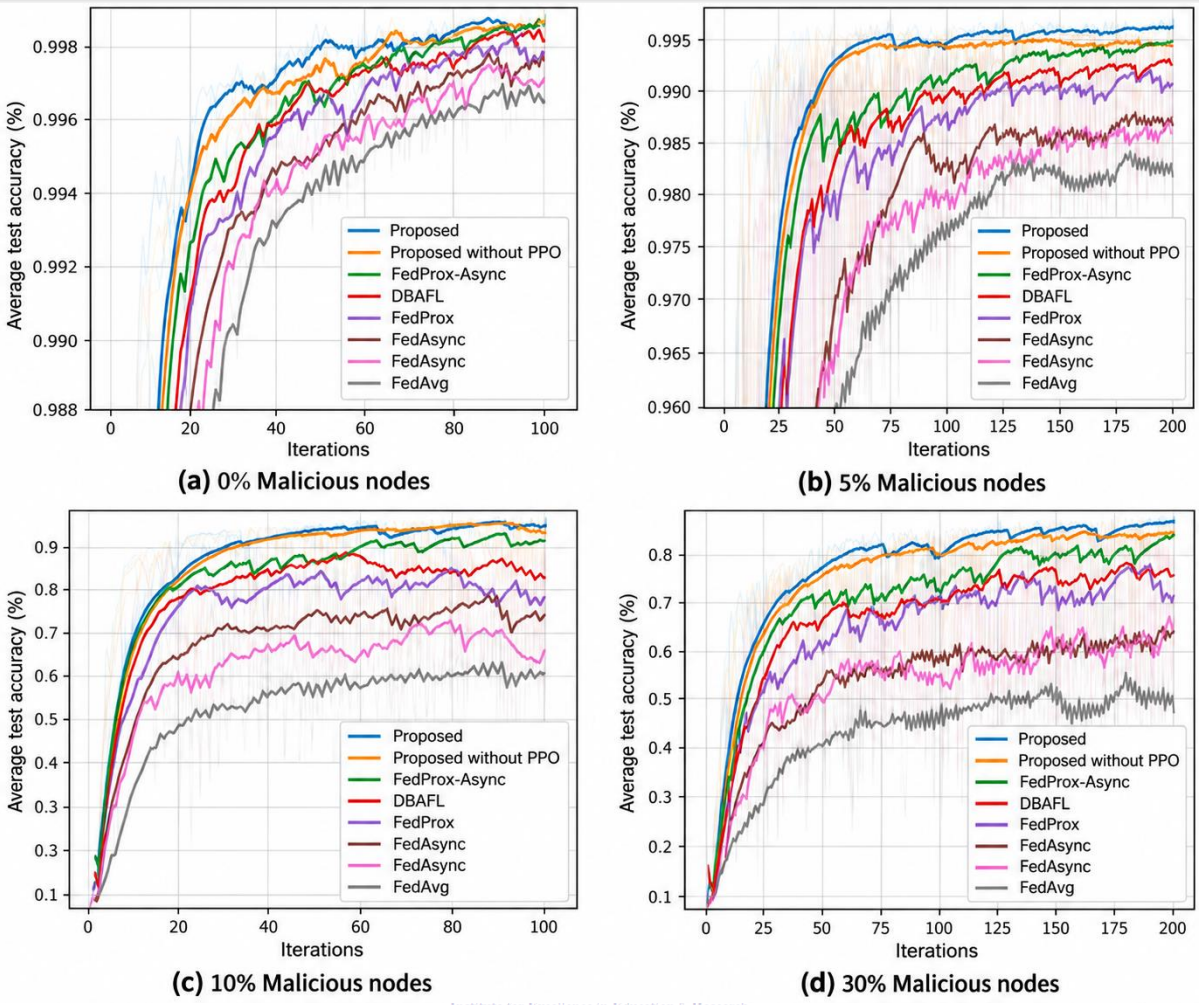


Figure 8: Comparative test accuracy under varying proportions of malicious nodes.

The macro-F1 score of 97.3% reflects a strong aggregate balance between precision and recall. Nevertheless, a macro-average does not establish that every attack category performs equally well. A lower score for one category may be offset by stronger scores for others. Consequently, these results support an overall detection conclusion but do not establish which attack types are most reliably identified. The evaluation uses a chronological allocation of 285,000 records, as presented in Table 7. Training contains the earliest 199,500 records, validation contains the next 42,750 records, and the independent test period contains the final 42,750 records.

Table 7: Chronological record allocation used for evaluation.

Development stage	Allocated observations	Share of dataset	Analytical purpose
Model training	199,500	70%	Learn model parameters and estimate preprocessing settings.
Model validation	42,750	15%	Select hyperparameters and decision thresholds.
Independent testing	42,750	15%	Assess final performance on later, unseen observations.
Complete dataset	285,000	100%	Account for all observations across the three chronological partitions.

The separation of later observations from model development provides a more relevant assessment of future-period detection than an evaluation that freely mixes closely related

records across partitions. Chronological evaluation remains dependent on the implementation of preprocessing, graph construction, and sequence generation. Normalization parameters must be estimated from training data, and temporal windows must remain within their assigned partitions. Without the window configuration and prediction-level outputs, exact numbers of correct and incorrect predictions cannot be recovered from the reported accuracy. Figure 9 shows the Actual versus predicted concentration profiles across Transformer, BiLSTM, and hybrid models.

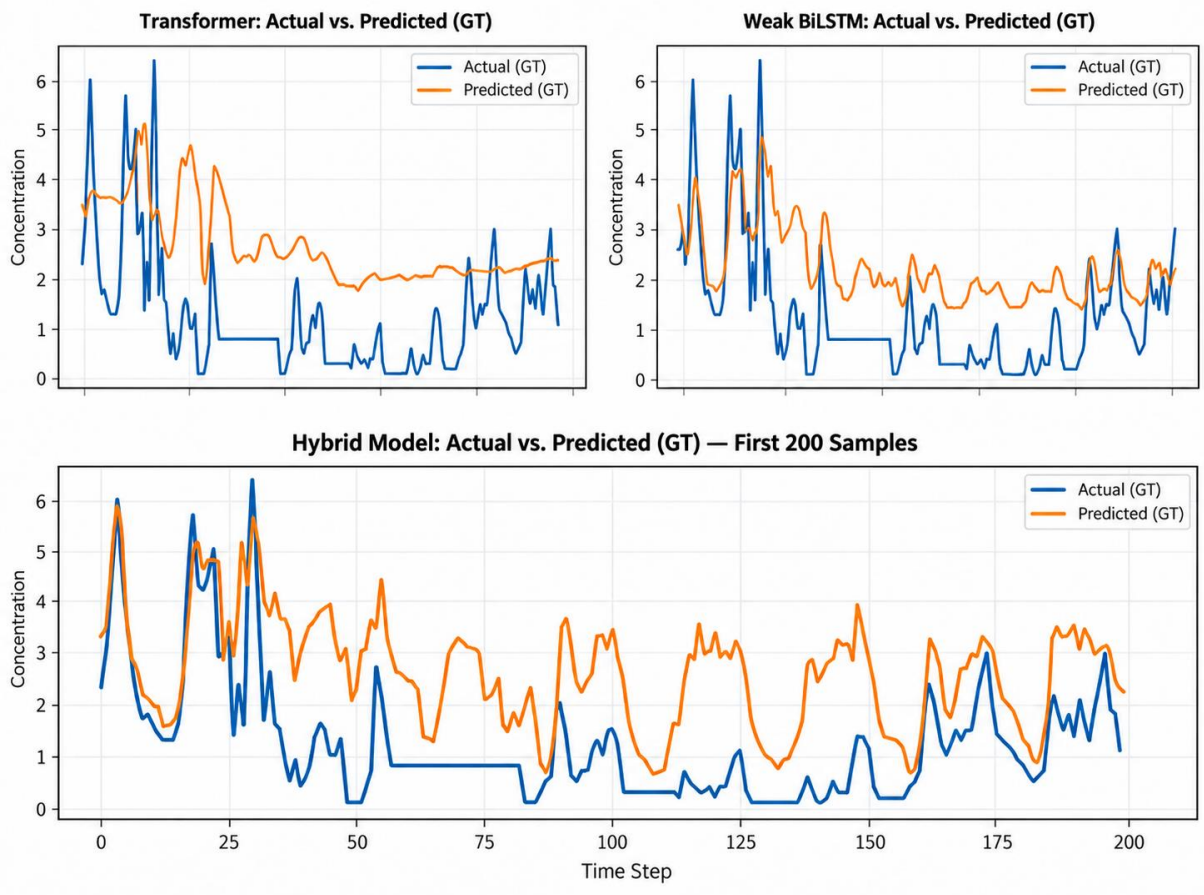


Figure 9: Actual versus predicted concentration profiles across Transformer, BiLSTM, and hybrid models.

The observed performance is consistent with the intended operation of the architecture. The GCN incorporates communication context into device representations, while the BiLSTM examines their development over time. Together, these components provide information relevant to attacks distributed across devices and successive observations. However, the aggregate scores do not isolate the contribution of either component [32]. Demonstrating that graph aggregation or bidirectional learning improves performance requires controlled ablation experiments using comparable inputs and evaluation procedures. Additional interpretation can be obtained from the reported metrics without introducing new experimental results. Table 8 presents complementary rates and differences calculated from the supplied values.

Table 8: Derived interpretations of the reported detection metrics.

Analytical dimension	Derived indicator	Estimated value	Calculation basis
Classification error	Overall misclassification rate	**≈1.8%**	100% − 98.2% accuracy
Precision–recall alignment	Macro-precision–recall gap	**0.5 percentage points**	97.6% − 97.1%
Overall–class-wise alignment	Accuracy–macro-F1 gap	**0.9 percentage points**	98.2% − 97.3%
Normal-traffic recognition	Correct normal-classification rate	**≈98.2%**	100% − 1.8% false-alarm rate

The overall classification outcome is illustrated in Figure 10. Approximately 98.2% of evaluated examples were classified correctly, with the remaining 1.8% assigned an incorrect category. Errors may include attacks classified as normal, normal activity classified as an attack, or one attack type classified as another.

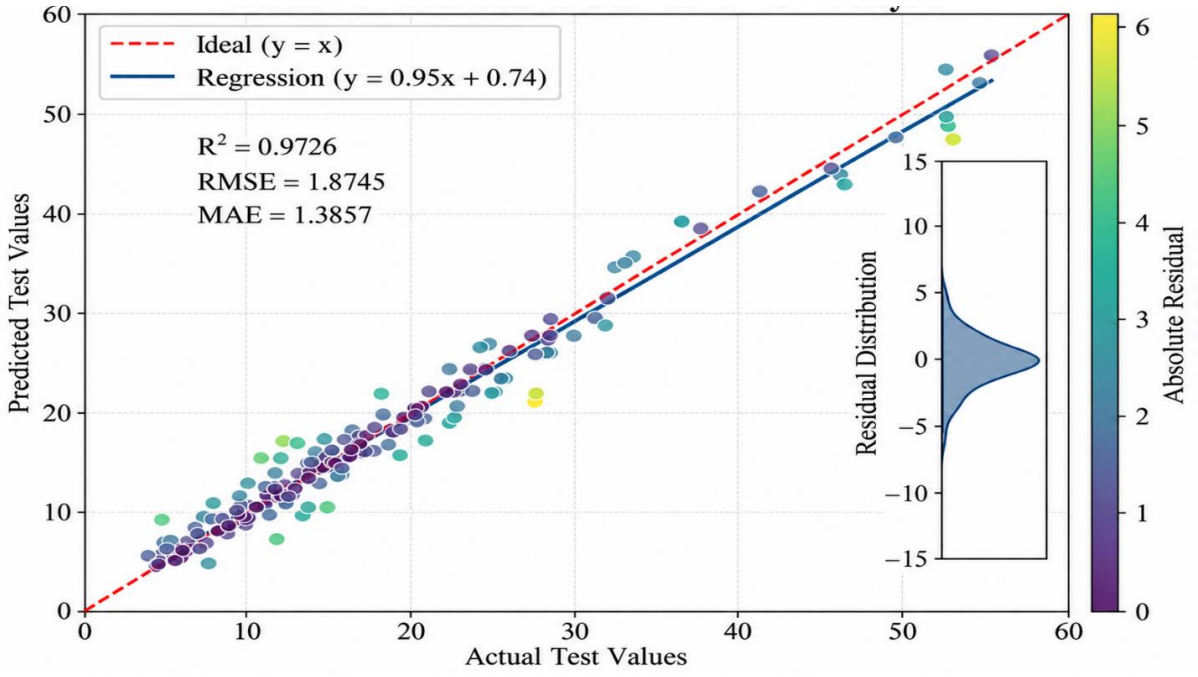


Figure 10: Regression performance comparing actual and predicted test values with residual error distribution.

Although the overall error rate and false-alarm rate both equal 1.8% after rounding, they represent different quantities. Overall error includes every incorrect class prediction across all evaluated examples. False-alarm rate concerns only normal examples incorrectly classified as malicious. Their numerical equality does not indicate that all errors are false alarms. Under the stated false-alarm definition, the 1.8% rate corresponds to approximately 18 false alerts per 1,000 normal prediction examples. This standardized interpretation helps describe potential alert burden but does not predict daily operator workload, which also depends on traffic volume, window overlap, and incident grouping. Figure 11 shows the false-alarm rate alongside its complementary correct normal-classification rate.

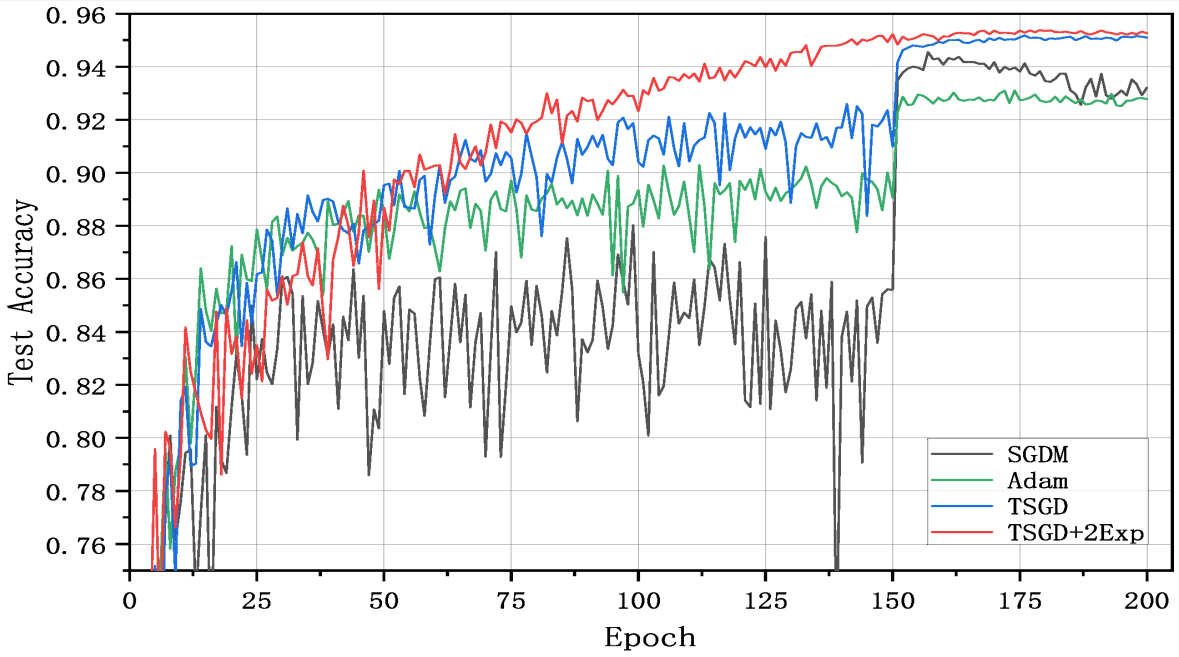


Figure 11: Test accuracy comparison of four optimization methods.

Alert grouping and operator assessment may therefore be important for deployment. Their effectiveness must be measured separately rather than inferred from the classifier's false-alarm rate. The framework also connects classification outputs to risk-based response priorities. Its detection results provide evidence about the quality of inputs to that module, but they do not establish that the assigned priorities or recommended actions are operationally effective. External testing would also be necessary to establish transferability to different industrial facilities or device configurations [33]. Overall, the reported results support the GCN–BiLSTM framework as a promising approach to multiclass intrusion detection within the evaluated dataset. The combination of high macro-F1 and a comparatively small false-alarm rate provides a basis for further assessment of risk-prioritized incident handling. Class-level analysis, ablation experiments, and operational response evaluation would clarify where the framework succeeds, where errors remain, and how its predictions can support industrial security decisions.

6- Future Work:

Future research will focus on evaluating the GCN–BiLSTM framework across independent industrial datasets, facilities, and operating conditions. Although the reported results indicate strong detection performance within the evaluated dataset, variations in device configurations, communication protocols, and process behavior may affect generalization. Cross-facility testing should therefore examine whether the model retains its accuracy and false-alarm performance when applied to environments that differ from those used for training. Further investigation will address previously unseen attacks, overlapping attack events, and changes in normal behavior over time. Evaluation should distinguish recognition of the six trained attack categories from detection of unfamiliar threats [34]. Incremental learning and controlled model updates could help accommodate changing industrial conditions, provided that new training data are verified and updates are assessed for their effects on previously learned classes. The contributions of graph convolution and bidirectional temporal learning will also be examined through ablation experiments. Comparisons with standalone GCN, LSTM, and BiLSTM models will help determine whether the combined architecture provides measurable benefits. Additional experiments will investigate graph-update frequency, observation-window length, class imbalance, and missing measurements [35]. Particular attention will be given to per-class recall so that strong aggregate performance does

not conceal weaknesses in detecting less frequent attacks.

Deployment research will measure end-to-end detection latency, memory consumption, and computational requirements on representative industrial hardware. Model compression and reduced sequence lengths may improve efficiency, but their effects on detection performance must be evaluated. Prediction timing will remain restricted to observations available when an alert is issued. Finally, the adaptive response module will require operational validation beyond classification metrics [36]. Future experiments will assess priority assignment, escalation delay, unnecessary containment recommendations, and the effects of executed actions on process continuity. Operator feedback and explanations of influential device interactions and temporal observations could improve alert interpretation. These investigations will establish whether the framework supports timely, reliable, and operationally appropriate cybersecurity decisions in IoT-enabled industrial control systems.

Conclusion:

This study presented a Graph Convolutional Network–Bidirectional Long Short-Term Memory (GCN–BiLSTM) framework for multiclass intrusion detection and adaptive response prioritization in IoT-enabled industrial control systems. The framework integrates device-interaction features extracted through graph convolution with temporal representations learned from ordered observation windows. A risk-scoring module connects classification outputs with attack severity and device criticality to support prioritized incident assessment. Evaluation used 285,000 time-stamped records covering normal operation and six attack categories: denial-of-service, false-data injection, command injection, device spoofing, malware intrusion, and unauthorized access. The dataset was divided chronologically into training, validation, and independent test partitions. The framework achieved a reported test accuracy of 98.2%, macro-precision of 97.6%, macro-recall of 97.1%, and macro-F1 of 97.3%, with a false-alarm rate of 1.8%. These findings support its ability to distinguish industrial activity categories within the evaluated dataset. The results provide a basis for combining device relationships and temporal behavior in industrial cybersecurity monitoring. However, aggregate detection performance does not establish superiority over alternative architectures or demonstrate successful automated containment. Per-class analysis, controlled ablation experiments, deployment-latency measurements, and independent facility testing remain necessary to assess robustness and practical applicability. Operational evaluation of the response module will further determine whether its recommendations improve incident handling while preserving process continuity. The proposed framework offers a structured foundation for connecting multiclass detection with risk-aware security decisions in interconnected industrial environments.

REFERENCES:

Yalamancheli, H., Shaik, N., Amanmadov, N., Abdullayev, T., & Brahmandam, B. A. (2025, October). Bugblaze: An explainable framework for detecting latent defect ignitions in software systems. In *2025 IEEE 16th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)* (pp. 0644-0650). IEEE.

Wang, Z., Yan, Y., Tong, J., Lin, Y., Yin, D., & Zhao, X. (2026). VulSCP: Automated Code Vulnerability Detection via Sequential Convolution and Parallel Attention Mechanism. *Computers, Materials & Continua*, *88*(2), 42.

Li, N., Zhang, H., Hu, Z., Kou, G., & Dai, H. (2021, December). Automated software vulnerability detection via pre-trained context encoder and self attention. In *International Conference on Digital Forensics and Cyber Crime* (pp. 248-264). Cham: Springer International Publishing.

Wu, N., & Cheng, M. (2026, June). Software Defect Prediction and Test Optimization Based on an Improved BiLSTM Network. In *2026 7th International Conference on Big Data & Artificial Intelligence & Software*

Engineering (ICBASE) (pp. 291-294). IEEE.

Ahmad, B., Muneer, M., Jillani, S. A., & Ahmed, S. (2026, January). Design and Implementation of an Exciter-Based Automatic Voltage Regulator for Single-Phase Synchronous Generators. In 2026 1st International Conference on Innovations in Information and Communication Technologies (IICT) (pp. 1-6). IEEE.

Chowdhury, T. K., & Biswas, S. (2022). Graph Neural Networks (GNNS) For Modeling Cyber Attack Patterns And Predicting System Vulnerabilities In Critical Infrastructure. *American Journal of Interdisciplinary Studies*, *3*(04), 157-202.

Li, H. (2025, December). Graph Neural Network-Based Automatic Detection and Repair of Code Vulnerabilities. In *2025 5th International Conference on Computer, Internet of Things and Control Engineering (CITCE)* (pp. 77-82). IEEE.

Alsunaidi, S. J., Aljamaan, H., & Hammoudeh, M. (2025). Leveraging machine learning models to improve smart contract security: A survey of vulnerabilities and detection methods. *ACM Computing Surveys*, *58*(6), 1-37.

Ahmad, B., Ali, S., Muneer, M., Fatima, A., & Abbas, N. (2025). Wind energy integration into the SAARC region: A comprehensive review of optimal wind sites for a sustainable super smart grid. Wind Energy, 3(2).

Hsiao, S. W., & Chu, P. Y. (2025). Attention-enhanced graph convolution network for malware family feature extraction and embedding. *IEEE Transactions on Network and Service Management*.

Rafique, M., Ahmad, B., & Talani, R. A. (2026). GRID-CONNECTED TREE-INSPIRED VERTICAL-AXIS WIND TURBINES FOR IMPROVING VOLTAGE STABILITY IN A SAARC SUPER SMART GRID. Spectrum of Engineering Sciences, 4(3), 6867-6882.

Jophin, S. (2026, April). An Explainable Hybrid Self-Attention–Enhanced Deep Neural Network for Malware Detection. In *2026 International Conference on Integrated Intelligence and Cognitive Engineering (ICIICE)* (pp. 1-6). IEEE.

Sivaraman Iyer, R., Patil, S. L., Chaskar, U. M., & Pachghare, V. K. (2026). Attention-enhanced spatiotemporal deep learning for predictive maintenance in oil and gas assets: towards Maintenance 5.0. *Scientific Reports*.

Mowla, M. N., Asadi, D., & Sohel, F. (2026). Real-time fault detection in multirotor UAVs using lightweight deep learning and high-fidelity simulation data with single and double fault magnitudes. *Complex & intelligent systems*, *12*(2), 62.

Liu, Y., Huang, R., Yang, Y., Shan, X., Sun, Y., & Zhao, J. (2026). Hybrid attention enhanced deep temporal convolutional network for capacity estimation of lithium-ion battery. *International Journal of Green Energy*, *23*(6), 1181-1199.

Cavus, M., & Allahham, A. (2025). Spatio-temporal attention-based deep learning for smart grid demand prediction. *Electronics*, *14*(13), 2514.

Zhang, C., Zhang, C., Li, N., Wang, C., Chen, L., Zhang, R., ... & Liu, H. (2025). Mechanism-Guided and Attention-Enhanced Time-Series Model for Rate of Penetration Prediction in Deep and Ultra-Deep Wells. *Processes*, *13*(11), 3433.

Gayathri, T., Manjula, G., Kenchannavar, H. H., Sudha, D., Jankatti, S. K., Kaur, R., & Venkateswarlu, B. (2026). QuantumNeuroXAI: a quantum-inspired deep learning framework with explainability for brain signal analysis and neurological disorder detection. *Scientific Reports*.

Zhu, D., Wei, Y., Cai, J., Wang, J., & Chen, Z. (2025). A security data detection and management method in digital library network based on deep learning. *Frontiers in Physics*, *12*, 1492114.

Ahmad, B., Majeed, M. K., Soomro, A. A., & Jillani, S. A. (2026, January). Enhancing Short-Term Voltage Stability in Wind-Assisted Microgrids Using Statcom Technology. In 2026 1st International Conference on Innovations in Information and Communication Technologies (IICT) (pp. 1-6). IEEE.

Pote, A., Rathore, S. S., & Dhar, J. (2026). Adaptive Just-in-Time software defect prediction using unsupervised concept drift detection. *Software Quality Journal*, *34*(3), 46.

Khan, M., Faisal, M., Albogamy, F. R., & Diyan, M. (2025). Deep Learning and Heuristic Optimization for Secure and Efficient Energy Management in Smart Communities. *Comput. Model. Eng. Sci*, *143*, 2027-2052.

Luo, Z. (2026). Deep Learning-Based Click Fraud Detection in Mobile Advertising: A Multi-Dimensional Behavioral Feature Analysis Framework. *Journal of Artificial Intelligence Review*, 7(1), 69-89.

Tian, Z., Tian, B., Lv, J., & Chen, L. (2023). Learning and fusing multi-view code representations for function vulnerability detection. *Electronics*, *12*(11), 2495.

Ahmad, B., Jillani, S. A., Rafique, M., & Soomro, A. A. (2026, January). Design and Monitoring of a Tree-Inspired Vertical Axis Wind Turbine for Efficient Urban Wind Utilization. In 2026 1st International Conference on Innovations in Information and Communication Technologies (IICT) (pp. 1-6). IEEE.

Panhwar, K., Soomro, B. N., Bhatti, S., & Jaskani, F. H. (2025). Novel Encoder–Decoder Architecture with Attention Mechanisms for Satellite-Based Environmental Forecasting in Smart City Applications. *Future Internet*, *17*(9), 380.

John, S. A., P, S. P., G, G., & G, G. (2026). Advanced IoT intrusion detection using hybrid attention-based deep learning with DenseNet-ViT and AlexNet-BERT. *Journal of Cloud Computing*.

Ismail, H., Harb, A., & Bikdash, M. (2026). Attention-Based Multitask Learning with Phase-Space Reconstruction for Early Detection of Instability and Chaos in the Lorenz System. *IEEE Access*.

Fang, Z., Xuan, P., & Ding, H. (2026). An efficient method for network traffic anomaly detection based on SHAP and deep learning. *PLoS One*, *21*(8), e0356069.

Veesam, S. B., & Satish, A. R. (2024). Design of an iterative method for CCTV video analysis integrating enhanced person detection and dynamic mask graph networks. *IEEE access*, *12*, 157630-157656.

Alam, M. S., Islam, M. R., Fan, R., Alam Shazid, M. S., & Hasan, A. S. (2025). Attention-guided multi-task learning for fault detection, classification, and localization in power transmission systems. *Energies*, *18*(24), 6547.

Abdian, A. Z., Javidi, M. M., Mansouri, N., Mehranfar, F., & Cheperli, S. (2026). MultiDentNet: a unified deep learning framework for multi-class dental condition screening and preliminary oral lesion triage: A. Zeydi Abdian et al. *Scientific Reports*.

Lakshminarayanan, A. C., & Thandavamoorthi, J. (2026). Attention-Enhanced Convolutional BiLSTM Model for Predicting Recovery Outcomes in Sports Injuries. *Indian Journal of Orthopaedics*, *60*(2), 355-377.

Wu, C., Wu, H., Dai, B., Yang, J., & Wu, J. (2025, November). Multi-label Vulnerability Detection for Go Smart Contracts with Call Graphs and Semantic Code Embeddings. In *International Conference on Financial Technology* (pp. 133-146). Singapore: Springer Nature Singapore.

Zheng, S., Zhang, Y., & Chen, Y. (2024). Leveraging financial sentiment analysis for detecting abnormal stock market volatility: An evidence-based approach from social media data. *Academia Nexus Journal*, *3*(3).

IEER
Institute for Excellence in Education & Research

Kouraichi, M., Mansouri, M., Trabelsi, M., M'Halla, A., Abdel-Khalik, A. S., & Sakly, A. (2025). Deep learning for fault diagnosis in power transmission lines: Current trends, limitations, and future directions. *IEEE Access*, *13*, 192105-192142.

Ozdag, M. (2025). AI-Driven Vulnerability Analysis in Smart Contracts: Trends, Challenges and Future Directions. *arXiv preprint arXiv:2506.06735*.

IEER
Institute for Excellence in Education & Research