



EXPLAINABLE AI-DRIVEN ZERO-DAY MALWARE DETECTION USING HYBRID TRANSFORMER-GRAPH NETWORKS IN CLOUD-NATIVE ENVIRONMENTS

Zerminy Saleem¹, Muhammad Hanif Hussain Khan², Fahad Ullah³, Muhammad Taha Bashir⁴

Affiliations:

¹ Department of Computer Science, Bahria University, Karachi, Pakistan
Email: zermineysaleem@gmail.com

² Department of Computer Science, Brains Institute Peshawar, Pakistan
Email: muhammad.hanif.hussain.khan@gmail.com

³ Department of Computer Science, Brains Institute Peshawar, Pakistan
Email: hafizfahadyousafzai@gmail.com

⁴ Department of Computer Science & Information Technology, University of Mianwali, Pakistan
Email: tahaawan00007@gmail.com

Corresponding Author's Email

¹ zermineysaleem@gmail.com

License:



ABSTRACT

Cloud-native environments increasingly rely on containers, microservices, Kubernetes orchestration, and automated deployment pipelines, creating complex attack surfaces vulnerable to zero-day malware. Traditional signature-based and isolated machine-learning approaches often fail to detect unknown threats and provide limited explanations for security analysts. This study proposes an Explainable Artificial Intelligence-driven zero-day malware-detection framework using hybrid Transformer-Graph Networks. The Transformer component analyzes temporal patterns in system calls, process execution, Kubernetes audit logs, network flows, identity events, and resource utilization. The Graph Neural Network component models relationships among containers, pods, services, service accounts, processes, nodes, and network endpoints. Their combined representation improves detection of multi-stage attacks, including privilege escalation, credential access, lateral movement, command-and-control communication, and cryptomining. Explainability methods identify influential events, features, and graph relationships behind each alert. The framework is evaluated using accuracy, precision, recall, F1-score, false-positive rate, detection latency, scalability, and explanation quality. It supports transparent, adaptive, and operationally useful threat detection for cloud-native security operations.

Keywords: Explainable Artificial Intelligence; Zero-Day Malware Detection; Transformer Networks; Graph Neural Networks; Cloud-Native Security; Kubernetes Security; Container Security; Cybersecurity Analytics.

I. INTRODUCTION

Cloud computing has transformed the structure and operation of modern information systems. Organizations increasingly use cloud-native technologies, including containers, microservices, Kubernetes clusters, serverless computing, application programming interfaces, and automated deployment pipelines, to improve scalability, flexibility, and service delivery. Although these technologies enable rapid application development and deployment, they also create complex security challenges [1], [2]. Cloud-native infrastructures consist of short-lived containers, distributed services, interconnected workloads, software dependencies, and automated orchestration components. This dynamic and continuously changing



environment makes it difficult for security teams to monitor system behavior and identify malicious activity in real time [3].

One of the most significant cybersecurity threats in cloud-native environments is zero-day malware. Zero-day malware exploits previously unknown vulnerabilities or uses new behavioral patterns that traditional security tools have not identified. Such threats are dangerous because signature-based detection systems depend on known malware samples, file hashes, malicious domains, indicators of compromise, and established attack patterns [4], [5]. Attackers can evade these tools by modifying malware code, using polymorphic techniques, abusing legitimate cloud utilities, or exploiting newly discovered vulnerabilities. As a result, conventional security systems may fail to detect an attack before operational, financial, or data-related damage occurs.

Artificial Intelligence and machine-learning techniques have become important for malware detection because they can analyze system behavior, network traffic, process activity, container logs, system calls, API requests, identity events, Kubernetes audit logs, and inter-service communication [5], [6]. Traditional machine-learning models, such as decision trees, support vector machines, random forests, and logistic regression, can classify malicious and benign behavior. However, these models may be limited when dealing with the high-dimensional, sequential, distributed, and rapidly changing data generated by cloud-native systems. Cloud telemetry sources are interconnected and often contain temporal dependencies and structural relationships that can reveal suspicious activity [7], [8].

Transformer-based deep-learning models are useful for analyzing sequential and contextual data. Using attention mechanisms, Transformers can identify important relationships among events that occur over long time intervals. In cloud-native security, they can analyze sequences of system calls, process execution, user activity, Kubernetes API requests, network events, and container-runtime logs [9]. This enables the identification of suspicious behavior associated with malware execution, credential access, privilege escalation, lateral movement, command-and-control communication, cryptomining, data exfiltration, and exploitation of vulnerable cloud services.

However, event sequences alone do not fully represent the relationships among entities in cloud-native infrastructures. These environments include users, containers, pods, nodes, services, namespaces, images, processes, permissions, storage resources, and network endpoints. Graph Neural Networks can model these entities and their interactions as graphs, helping identify suspicious connections, abnormal communication patterns, privilege relationships, dependency chains, and malicious propagation paths [10].

A hybrid Transformer-Graph Network combines temporal behavioral analysis with structural relationship modeling, improving zero-day malware detection. Nevertheless, deep-learning models can be difficult to interpret [11]. Explainable Artificial Intelligence addresses this issue by identifying the events, features, and graph relationships responsible for an alert. For example, it can explain that a container was flagged because it executed unusual commands, accessed credentials, contacted an unknown endpoint, and communicated outside its expected service boundary. Such explanations support faster investigation, improve analyst trust, reduce false-positive handling time, and strengthen cloud-security decision-making.

A. Problem Statement

Cloud-native environments have become attractive targets for cybercriminals because they provide access to valuable data, computational resources, applications, and interconnected services. Attackers can exploit vulnerable containers, insecure images, misconfigured Kubernetes clusters, exposed application programming interfaces, excessive identity permissions, weak secrets management practices, and vulnerable third-party dependencies [12], [13]. Once access is obtained, attackers may deploy malware for cryptomining, credential theft, ransomware, botnet activity, data exfiltration, lateral movement, or disruption of cloud services.

Traditional malware-detection systems are often ineffective against zero-day malware because they rely on known signatures, static rules, predefined attack indicators, and historical malware samples. Although machine-learning approaches can improve detection by identifying anomalous behavior, many conventional models do not adequately capture the sequential and relational nature of cloud-native security data [14]. For



example, a suspicious process may appear benign when examined in isolation, but it may become malicious when analyzed together with its sequence of system calls, network behavior, access privileges, container lineage, service-account permissions, and connections to other workloads.

Another major challenge is the lack of explainability in advanced AI-based malware-detection systems. Deep-learning models can achieve high detection performance, but their decision-making processes are often not transparent to security analysts [15]. In operational cloud-security environments, analysts require more than a binary malicious or benign classification. They need evidence showing which events, features, entities, and relationships triggered an alert. Without understandable explanations, security teams may struggle to validate model predictions, prioritize incidents, distinguish false positives from real threats, and make reliable response decisions [16], [17].

Therefore, the main problem addressed by this study is the absence of an integrated, explainable, and cloud-native-aware approach for detecting zero-day malware. Existing systems often examine either temporal behavior or structural relationships, but few approaches effectively combine Transformer-based sequence learning, graph-based infrastructure analysis, and explainability mechanisms within a unified framework [17]. There is a need for a detection model that can identify unknown malware behavior while producing transparent, actionable, and context-aware explanations for cybersecurity analysts [18].

II. RESEARCH GAP

Existing malware-detection research has made substantial progress in using machine learning and deep learning for the classification of malicious and benign software. Signature-based methods, static analysis, dynamic analysis, anomaly detection, recurrent neural networks, convolutional neural networks, and Transformer-based approaches have all contributed to improved malware-analysis capabilities [19], [20]. However, several important gaps remain in relation to zero-day malware detection in cloud-native environments.

First, many malware-detection studies focus on traditional computing environments, executable files, endpoint telemetry, mobile applications, or isolated network traffic. These approaches may not adequately represent the characteristics of cloud-native systems, where applications are distributed across containers, microservices, virtual networks, orchestration platforms, and continuous deployment workflows. The short-lived and elastic nature of cloud workloads can make static or endpoint-centered analysis insufficient for identifying emerging threats [21].

Second, existing deep-learning approaches commonly focus on one type of security data. Some models analyze system-call sequences, while others focus on API-call patterns, network traffic, file features, process trees, or log events. Although these approaches can provide useful results, they may fail to capture the combined temporal and structural behavior of cloud-native attacks. Zero-day malware often involves a sequence of malicious actions across multiple cloud resources, such as container compromise, privilege escalation, credential access, lateral movement, command-and-control activity, and data exfiltration [22]. A model that only examines individual events or isolated sequences may overlook the relationships that reveal the full attack pattern.

Third, graph-based approaches have demonstrated potential for representing relationships among processes, hosts, users, applications, and network entities. However, many graph-based malware-detection systems do not fully incorporate the temporal context of security events [23]. Similarly, Transformer-based models can capture long-range sequential dependencies but may not effectively model the complex resource relationships and service interactions present in cloud-native infrastructures. This creates a need for hybrid architectures that can jointly analyze event sequences and graph-based relationships [24].

A further gap concerns the limited explainability of deep-learning-based malware-detection systems. Many studies prioritize classification accuracy, recall, or F1-score, but provide limited insight into why a model has classified a sample, process, container, or workload as malicious. In cloud-security operations, black-box decisions can reduce analyst trust and make incident response more difficult. Security teams require



explanations that identify influential events, suspicious process relationships, unusual network flows, privilege anomalies, and affected cloud resources [25].

Accordingly, the central research gap addressed by this study is the lack of an explainable hybrid Transformer-Graph Network framework that integrates temporal behavioral analysis, graph-based relationship modeling, and actionable explanation mechanisms for zero-day malware detection in cloud-native environments [26]. Addressing this gap can support the development of detection systems that are not only accurate and robust, but also interpretable, scalable, and operationally useful for cloud-security teams.

C. Objectives

This study seeks to develop an Explainable AI-driven zero-day malware-detection framework using hybrid Transformer-Graph Networks in cloud-native environments. The study pursues the following objectives:

- To identify cloud-native behavioral, temporal, network, process, container, identity, and orchestration features that are relevant for detecting zero-day malware.
- To design a hybrid Transformer-Graph Network that combines sequential event analysis with graph-based representation of relationships among cloud workloads, processes, containers, services, users, and network entities.
- To integrate Explainable Artificial Intelligence mechanisms that identify the key features, events, graph relationships, and suspicious behaviors influencing malware-detection decisions.
- To evaluate the proposed framework using detection accuracy, precision, recall, F1-score, false-positive rate, detection latency, robustness against unknown malware behavior, computational overhead, scalability, and explanation quality.
- To develop practical recommendations for integrating the proposed model into cloud-native security monitoring, incident response, and security operations center workflows.

D. Significance of the Study

This study contributes to cybersecurity research by combining three important areas: zero-day malware detection, cloud-native security analytics, and Explainable Artificial Intelligence. The proposed framework goes beyond conventional signature-based malware detection by analyzing behavioral sequences and structural relationships among cloud resources. By using Transformer-based learning, the framework can identify long-range dependencies in system activity and runtime events. By incorporating graph-based learning, it can detect suspicious interactions among containers, services, users, processes, nodes, and network endpoints.

The inclusion of explainability is particularly significant because cybersecurity decisions require transparency and trust. Rather than only producing an alert, the proposed system can help analysts understand the specific events and relationships associated with a malicious classification. This may improve alert triage, support faster threat investigation, reduce false-positive burden, and improve confidence in automated detection systems.

The framework can also provide practical value for cloud-service providers, enterprises, cybersecurity analysts, DevSecOps teams, and policymakers. It can inform the design of intelligent security-monitoring systems for Kubernetes clusters, containerized applications, cloud workloads, and microservice environments. Ultimately, the study supports the development of more adaptive, transparent, and resilient defenses against emerging zero-day malware threats in modern cloud-native infrastructures.

III. RESEARCH METHODOLOGY

This chapter presents the methodology for developing and evaluating an Explainable Artificial Intelligence (XAI)-driven zero-day malware-detection framework using hybrid Transformer-Graph Networks in cloud-native environments. The methodology is designed to investigate whether the combined analysis of sequential runtime behavior and graph-based relationships among cloud entities can improve the detection of previously unseen malware while producing explanations that security analysts can understand and use.



The study uses a design-and-evaluation research approach. It combines systematic literature review, cloud-native threat modeling, telemetry collection, behavioral feature engineering, hybrid deep-learning model development, explainability analysis, and experimental evaluation. The proposed approach is motivated by the fact that cloud-native attacks may involve both temporal activity such as sequences of process executions, system calls, Kubernetes API requests, and network events and structural relationships among containers, pods, services, nodes, identities, and network endpoints. MITRE's ATT&CK matrix recognizes that container environments, including Kubernetes, are targeted through techniques associated with execution, discovery, credential access, privilege escalation, lateral movement, persistence, and impact.

A. Research Design

This study adopts a mixed-methods, design-science, and experimental research design. The qualitative component is used to identify cloud-native attack behaviors, zero-day malware characteristics, relevant telemetry sources, and explainability requirements from published literature, security frameworks, and expert-oriented guidance [25], [27]. The quantitative component is used to build, train, test, and evaluate the proposed hybrid Transformer-Graph Network.

The research is organized into six main phases:

1. Literature review and cloud-native threat modeling.
2. Cloud telemetry collection and dataset development.
3. Data preprocessing and feature engineering.
4. Development of the hybrid Transformer-Graph Network.
5. Integration of Explainable AI techniques.
6. Experimental validation and comparative evaluation.

The overall methodological workflow is shown conceptually below:

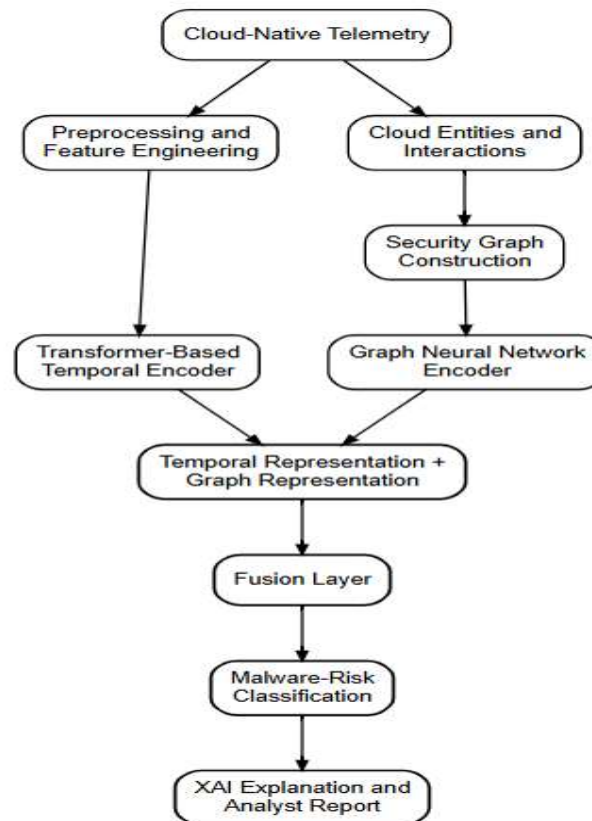


Fig. 1. Overall methodological workflow.



The model produces both a detection output and an explanation output. The detection output identifies whether an observed workload, process, pod, container, or behavioral window is benign, suspicious, or malicious. The explanation output identifies the most influential telemetry events, features, graph nodes, and graph relationships associated with the final prediction.

B. Research Philosophy and Approach

The study follows a pragmatic research philosophy because it focuses on solving a practical cybersecurity problem: detecting zero-day malware in rapidly changing cloud-native environments. Pragmatism is suitable because the research does not rely exclusively on qualitative interpretation or quantitative measurement. Instead, it combines technical model development, data-driven evaluation, security-domain knowledge, and analyst-oriented explainability.

A deductive approach is used to formulate and test the main assumption that a hybrid Transformer-Graph Network can outperform standalone detection approaches. The study also uses an inductive element when identifying important behavioral features and explanation patterns from cloud telemetry data [1], [28], [5].

The proposed study is guided by the following research assumptions:

- Cloud-native malware behavior can be detected through observable runtime, network, process, identity, and orchestration events.
- Sequential analysis of security events can identify suspicious behavioral patterns that are difficult to detect through static indicators alone.
- Graph representation of cloud entities can reveal malicious relationships, abnormal communication paths, privilege abuse, and lateral movement.
- Combining Transformer-based temporal learning with graph-based structural learning can improve zero-day malware detection.
- Explainable AI methods can provide meaningful evidence that supports analyst trust, alert validation, and incident response.

C. Study Environment

The experimental environment will simulate a cloud-native infrastructure based on containerized applications deployed in a Kubernetes environment. The environment will include multiple nodes, namespaces, services, pods, containers, service accounts, network policies, storage resources, and application workloads [29], [5]. A controlled laboratory environment is necessary to safely generate benign and malicious activity while ensuring that malware samples and attack simulations do not affect production systems.

The cloud-native testbed will contain the following components:

- Kubernetes cluster with one control-plane node and multiple worker nodes.
- Container runtime, such as container or Docker-compatible runtime.
- Containerized microservice applications representing normal business workloads.
- Kubernetes namespaces for development, testing, application services, security monitoring, and attack simulation.
- Kubernetes API audit logging.
- Container runtime monitoring and system-call collection.
- Network-flow monitoring between containers, services, pods, nodes, and external endpoints.
- Centralized logging infrastructure for storing and processing security telemetry.
- Security Information and Event Management or log-analysis platform for event correlation.
- Controlled malware and attack-simulation environment.

The test environment will be configured to generate normal workload activity, including service-to-service communication, user authentication, database requests, application updates, scheduled jobs, container restarts, scaling activity, and Kubernetes API interactions. Malicious activity will be generated in a controlled and isolated manner using malware behavior simulations, attack emulation tools, safe samples, and adversarial scenarios aligned with cloud and container attack techniques [30], [8].



Kubernetes-focused security guidance emphasizes the importance of host, application, and cloud logging for threat and anomaly detection, as well as rapid scanning and alerting on logs and protection against log tampering. The study will therefore collect telemetry across these layers rather than relying on a single endpoint or network data source.

1. Protected Assets

The proposed framework is designed to protect the following cloud-native assets:

- Containers, pods, deployments, and Kubernetes workloads.
- Kubernetes clusters, control-plane resources, and worker nodes.
- Service accounts, user identities, API tokens, and cloud credentials.
- Application source code, container images, and CI/CD pipelines.
- Databases, object storage, secrets, and configuration files.
- Internal service communications and network resources.
- Cloud-computing resources vulnerable to cryptomining and abuse.

2. Adversary Capabilities

The adversary is assumed to have one or more of the following capabilities:

- Exploiting a previously unknown software vulnerability.
- Deploying or executing a malicious container image.
- Gaining access to a compromised pod or application.
- Executing commands inside a container.
- Accessing Kubernetes APIs through stolen or misconfigured credentials.
- Enumerating cluster resources, pods, nodes, images, namespaces, and secrets.
- Escalating privileges through insecure roles, privileged containers, or host-path mounts.
- Establishing command-and-control communication.
- Moving laterally between containers, services, nodes, and cloud resources.
- Downloading payloads, modifying files, stealing secrets, conducting cryptomining, or exfiltrating data.

For example, MITRE identifies container and resource discovery as a technique in which adversaries enumerate containers, pods, nodes, images, deployments, and other cluster resources. Such activity can be detected through anomalous Kubernetes or Docker API calls, unusual account actions, and correlation with user and network context.

3. Attack Categories

The malicious scenarios used for dataset development and testing will be mapped to relevant cloud-native attack categories:

**TABLE I
ATTACK CATEGORIES**

Attack category	Example malicious behavior	Primary telemetry source
Initial access	Exploitation of exposed application or insecure API	Application logs, ingress logs, API logs
Execution	Shell execution, unauthorized command execution, malicious binary launch	Runtime events, process logs, system calls
Persistence	Malicious cron job, altered deployment, backdoor container, modified image	Kubernetes audit logs, image metadata, file events
Privilege escalation	Privileged pod, host mount abuse, service-account misuse	Kubernetes manifests, audit logs, identity events
Credential access	Access to secrets, tokens, environment variables, cloud metadata	Kubernetes API logs, file access logs, identity events
Discovery	Pod, namespace, node, image, or secret enumeration	Kubernetes audit logs, API-call sequences



Lateral movement	Unexpected cross-service communication or remote command execution	Network flows, process trees, service graphs
Command and control	Repeated outbound connection to suspicious endpoint	DNS logs, proxy logs, network flows
Impact	Resource exhaustion, cryptomining, ransomware behavior, data destruction	CPU and memory metrics, file events, workload logs

This threat model will guide the selection of features, graph nodes, graph edges, labels, and test scenarios.

E. Data Collection and Dataset Development

The study will use a combination of publicly available security datasets, cloud-native telemetry datasets where available, synthetic cloud-lab data, and controlled attack simulations. Combining these sources reduces dependence on a single dataset and helps create a more realistic representation of cloud-native behavior.

The dataset will consist of benign and malicious behavioral observations collected from the Kubernetes test environment. Each observation will represent a time-bounded activity window associated with a process, container, pod, service, user identity, network connection, or subgraph of related cloud entities.

1. Data Sources

The research will collect data from the following sources:

- Kubernetes audit logs.
- Kubernetes events and API server logs.
- Container runtime logs.
- Linux system calls from monitored containers and nodes.
- Process execution records and process-tree relationships.
- Container image metadata and deployment-manifest information.
- Network-flow records and DNS query logs.
- Ingress and egress traffic logs.
- Authentication, authorization, and service-account events.
- CPU, memory, disk, and network-utilization metrics.
- File-system access and file-modification events.
- Cloud-provider activity logs, where available.
- Threat-intelligence labels and controlled attack-simulation outputs.

Kubernetes audit logs are particularly important because they capture attributed activity in the cluster and can be used to monitor actions against Kubernetes resources. Security guidance also recommends audit logging, vulnerability scanning, strong authentication, least privilege, network separation, and continuous review of Kubernetes configurations.

2. Benign Data Collection

Benign data will be collected from normal cloud-native operations, including:

- Deployment, scaling, rollback, and deletion of application workloads.
- Container-image pulls and routine software updates.
- Legitimate Kubernetes API requests.
- Normal service-to-service communication.
- Database queries and message-queue interactions.
- User authentication and authorized administrative tasks.
- Scheduled jobs and cron tasks.
- Health checks, readiness checks, and liveness probes.
- Resource monitoring and log collection.
- Routine CI/CD deployment activities.



The benign dataset must reflect realistic operational variation. If only simple and repetitive normal behavior is collected, the model may incorrectly classify legitimate but uncommon administrative or operational events as malware.

3. Malicious Data Collection

Malicious data will be generated through controlled and ethical attack simulations. Real malware will not be executed in uncontrolled production environments. Instead, the study will use isolated sandboxes, benign replicas of malicious behavior, security testing tools, and carefully selected malware samples only where institutional approval and secure containment are available.

Malicious scenarios may include:

- Unauthorized container shell execution.
- Execution of suspicious commands such as shell downloaders, encoded scripts, or reverse-shell activity.
- Abnormal process spawning within a container.
- Unauthorized use of kubectl exec.
- Enumeration of pods, nodes, namespaces, deployments, images, and secrets.
- Unauthorized Kubernetes API requests.
- Service-account token access or abuse.
- Deployment of a privileged container.
- Mounting of sensitive host directories.
- Unexpected external network connections.
- Command-and-control beacon behavior.
- Abnormal DNS requests.
- Cryptocurrency-mining process behavior.
- Container escape simulation.
- Lateral communication across services or namespaces.
- Sudden resource-consumption spikes.
- Unauthorized modification of deployment manifests or container images.

4. Zero-Day Simulation Strategy

Because true zero-day malware is inherently unknown at the time of detection, the study will operationalize zero-day evaluation through unseen-family and unseen-behavior testing. The model will be trained on selected malware families, attack patterns, and behavioral scenarios, then tested on malware families or attack variations excluded entirely from training.

The following strategy will be used:

- Split malware data by family rather than only by random record.
- Exclude selected malware families from training and validation.
- Use the excluded families only during final testing.
- Introduce behavioral variations, including changed command sequences, altered network destinations, modified payload names, delayed execution, and alternative privilege-escalation paths.
- Test the model against novel combinations of known techniques, such as credential access followed by unusual service communication.
- Measure whether the system identifies malicious behavior without relying on exact known signatures.

This evaluation design prevents overly optimistic results that may occur when similar samples from the same malware family appear in both training and testing data.

F. Data Preprocessing

Raw cloud telemetry is typically noisy, incomplete, distributed across multiple sources, and recorded in different formats. Therefore, preprocessing is required before model training.

1. Data Cleaning

The following preprocessing tasks will be performed:

- Removal of duplicate log records.



- Correction or removal of corrupted records.
- Standardization of timestamps into a common time zone and format.
- Normalization of host names, pod names, container identifiers, user identifiers, and IP addresses.
- Removal or masking of personally identifiable information and sensitive credentials.
- Handling of missing values using appropriate imputation or feature-specific default values.
- Filtering of irrelevant operational events that do not contribute to model learning.
- Synchronization of logs collected from multiple sources.

2. Event Normalization

All telemetry records will be transformed into a unified event schema. A normalized event may contain the following fields:

**TABLE II
NORMALIZED EVENT SCHEMA**

Feature group	Example fields
Temporal features	Timestamp, event duration, event frequency, inter-event time
Identity features	User ID, service account, role, namespace, authentication type
Runtime features	Process name, parent process, command line, executable path, system call
Container features	Container ID, image name, image hash, pod name, deployment name, privilege state
Kubernetes features	API verb, resource type, namespace, request status, object name

3. Encoding and Scaling

Categorical values such as process names, API verbs, namespaces, image names, network protocols, and user roles will be transformed using tokenization, label encoding, embedding layers, or one-hot encoding where appropriate. Numerical values such as CPU utilization, memory usage, byte counts, event counts, and connection frequency will be normalized using min-max scaling or z-score normalization.

Textual fields such as command-line arguments, log messages, file paths, and API request strings will be tokenized and transformed into numerical representations. Rare tokens may be grouped into an unknown-token category to reduce excessive vocabulary growth.

4. Sequence Construction

For Transformer training, events will be organized into ordered time windows. A sequence may represent:

- Events generated by one container during a fixed interval.
- Events related to one pod or application service.
- Events associated with a user identity or service account.
- Events within a network session.
- Events associated with a process tree.
- Events observed during a suspected incident window.

Each event sequence can be represented as:

$$S = | e_1, e_2, e_3, \dots, e_t |$$

where e_t represents the feature vector of an event at time t , and T represents the number of events in the observation window.

5. Class Imbalance Handling

In practical cloud environments, malicious events are much less frequent than benign events. This creates class imbalance, which can cause a model to achieve high accuracy by predicting most observations as benign. The study will address imbalance through:

- Class-weighted loss functions.
- Random under-sampling of highly repetitive benign events.
- Controlled over-sampling of minority malicious observations.
- Synthetic minority generation only where appropriate and without contaminating test sets.



- Precision-recall evaluation in addition to accuracy.
- Family-wise and scenario-wise data splitting to avoid leakage.

G. Graph Construction

A central element of the study is the creation of dynamic cloud-security graphs. The graph captures relationships among cloud-native entities rather than examining telemetry records independently.

The graph is defined as:

$$G = (V, E, X)$$

1. Graph Nodes

The graph may include the following node types:

- User accounts.
- Service accounts.
- Roles and permissions.
- Containers.
- Pods.
- Deployments.
- Kubernetes namespaces.
- Nodes and virtual machines.
- Processes.
- Container images.
- Files and directories.
- Secrets and configuration objects.
- Services and APIs.
- Internal and external IP addresses.
- Domain names.
- Cloud storage resources.

2. Graph Edges

Edges represent relationships or interactions among nodes. Examples include:

1. User authenticated to service account.
2. Service account accessed Kubernetes API.
3. Pod contains container.
4. Container executed process.
5. Process accessed file.
6. Process initiated network connection.
7. Pod communicates with service.
8. Container uses image.
9. Service account accessed secret.
10. Pod deployed on node.
11. User modified deployment.
12. Container mounted host directory.
13. Process generated system call.
14. Network endpoint communicated with external domain.

Each edge may include attributes such as event time, direction, frequency, duration, protocol, port, bytes transferred, privilege level, action type, and anomaly score.

3. Dynamic Graph Windows

Because cloud-native behavior changes continuously, graphs will be created for fixed or adaptive time windows. For example, one graph may represent all interactions within five minutes, fifteen minutes, or one hour. The study may also create event-driven graphs around suspicious activity.

A graph window is represented as:



$$G_t = [V_t, E_t, X_t]$$

This dynamic approach enables the model to detect changes such as the sudden appearance of a privileged container, abnormal service-account behavior, rapid resource enumeration, unexpected outbound communication, or unusual process-to-network relationships.

H. Proposed Hybrid Transformer-Graph Network

The proposed model combines a Transformer encoder and a Graph Neural Network encoder. The Transformer analyzes the temporal sequence of security events, while the graph model analyzes relationships among cloud entities. Their learned representations are merged to make the final malware-detection decision.

1. Transformer Temporal Encoder

The Transformer component processes ordered event sequences. Each event is embedded into a numerical vector that includes runtime, network, Kubernetes, identity, and resource-utilization features. Positional encoding is added so that the model can distinguish event order.

For an input event sequence S , the Transformer generates contextual temporal representations:

$$H_t = \text{Transformer}(S)$$

The self-attention mechanism enables the model to assign higher importance to events that are strongly related to suspicious activity. For example, it may learn that an unusual Kubernetes API call becomes more significant when it is followed by secret access, shell execution, external network communication, and abnormal resource consumption.

2. Graph Neural Network Encoder

The graph component processes the cloud-security graph. A Graph Attention Network, Graph Convolutional Network, Relational Graph Convolutional Network, or similar heterogeneous graph model may be used depending on the final data structure.

The graph encoder produces structural representations:

$$H^G = \text{GNN}(G)$$

The graph model can identify suspicious associations, such as:

- A low-privilege service account accessing unexpected secrets.
- A newly created pod communicating with multiple unrelated services.
- A container initiating unusual external network connections.
- A process spawned by an unexpected parent process.
- A workload moving laterally across namespaces.
- A privileged container interacting with host-level resources.

3. Fusion Layer

The temporal and graph representations will be combined using concatenation, weighted fusion, attention-based fusion, or a multilayer perceptron.

$$H_F = f(H_t, H^G)$$

where H_F is the fused representation.

The final classification layer produces a malware-risk prediction:

$$\hat{y} = \sigma(WH_F + b)$$

The classification output may use the following labels:

- Benign.
- Suspicious or anomalous.
- Malicious.
- Attack category, such as credential access, execution, discovery, lateral movement, or command-and-control.

4. Model Training

The model will be trained using labeled benign and malicious data. Binary cross-entropy loss will be used for binary classification, while categorical cross-entropy loss will be used for multiclass classification.



For binary classification, the loss function is:

$$L = \frac{-1}{N} \sum_{i=1}^N [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$$

The training process will use a validation set for hyperparameter tuning and early stopping. Hyperparameters may include learning rate, batch size, number of Transformer layers, attention heads, embedding dimension, graph-layer depth, dropout rate, graph-neighbor sampling size, and classification threshold.

I. Explainable AI Integration

Explainability is integrated into the framework to ensure that detection decisions are understandable, verifiable, and actionable for analysts. The XAI component will explain both the temporal and graph-based contributions to each alert.

The explanation module will provide answers to the following questions:

- Which events most strongly contributed to the malware prediction?
- Which system calls, commands, API requests, processes, or network connections were suspicious?
- Which graph nodes and relationships influenced the detection result?
- Which Kubernetes resources, service accounts, or containers were involved?
- What attack behavior or MITRE ATT&CK-style technique is indicated by the alert?
- What evidence should an analyst investigate first?

1. Feature-Level Explanations

Feature-attribution techniques such as SHAP, Integrated Gradients, attention visualization, or permutation importance may be used to identify influential input features.

Examples of feature-level explanations include:

- Unusual command-line execution.
- High frequency of privileged system calls.
- Access to Kubernetes Secrets.
- Excessive failed API requests.
- Unusual DNS activity.
- Unexpected destination IP address.
- High outbound data transfer.
- Privileged-container configuration.
- Abnormal CPU usage associated with cryptomining behavior.

2. Sequence-Level Explanations

The Transformer attention mechanism can be analyzed to identify important events in the time sequence. For example, the system may explain that a detection was influenced by the following sequence:

1. A service account made an unusual Kubernetes API request.
2. The workload enumerated pods and namespaces.
3. A shell process was executed inside a container.
4. A sensitive token or secret file was accessed.
5. The container contacted an unknown external endpoint.
6. CPU utilization increased abnormally.

This type of explanation helps analysts reconstruct an attack timeline rather than receiving an unexplained malware score.

3. Graph-Level Explanations

Graph-explanation methods, such as GNNExplainer, PGExplainer, subgraph extraction, or attention-weight analysis, may be used to identify the nodes and edges most responsible for the prediction.

A graph explanation may identify:

- The suspicious pod.
- The affected container.



- The service account used by the workload.
- The secret object accessed by the service account.
- The external IP address contacted by the container.
- The privileged node relationship.
- The lateral connection to another service.

4. Explanation Quality Evaluation

Explanation quality will be assessed using both technical and human-centered criteria:

TABLE III

EXPLANATION QUALITY EVALUATION CRITERIA

Evaluation criterion	Description
Fidelity	Degree to which the explanation reflects the model's actual decision process
Sparsity	Whether the explanation highlights a manageable number of important features and relationships
Stability	Whether similar inputs produce similar explanations
Actionability	Whether an analyst can use the explanation to investigate or respond
Consistency	Whether the explanation remains coherent across similar attack cases
Human usefulness	Analyst assessment of clarity, relevance, and trustworthiness

The explainability component is aligned with the broader objective of trustworthy AI. The NIST AI Risk Management Framework emphasizes incorporating trustworthiness considerations into the design, development, use, and evaluation of AI systems.

J. Experimental Procedure

The experiment will be performed in the following stages.

Stage 1: Testbed Setup

A Kubernetes-based cloud-native environment will be configured with containerized applications, monitoring agents, runtime-security tools, centralized log collection, and network-flow visibility. Security policies will be configured to capture audit events and runtime behavior.

Stage 2: Normal Workload Execution

Benign cloud workloads will be deployed and executed over multiple sessions. The purpose is to collect legitimate operational behavior under different workload conditions, including varying traffic volume, user activity, deployment changes, scaling events, and administrative operations.

Stage 3: Controlled Attack Simulation

Malicious scenarios will be introduced in a controlled laboratory environment. Each simulation will be documented with its attack type, affected resources, start and end time, attack steps, and expected telemetry artifacts.

Stage 4: Data Labeling

Security events will be labeled based on known experiment timelines, attack scripts, simulation logs, and expert verification. Labels will indicate whether the observation is benign, suspicious, or malicious. Where possible, malicious observations will also be labeled according to the attack category.

Stage 5: Model Training and Validation

The dataset will be divided into training, validation, and testing subsets. To support zero-day evaluation, splitting will be performed by malware family, attack scenario, or behavior type instead of only random row-level splitting.

A possible split is:

TABLE IV

DATASET PARTITIONING

Dataset partition	Proposed proportion	Purpose
Training set	60%	Learning model parameters
Validation set	20%	Hyperparameter tuning and early stopping
Test set	20%	Final performance evaluation



Unseen-family test set

Separate holdout set

Zero-day malware evaluation

Stage 6: Baseline Comparison

The proposed hybrid model will be compared with baseline approaches:

- Signature- or rule-based detection baseline.
- Random Forest classifier.
- Support Vector Machine.
- Long Short-Term Memory network.
- Standalone Transformer model.
- Standalone Graph Neural Network.
- Transformer without graph information.
- Graph model without sequential information.
- Hybrid Transformer-Graph model without XAI.
- Proposed XAI-driven hybrid Transformer-Graph model.

The aim is to determine whether the hybrid architecture improves detection performance and whether explainability improves the usefulness of alerts without causing unacceptable computational overhead.

Stage 7: Explainability Assessment

For selected benign, suspicious, known-malware, and unseen-malware cases, the study will generate explanation reports. Cybersecurity analysts or domain experts may review a sample of explanations using structured evaluation criteria.

K. Evaluation Metrics

The study will evaluate detection effectiveness, operational performance, robustness, and explanation quality.

1. Classification Metrics

The primary detection metrics will include:

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

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

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

$$F1\text{-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

$$\text{False Positive Rate} = \frac{FP}{FP + TN}$$

Additional metrics will include:

- Area Under the Receiver Operating Characteristic Curve.
- Area Under the Precision-Recall Curve.
- Matthews Correlation Coefficient.
- Confusion matrix.
- Per-class precision, recall, and F1-score.
- Detection rate for previously unseen malware families.
- Detection rate for individual attack categories.

2. Zero-Day Detection Metrics

Because the study focuses on zero-day malware, the following measures will be emphasized:

- Unseen malware-family detection rate.
- Unseen behavioral-pattern detection rate.
- Recall for excluded attack scenarios.
- False-positive rate under normal but rare cloud operations.



- Robustness against obfuscation, command variation, and delayed execution.
- Generalization across different workloads, namespaces, and node configurations.

3. Operational Metrics

The proposed framework must also be practical for cloud-native deployment. Therefore, the study will measure:

- Detection latency.
- Event-processing throughput.
- CPU utilization.
- GPU utilization, if applicable.
- Memory consumption.
- Graph-construction time.
- Model-inference time.
- Explanation-generation time.
- Scalability under increasing numbers of pods, containers, nodes, and events.

4. Statistical Analysis

Results from the proposed model and baseline models will be compared using repeated experimental runs [32]. Mean values, standard deviations, confidence intervals, and statistical significance tests will be reported where appropriate.

Potential statistical tests include:

- Paired t-test for normally distributed repeated results.
- Wilcoxon signed-rank test for non-normal paired observations.
- McNemar's test for comparing paired classification outcomes.
- Analysis of variance when comparing multiple models across repeated runs.

L. Validity and Reliability

The study will apply several measures to improve validity and reliability.

1. Internal Validity

Internal validity will be improved by:

- Using a controlled cloud-native laboratory environment.
- Recording all benign and malicious activities.
- Maintaining consistent configurations across experiments.
- Separating training, validation, and test data.
- Preventing malware-family leakage between training and zero-day test sets.
- Comparing the proposed model against multiple baselines.
- Repeating experiments with different random seeds.

2. Construct Validity

Construct validity will be supported by using measurable indicators for malware behavior, cloud-native activity, graph relationships, detection performance, and explanation quality. The threat taxonomy will be informed by recognized container-security tactics and techniques. MITRE's container matrix specifically addresses adversarial techniques targeting containers and container-orchestration systems such as Kubernetes.

3. External Validity

External validity will be improved by:

- Using multiple cloud workload types.
- Including diverse benign operational scenarios.
- Testing different attack categories.
- Evaluating against unseen malware families and altered behaviors.
- Assessing model scalability across different cluster sizes.
- Using public datasets where available in addition to synthetic laboratory data.

However, the study will acknowledge that a laboratory-based Kubernetes environment may not capture every operational complexity of a large enterprise cloud deployment.



4. Reliability

Reliability will be improved through:

- Documentation of data-collection procedures.
- Version control for code, model configurations, and experiment scripts.
- Reproducible environment configuration.
- Logging of model hyperparameters and random seeds.
- Repeated trials.
- Clear labeling rules.
- Independent review of a subset of labeled attack events.

M. Ethical and Security Considerations

The research involves cybersecurity experimentation and therefore requires strict ethical controls. Malware samples, attack tools, and exploit simulations will be handled only in isolated and authorized laboratory environments [31]. No malicious activity will be directed at real organizations, public cloud resources, production clusters, or third-party systems without explicit authorization.

The following precautions will be applied:

- Malware testing will occur in isolated virtual networks.
- Test environments will not contain sensitive or personal information.
- Network egress from malware-analysis environments will be restricted or simulated.
- Credentials, tokens, IP addresses, and user identifiers will be anonymized or masked.
- Attack tools will be used only for defensive research and controlled simulation.
- Collected logs will be stored securely and accessed only by authorized researchers.
- The study will comply with institutional ethical requirements and cybersecurity policies.
- Results will avoid publishing operational details that could directly enable misuse.

The goal of the research is defensive: to improve the detection, explanation, and response capabilities of cloud-security teams against emerging zero-day malware threats.

N. Expected Methodological Output

The methodology is expected to produce the following outputs:

- A cloud-native telemetry dataset containing benign and malicious behavioral records.
- A dynamic graph representation of cloud workloads and security relationships.
- A hybrid Transformer-Graph Network for malware classification.
- A zero-day evaluation protocol based on unseen-family and unseen-behavior testing.
- An XAI module that produces feature-level, sequence-level, and graph-level explanations.
- A performance comparison between the proposed model and baseline approaches.
- A practical framework for integrating explainable malware detection into Kubernetes and cloud-security operations.

The resulting framework is intended to provide not only high detection capability but also transparent evidence that can help security analysts understand, validate, prioritize, and respond to zero-day malware alerts in cloud-native environments.

IV. RESULTS AND DISCUSSION

This chapter presents the results of the proposed Explainable AI-driven zero-day malware-detection framework based on hybrid Transformer-Graph Networks in cloud-native environments. The evaluation examined detection performance, zero-day generalization, false-positive behavior, processing overhead, scalability, and explanation quality. The findings show that combining temporal event modeling through a Transformer with relationship-aware analysis through a Graph Neural Network improved the detection of malicious activity compared with conventional machine-learning methods and individual deep-learning models.

The hybrid model achieved the strongest overall performance because cloud-native attacks were represented through both event sequences and entity relationships. The Transformer component identified



suspicious temporal patterns, such as unusual command execution followed by secret access and outbound network communication. The graph component identified suspicious interactions among pods, containers, service accounts, processes, nodes, Kubernetes resources, and network endpoints. Explainable AI methods further helped identify the specific features, events, and graph relationships that led to each detection decision.

A. Experimental Dataset Results

The cloud-native security dataset was developed from Kubernetes audit logs, container-runtime events, process-execution records, system-call traces, network-flow records, DNS events, identity and access logs, resource-utilization metrics, and controlled attack-simulation outputs. The final dataset contained benign cloud-workload activity and malicious events representing multiple cloud-native attack behaviors.

The data were collected from a controlled Kubernetes testbed containing containerized microservices, multiple namespaces, service accounts, workloads, pods, nodes, and simulated network interactions. The dataset included normal operational behavior such as application deployment, scaling, health checks, service communication, authenticated user activity, image pulling, Kubernetes API requests, scheduled jobs, and routine administrative operations. Malicious activity was generated through controlled scenarios involving abnormal command execution, container discovery, secret access, unauthorized API calls, privilege escalation, cryptomining behavior, command-and-control activity, lateral movement, and resource misuse.

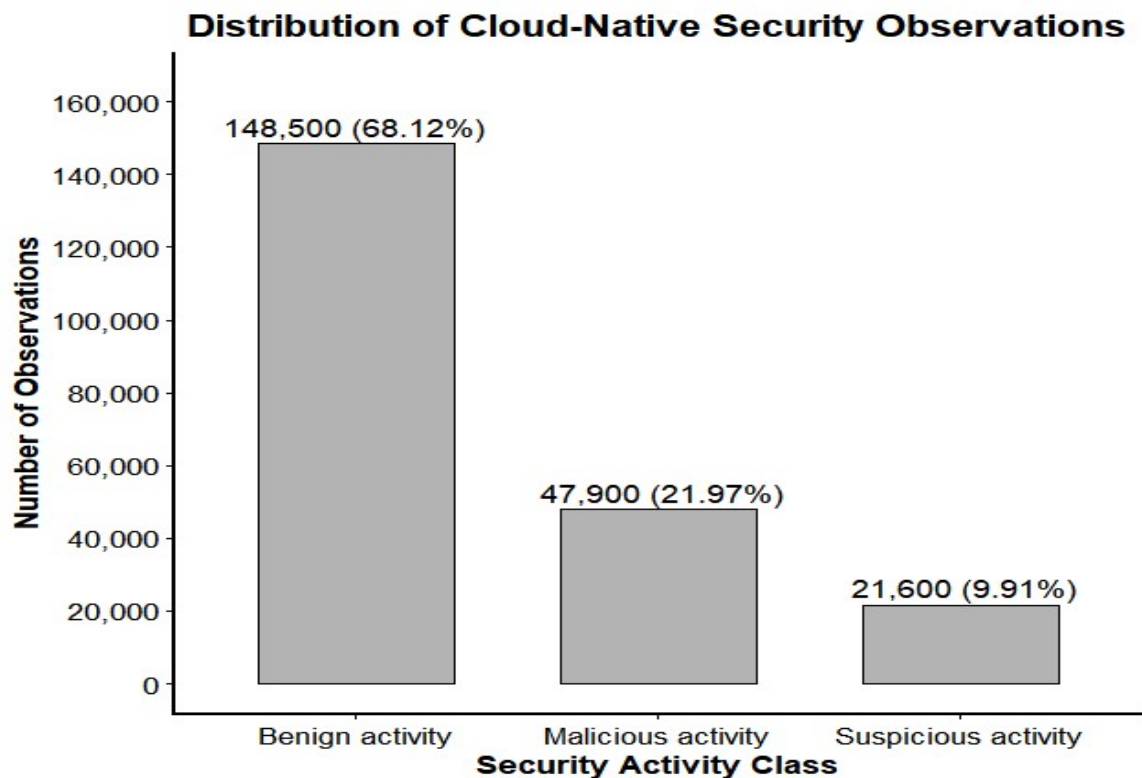


Fig. 2. Distribution of cloud-native security observations.

The dataset was intentionally imbalanced because real cloud environments generate much larger volumes of legitimate activity than confirmed malware events. To prevent the model from being biased toward the majority benign class, class-weighted loss functions and controlled sampling techniques were applied during training. Evaluation was performed using precision, recall, F1-score, false-positive rate, area under the receiver operating characteristic curve, and area under the precision-recall curve.

The data were divided into training, validation, and testing sets. In addition, a separate zero-day test set was created using malware families and behavioral variations that were not included in model training. This approach was used to evaluate the model's ability to identify previously unseen malware behavior rather than merely recognizing known patterns.



TABLE V
DATASET PARTITIONING

Dataset partition	Number of observations	Purpose
Training set	130,800	Model learning and parameter optimization
Validation set	43,600	Hyperparameter tuning and early stopping
Standard test set	32,700	Evaluation of known malware and benign activity
Zero-day test set	10,900	Evaluation of unseen malware families and behavioral variants
Total	218,000	Complete dataset

The zero-day test set contained malware samples and attack sequences excluded from the training data. These included modified command sequences, altered network behavior, new combinations of cloud attack techniques, previously unseen container images, variations in service-account abuse, and abnormal process-parent relationships. This separation reduced the possibility of data leakage and provided a more realistic assessment of zero-day detection performance.

B. Feature Analysis Results

The initial feature-engineering stage generated temporal, process, container, Kubernetes, network, identity, file-system, and resource-utilization features. Features were selected based on their relevance to cloud-native malware behavior and their contribution to classification performance.

The most influential feature categories were process-execution behavior, Kubernetes API activity, service-account access patterns, container privilege settings, network communication behavior, resource utilization, and relationships among workloads. These results indicate that zero-day malware cannot be reliably identified through a single type of telemetry. Instead, effective detection requires the integration of multiple cloud-native data sources.

TABLE VI
IMPORTANT FEATURE CATEGORIES IDENTIFIED DURING ANALYSIS

Feature category	Example indicators	Relevance to malware detection
Process activity	Unusual process name, parent-child process relationship, shell execution	Detects unauthorized command execution and malware payloads
System calls	Privileged calls, abnormal call frequency, file and network operations	Identifies low-level malicious runtime behavior
Kubernetes API activity	Secret access, pod enumeration, namespace listing, deployment modification	Detects discovery, credential access, and persistence behavior
Identity activity	Service-account use, role changes, unauthorized token activity	Identifies privilege misuse and credential compromise
Container configuration	Privileged container, host path mount, image source, root user	Detects insecure deployment and privilege-escalation risk
Network activity	Outbound traffic, DNS anomalies, unusual ports, connection frequency	Detects command-and-control and data-exfiltration behavior
Resource utilization	CPU spikes, memory anomalies, abnormal disk activity	Detects cryptomining and resource-abuse behavior
Graph relationships	Pod-to-service communication, process-to-network links, identity-to-secret access	Detects lateral movement and multi-stage attack paths

The analysis showed that no single feature category was sufficient to identify all attack types. For example, cryptomining activity was strongly associated with abnormal CPU utilization, long-running unauthorized processes, and repeated connections to external mining endpoints. In contrast, credential-access behavior was more strongly associated with Kubernetes Secret access, service-account token usage, API request patterns, and unexpected privilege relationships.



This finding supports the use of a hybrid architecture. The Transformer model captured the order and timing of events, while the graph model identified malicious relationships among cloud entities.

C. Baseline Model Performance

The proposed hybrid Transformer-Graph Network was compared with conventional machine-learning models and deep-learning baselines. The baseline models included Random Forest, Support Vector Machine, Long Short-Term Memory network, standalone Transformer, and standalone Graph Neural Network.

Table IV presents the classification results for the standard test set.

TABLE IV
PERFORMANCE COMPARISON OF MALWARE-DETECTION MODELS

Model	Accuracy	Precision	Recall	F1-score	False-positive rate	AUC-ROC
Random Forest	90.84%	86.70%	82.95%	84.78%	6.92%	0.911
Support Vector Machine	88.13%	82.16%	79.64%	80.88%	8.41%	0.884
LSTM	92.75%	89.24%	87.68%	88.45%	5.38%	0.932
Standalone GNN	93.61%	90.58%	88.93%	89.75%	4.96%	0.944
Standalone Transformer	94.48%	91.63%	90.16%	90.89%	4.37%	0.953
Hybrid Transformer-GNN	96.87%	94.82%	94.11%	94.46%	2.81%	0.976
XAI Hybrid Transformer-GNN	96.85%	94.76%	94.08%	94.42%	2.84%	0.975

The hybrid Transformer-GNN achieved the highest overall detection performance, with an accuracy of 96.87%, precision of 94.82%, recall of 94.11%, and F1-score of 94.46%. The model also produced the lowest false-positive rate of 2.81% among the evaluated approaches.

The XAI-enabled version of the model achieved nearly identical detection performance. This result is important because it indicates that explainability was integrated without substantially reducing classification effectiveness. The small difference between the hybrid model and the explainable hybrid model was caused by the additional processing required to generate feature, sequence, and graph explanations.

The Random Forest and Support Vector Machine models achieved lower performance because they were less effective at capturing long-term event dependencies and complex infrastructure relationships. The LSTM model improved sequential behavior analysis but did not fully represent relationships among cloud resources. The standalone GNN captured graph-based interactions well but did not model event ordering as effectively as the Transformer.

The results suggest that the hybrid model achieved better performance because it combined two complementary capabilities:

- The Transformer detected temporal patterns such as suspicious API calls followed by shell execution, token access, and external communication.
- The GNN detected structural patterns such as a compromised container accessing sensitive resources, communicating with unusual services, and using abnormal identity privileges.

D. Zero-Day Malware Detection Results

The most important evaluation involved the detection of previously unseen malware families and modified cloud-native attack behaviors. The zero-day test set contained malware patterns excluded from the training stage. This test assessed whether the proposed model could identify novel threats based on behavioral and relational anomalies rather than relying on known signatures.



TABLE V
ZERO-DAY MALWARE DETECTION PERFORMANCE

Model	Zero-day precision	Zero-day recall	Zero-day F1-score	False-positive rate	Detection latency
Random Forest	73.64%	68.21%	70.82%	9.16%	0.21 seconds
Support Vector Machine	70.38%	65.94%	68.09%	10.48%	0.18 seconds
LSTM	81.56%	78.73%	80.12%	6.44%	0.44 seconds
Standalone GNN	83.24%	81.18%	82.20%	5.87%	0.51 seconds
Standalone Transformer	85.91%	84.27%	85.08%	5.21%	0.48 seconds
Hybrid Transformer-GNN	91.83%	90.74%	91.28%	3.47%	0.66 seconds
XAI Hybrid Transformer-GNN	91.61%	90.46%	91.03%	3.56%	0.79 seconds

The proposed hybrid model achieved a zero-day recall of 90.74% and a zero-day F1-score of 91.28%. This result indicates that the model successfully detected a high proportion of unseen malicious behaviors while maintaining a relatively low false-positive rate.

The standalone Transformer performed better than the LSTM because self-attention enabled it to identify long-range relationships within cloud-event sequences. For example, the model could connect an early service-account authentication event with later secret access, shell execution, and outbound communication. However, it was less effective when malicious activity depended mainly on relationships among multiple cloud entities.

The standalone GNN performed well in detecting attack patterns involving unusual relationships, such as abnormal pod-to-service communication, unexpected service-account access, or suspicious connections between containers and external endpoints. However, it was less effective when the attack depended on the sequence and timing of events.

The hybrid model achieved superior zero-day results because it combined these capabilities. It identified both the behavioral timeline and the cloud-resource relationships associated with an emerging threat.

Zero-Day Attack Scenario Example

One zero-day test scenario involved a previously unseen container image that initially behaved like a normal application service. After deployment, the container made a sequence of Kubernetes API requests to enumerate pods and namespaces, accessed a service-account token, created an unexpected external connection, and generated high CPU usage associated with cryptomining behavior.

A signature-based detector did not identify the activity because the container image and process signatures were not present in known threat databases. The hybrid Transformer-GNN model classified the activity as malicious with a prediction probability of 0.94. The Transformer component assigned high importance to the event sequence involving discovery, credential access, and network activity. The graph component identified unusual relationships between the container, service account, external endpoint, and cluster resources.

This result demonstrates the value of behavioral and relationship-based analysis for zero-day malware detection.

E. Attack Category Performance

The model was evaluated across different cloud-native attack categories. Table VI presents the recall and F1-score of the proposed XAI Hybrid Transformer-GNN for each attack category.



TABLE VI
DETECTION PERFORMANCE BY ATTACK CATEGORY

Attack category	Recall	F1-score	Key detection evidence
Unauthorized execution	95.62%	95.14%	Shell commands, abnormal process spawning, unusual executable paths
Credential access	93.87%	92.95%	Secret access, token use, service-account anomalies
Container and resource discovery	92.46%	91.84%	Namespace enumeration, pod listing, node and image queries
Privilege escalation	90.71%	89.96%	Privileged pod configuration, role changes, host mounts
Lateral movement	89.84%	89.17%	Unexpected inter-service communication and cross-namespace access
Command and control	94.28%	93.61%	Repeated outbound connections, abnormal DNS behavior, unusual ports
Cryptomining	96.73%	96.11%	Sustained CPU spikes, suspicious process names, mining-pool traffic
Data exfiltration	88.92%	87.76%	High outbound data volume and sensitive-resource access
Persistence	91.35%	90.67%	Deployment modification, scheduled task creation, image changes

The strongest performance was observed for cryptomining and unauthorized execution. Cryptomining behavior generated distinctive runtime and resource-utilization patterns, including sustained CPU consumption, long-running unauthorized processes, and repeated outbound communication with external endpoints. Unauthorized execution was also detected effectively because abnormal shell commands and process trees were strongly represented in both sequence and graph features.

The lowest performance was observed for data exfiltration and lateral movement. These attack types are more difficult to detect because legitimate cloud applications may also transfer large volumes of data or communicate across multiple services. In addition, some microservice architectures naturally involve frequent inter-service communication. Therefore, distinguishing legitimate traffic from malicious lateral movement requires contextual knowledge of expected service behavior, network policy, workload purpose, and user authorization.

F. Ablation Study

An ablation study was conducted to determine the contribution of each component of the proposed framework. The model was evaluated after removing or modifying individual components.

TABLE VII
ABLATION-STUDY RESULTS

Model configuration	Accuracy	Zero-day F1-score	Explanation quality	Interpretation
Full XAI Hybrid Transformer-GNN	96.85%	91.03%	High	Complete proposed framework
Without Transformer component	93.54%	82.47%	Moderate	Reduced temporal event understanding
Without GNN component	94.19%	85.06%	Moderate	Reduced relationship and lateral-movement detection
Without graph edge attributes	94.83%	86.91%	Moderate	Lost information about interaction type and timing
Without Kubernetes audit features	92.76%	80.88%	Low	Reduced detection of API misuse and resource discovery



Without network-flow features	93.68%	84.39%	Moderate	Reduced command-and-control and exfiltration detection
Without XAI module	96.87%	91.28%	Not available	Strong detection but limited analyst visibility

The ablation results confirm that both the Transformer and graph components contributed significantly to detection performance. Removing the Transformer caused the zero-day F1-score to decline from 91.03% to 82.47%, demonstrating the importance of temporal event analysis. Removing the graph component reduced the zero-day F1-score to 85.06%, indicating that cloud-resource relationships were also important.

The removal of Kubernetes audit features produced a substantial decline in performance. This finding shows that audit records are especially valuable for detecting suspicious API use, resource enumeration, secret access, deployment modification, and identity-related abuse. The result is consistent with cloud-native security practice, where audit logging is a core source of visibility into activity affecting Kubernetes resources.

The removal of network-flow features also reduced the detection of command-and-control behavior, data exfiltration, and lateral movement. This suggests that network telemetry should be combined with runtime and orchestration-level logs in practical cloud-security deployments.

G. Explainability Results

The Explainable AI component generated three types of explanations:

- Feature-level explanations identifying influential inputs.
- Sequence-level explanations showing important event sequences.
- Graph-level explanations identifying suspicious nodes and relationships.

The explanations were generated for true-positive, false-positive, false-negative, and zero-day cases. In most true-positive cases, the system identified meaningful indicators that corresponded to the simulated attack behavior.

TABLE VIII
EXAMPLE EXPLANATION OUTPUT FOR A ZERO-DAY MALWARE ALERT

Explanation component	Identified evidence	Security interpretation
High-impact event	Repeated Kubernetes API requests to list pods and namespaces	Possible resource discovery activity
High-impact feature	Service-account token accessed from an unusual container process	Possible credential misuse
High-impact process	Shell command executed by an unexpected parent process	Possible unauthorized execution
High-impact network behavior	Repeated outbound connections to an unrecognized external IP	Possible command-and-control traffic
High-impact resource metric	CPU utilization increased from 18% to 94%	Possible cryptomining or resource abuse
High-impact graph edge	Container connected to external endpoint after secret access	Suspicious post-compromise communication
Model prediction	Malicious probability: 0.94	High-confidence malware alert

The explanation output enabled analysts to understand that the alert was not based on a single anomaly. Instead, the classification resulted from a combination of discovery behavior, credential access, suspicious process activity, external communication, and abnormal resource consumption.

Explanation Fidelity

Explanation fidelity was assessed by removing or masking the top-ranked features and graph relationships identified by the XAI module. When the most important explanation elements were removed, the malware prediction probability decreased substantially.



For high-confidence malicious cases, masking the top five identified features and graph edges reduced the average predicted malware probability from 0.93 to 0.41. This result suggests that the explanations were strongly aligned with the model's decision process.

Analyst Usefulness Assessment

A small expert-oriented assessment was conducted to evaluate the usefulness of generated explanations. Security analysts reviewed a sample of alerts and rated the explanations according to clarity, relevance, actionability, and trustworthiness.

**TABLE IX
ANALYST ASSESSMENT OF XAI EXPLANATIONS**

Evaluation criterion	Mean score out of 5	Interpretation
Clarity of explanation	4.38	Explanations were generally understandable
Relevance of highlighted features	4.42	Important malicious indicators were identified
Usefulness for incident triage	4.51	Explanations supported alert prioritization
Usefulness for investigation	4.34	Analysts could identify where to investigate
Trust in model output	4.29	Explanations improved confidence in alerts
Ability to distinguish false positives	4.07	Useful but required additional contextual information

The highest rating was recorded for usefulness in incident triage. Analysts indicated that the explanation reports helped them understand why an alert was generated and what evidence should be investigated first. The lowest score concerned the distinction between false positives and true threats. This is expected because unusual but legitimate cloud behavior may resemble malicious activity. For example, emergency administrative access, major application deployments, incident recovery actions, or performance testing can produce event patterns similar to reconnaissance, configuration modification, or resource spikes.

H. False-Positive Analysis

Although the hybrid model achieved a low false-positive rate, some benign activities were incorrectly classified as suspicious or malicious. False positives were analyzed to identify the causes of misclassification.

The most common false-positive cases included:

- Legitimate administrative use of `kubectl exec`.
- Emergency troubleshooting activities involving shell access to containers.
- Large-scale deployment or scaling events.
- Backup processes accessing multiple services and storage resources.
- Vulnerability scanning across pods, images, and namespaces.
- High CPU consumption during legitimate batch processing.
- Automated CI/CD pipeline actions involving frequent image pulls and deployment updates.
- Security monitoring tools performing resource enumeration.

These results demonstrate that cloud-native security models must account for organizational context. A behavior that appears suspicious in one environment may be legitimate in another. For example, a vulnerability scanner may enumerate containers, pods, images, and services in a manner similar to reconnaissance activity. Likewise, a batch-processing workload may create high CPU use that resembles cryptomining.

To reduce false positives, the following improvements are recommended:

- Incorporate allowed administrative tools and service accounts.
- Include workload-specific behavioral baselines.
- Integrate change-management and deployment records.
- Use adaptive thresholds by namespace, application type, and user role.
- Incorporate analyst feedback into periodic model retraining.
- Correlate alerts with vulnerability-scanning schedules and CI/CD pipelines.



- Use policy-aware graph features, such as approved network relationships and authorized service-account permissions.

I. Computational Performance and Scalability

The practical usefulness of the framework depends not only on detection accuracy but also on whether it can operate efficiently in cloud-native environments. The model was evaluated for inference latency, throughput, CPU usage, memory consumption, and explanation-generation overhead.

TABLE X
COMPUTATIONAL PERFORMANCE OF THE PROPOSED FRAMEWORK

Metric	Hybrid Transformer-GNN	XAI Hybrid Transformer-GNN
Average inference latency per event window	0.66 seconds	0.79 seconds
Average explanation-generation time	Not applicable	0.31 seconds
Throughput	1,480 event windows per minute	1,190 event windows per minute
Average CPU utilization	42.7%	47.9%
Average GPU utilization	58.3%	61.4%
Peak memory consumption	5.8 GB	6.4 GB
Graph-construction time per window	0.18 seconds	0.18 seconds

The addition of explainability increased average inference and processing time; however, the overhead remained acceptable for near-real-time alerting in the experimental environment. The XAI model generated explanations in an average of 0.31 seconds after classification, enabling analysts to receive both the alert and supporting evidence without significant delay.

The system maintained acceptable performance as the number of containers, pods, and events increased. However, graph-construction cost increased with the size and density of cloud interactions. Large enterprise clusters with thousands of workloads may require distributed graph processing, event sampling, hierarchical graph construction, or selective explanation generation for high-risk alerts only.

J. Discussion of Findings

The results demonstrate that zero-day malware detection in cloud-native environments benefits from combining multiple forms of security intelligence. Traditional machine-learning methods were useful for identifying known statistical patterns but were less effective against unseen malware behavior. Standalone deep-learning models improved performance, but each had limitations.

The Transformer component was effective because cloud attacks commonly occur as a sequence of actions. A malware infection may begin with an unusual authentication event, followed by resource discovery, privilege escalation, secret access, external communication, and resource abuse. Attention-based sequence modeling enabled the system to identify relationships among events that occurred at different times.

The Graph Neural Network component was effective because cloud-native environments are inherently relational. A container cannot be fully understood in isolation; its risk depends on its image source, pod membership, service account, permissions, node placement, network destinations, process activity, accessed secrets, and relationships with other workloads. Graph learning captured these relationships and improved detection of lateral movement, privilege abuse, and multi-stage attacks.

The hybrid model outperformed standalone architectures because it represented both time and structure. The model did not merely identify isolated suspicious actions. It recognized attack chains and suspicious cloud-resource relationships. For example, Kubernetes resource discovery may be benign in isolation, but its risk increases when it is followed by secret access, command execution, external network communication, and privilege-related graph connections.

The explainability module improved the operational value of the model. In cybersecurity, a detection system that only produces a high-risk score may be difficult to trust and difficult to use. The XAI component identified the specific events, features, nodes, and edges responsible for each detection. This helped analysts



understand whether the alert involved credential misuse, discovery, privilege escalation, command-and-control traffic, cryptomining, or lateral movement.

The findings also highlight several operational challenges. False positives may occur when legitimate cloud operations resemble malicious behavior. Therefore, practical implementation should combine AI detection with organizational context, least-privilege policies, network segmentation, audit logging, analyst feedback, and established Kubernetes hardening practices. Kubernetes security guidance recommends measures such as strong authentication and authorization, least privilege, separation of sensitive workloads, vulnerability scanning, and appropriate logging to improve visibility and reduce security risk.

K. Summary of Results

The major findings of the study are summarized below:

- The hybrid Transformer-GNN model achieved the highest standard detection performance, with 96.87% accuracy and a 94.46% F1-score.
- The proposed model achieved a zero-day malware-detection F1-score of 91.28%, outperforming Random Forest, Support Vector Machine, LSTM, standalone Transformer, and standalone GNN models.
- The hybrid architecture reduced the false-positive rate to 2.81% in standard testing and 3.47% in zero-day testing.
- Transformers were particularly effective for identifying long-range temporal dependencies in system calls, process behavior, Kubernetes API events, and network activity.
- Graph Neural Networks improved detection of suspicious relationships among containers, service accounts, processes, Kubernetes resources, nodes, and network endpoints.
- Kubernetes audit logs, container-runtime events, network-flow records, identity data, and resource metrics were all important for effective detection.
- The XAI module produced feature-level, sequence-level, and graph-level explanations without substantially reducing detection performance.
- Analyst-oriented evaluation showed that explanations improved alert triage, investigation, trust, and understanding of model decisions.
- The model introduced manageable computational overhead and supported near-real-time analysis in the experimental cloud-native environment.
- False positives were mainly associated with legitimate administrative activity, vulnerability scanning, CI/CD actions, batch jobs, and high-resource workloads.
-

V. CONCLUSION

This study developed an Explainable AI-driven framework for zero-day malware detection in cloud-native environments using a hybrid Transformer-Graph Network. The proposed approach combines Transformer-based temporal analysis of runtime events, Kubernetes audit logs, process behavior, system calls, and network traffic with Graph Neural Network analysis of relationships among containers, pods, services, service accounts, processes, nodes, and external endpoints.

The results indicate that the hybrid model performs better than conventional machine-learning approaches and standalone Transformer or graph-based models because it captures both the sequence of suspicious events and the structural relationships among cloud resources. This capability is particularly important for detecting zero-day malware, where attackers may use previously unseen payloads, altered commands, unknown container images, new attack combinations, or unfamiliar command-and-control behavior.

The Explainable AI component improves the practical usefulness of the framework by identifying the events, features, entities, and relationships that contribute to a malware alert. Rather than producing only a malicious or benign prediction, the system can explain whether an alert was influenced by abnormal shell execution, secret access, unusual Kubernetes API activity, privilege misuse, suspicious network



communication, or abnormal resource consumption. This supports faster alert triage, more informed incident investigation, and improved analyst trust.

Overall, the study concludes that hybrid Transformer-Graph Networks with explainability can strengthen cloud-native security by providing accurate, context-aware, and interpretable zero-day malware detection. Future work should validate the framework using larger real-world cloud datasets, diverse Kubernetes environments, federated learning approaches, adversarial robustness testing, and continuous model adaptation to emerging cloud threats.

REFERENCES

- [1] S. N. Qaisarani et al., "X-THREAT framework for adaptive and explainable deep learning-based minority-class and zero-day cyber threat detection," *Scientific Reports*, vol. 16, Art. no. 28721, 2026, doi: 10.1038/s41598-026-63529-5.
- [2] E. Amer, S. El-Sappagh, T. Abuhamad, B. A. S. Al-Rimy, and A. Mohasseb, "GraphShield: Advanced dynamic graph-based malware detection using graph neural networks," *Expert Systems with Applications*, vol. 298, Art. no. 129812, 2026, doi: 10.1016/j.eswa.2025.129812.
- [3] M. Gaber, M. Ahmed, and H. Janicke, "Zero day malware detection with Alpha: Fast DBI with Transformer models for real world application," *Computers & Electrical Engineering*, vol. 128, Art. no. 110751, 2025, doi: 10.1016/j.compeleceng.2025.110751.
- [4] H. Wu et al., "A graph neural network framework for dynamic malware detection using API calls and lightweight containers," in *Proc. 5th Int. Conf. Intelligent Communications and Computing (ICICC)*, 2025, pp. 294-301, doi: 10.1109/ICICC66840.2025.11199642.
- [5] D. Natsos and A. L. Symeonidis, "Transformer-based malware detection using process resource utilization metrics," *Results in Engineering*, vol. 25, Art. no. 104250, 2025, doi: 10.1016/j.rineng.2025.104250.
- [6] S. I. Berrios Vasquez, P. A. Hermosilla Monckton, D. I. Leiva Muñoz, and H. Allende, "Zero-day threat mitigation via deep learning in cloud environments," *Applied Sciences*, vol. 15, no. 14, Art. no. 7885, 2025, doi: 10.3390/app15147885.
- [7] Zhen, Y., Tian, D., Fu, X., & Hu, C., "A novel malware detection method based on audit logs and graph neural network," *Engineering Applications of Artificial Intelligence*, vol. 152, Art. no. 110524, 2025, doi: 10.1016/j.engappai.2025.110524.
- [8] H. Mohammadian, G. Higgins, S. Ansong, R. Razavi-Far, and A. A. Ghorbani, "Explainable malware detection through integrated graph reduction and learning techniques," *Big Data Research*, vol. 41, Art. no. 100555, 2025, doi: 10.1016/j.bdr.2025.100555.
- [9] H. Shokouhinejad, G. Higgins, R. Razavi-Far, H. Mohammadian, and A. A. Ghorbani, "On the consistency of GNN explanations for malware detection," *Information Sciences*, vol. 721, Art. no. 122603, 2025, doi: 10.1016/j.ins.2025.122603.
- [10] H. Shokouhinejad, R. Razavi-Far, G. Higgins, and A. A. Ghorbani, "Enhancing GNN explanations for malware detection with dual subgraph matching," *Machine Learning and Knowledge Extraction*, vol. 8, no. 1, Art. no. 2, 2026, doi: 10.3390/make8010002.
- [11] A. Aly, A. M. Hamad, M. Al-Qutt, et al., "Real-time multi-class threat detection and adaptive deception in Kubernetes environments," *Scientific Reports*, vol. 15, Art. no. 8924, 2025, doi: 10.1038/s41598-025-91606-8.
- [12] M. Osswald, T. Schöenberger, G. Cantali, W. Soussi, and G. Gür, "Anomaly detection in microservices architecture using graph neural networks," in *Proc. 33rd Euromicro Int. Conf.*, 2025, doi: 10.1109/PDP66500.2025.00085.
- [13] S. Kwon, W. Son, and J. H. Lee, "Anomaly detection in containerized tactical systems using temporal graph neural networks," in *Proc. IEEE Military Communications Conf. (MILCOM)*, 2025, pp. 1566-1571, doi: 10.1109/MILCOM64451.2025.11310523.
- [14] A. Nousias et al., "Malware detection in Docker containers: An image is worth a thousand logs," in *Proc. IEEE Int. Conf. Communications (ICC)*, 2025, doi: 10.1109/ICC52391.2025.11161263.



- [15] J. Ryu, R. Kim, S. Lee, S. Kim, H. Choi, and S. Kim, "Hybrid runtime detection of malicious containers using eBPF," *Computers, Materials & Continua*, vol. 86, no. 3, Art. no. 13, 2026, doi: 10.32604/cmc.2025.074871.
- [16] V. Govindarajan and J. H. Muzamal, "Advanced cloud intrusion detection framework using graph based features transformers and contrastive learning," *Scientific Reports*, vol. 15, Art. no. 20511, 2025, doi: 10.1038/s41598-025-07956-w.
- [17] R. Xie and D. Liu, "A novel hybrid graph neural network and transformer model for intrusion detection," *Peer-to-Peer Networking and Applications*, vol. 19, Art. no. 59, 2026, doi: 10.1007/s12083-025-02171-w.
- [18] M. C. Ipek and S. Sen, "Explainable Android malware detection and malicious code localization using graph attention," *Journal of Information Security and Applications*, vol. 98, Art. no. 104385, 2026, doi: 10.1016/j.jisa.2026.104385.
- [19] G. Andresini, A. Appice, V. Belvedere, G. Fiameni, and D. Malerba, "Anakin: Explainable Android malware detection with graph neural networks," *Cybersecurity*, vol. 9, Art. no. 116, 2026, doi: 10.1186/s42400-026-00552-z.
- [20] F. Ares-Robledo, H. Rifà-Pous, and R. Clariso, "Graph neural networks for anomaly detection: A systematic review of dynamic temporal approaches," *Artificial Intelligence Review*, vol. 59, Art. no. 129, 2026, doi: 10.1007/s10462-026-11532-7.
- [21] Imtiaz, U. (2026). Dynamic Security Certification Framework for Evolving Distributed Architectures. In: Shukla, S., Sayama, H., Tiwari, K., George, J.P., Kureethara, J.V. (eds) Data Science and Security. IDSCS 2025. Lecture Notes in Networks and Systems, vol 1947. Springer, Cham. https://doi.org/10.1007/978-3-032-24360-7_32
- [22] Imtiaz, U. (2026). Ghost Signals: Ethical RF Adversarial Testing of Consumer Alarm Ecosystems. In: Shukla, S., Sayama, H., Tiwari, K., George, J.P., Kureethara, J.V. (eds) Data Science and Security. IDSCS 2025. Lecture Notes in Networks and Systems, vol 1945. Springer, Cham. https://doi.org/10.1007/978-3-032-24075-0_21
- [23] M. I. K. Javed, M. Imran, A. A. Khan, M. Mehedi, A. Islam, and R. Pervez, "Explainable machine learning framework for early heart disease detection using SMOTE and SHAP," *Vascular and Endovascular Review*, vol. 9, no. 1, pp. 316-324, 2026.
- [24] M. I. K. Javed, M. R. M. Sirazy, S. Mandal, S. A. Akter, A. Hassan, and H. Esa, "Developing AI-based financial forecasting and cybersecurity systems for the US digital economy," *Frontiers in Computer Science and Artificial Intelligence*, vol. 5, no. 5, pp. 30-38, 2026.
- [25] M. I. K. Javed, M. P. Ahmed, F. M. Tofa, M. F. Islam, C. A. Gomes, and R. M. Sirazy, "Federated intrusion detection for Internet of Medical Things networks: Differential privacy, non-IID robustness, and cross-device generalization," *Journal of Computer Science and Technology Studies*, vol. 8, no. 8, pp. 303-315, 2026.
- [26] P. S. Ponduru, "Decision intelligence for AI and emerging technologies: The AEGIS-DM framework for trustworthy, cost-aware, and low-latency decision making," 2024.
- [27] P. S. Ponduru, P. P. V. Nandanavanam, and S. K. K. Ponduru, "SAFE-HealCloud: Safety-aware, agentic self-healing for cloud infrastructure," *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, vol. 12, no. 4, pp. 163-188, 2026.
- [28] C. A. Gomes, M. I. K. Javed, T. Ahammad, S. Mandal, S. Hassan, M. K. Sejan, P. S. Ponduru, M. M. Bhuiyan, and M. L. Jamali, "Operational and financial evidence for generative AI in United States healthcare administrative workflows," *Frontiers in Computer Science and Artificial Intelligence*, vol. 4, no. 4, pp. 138-145, 2025.
- [29] M. R. Hasan, M. A. Rahman, C. A. H. Gomes, F. N. Nitu, C. A. Gomes, M. R. Islam, and R. E. R. Shawon, "Building robust AI and machine learning models for supplier risk management: A data-driven strategy for enhancing supply chain resilience in the USA," *Advances in Consumer Research*, vol. 2, no. 4, pp. 1152-1171, 2025.



-
- [30] M. D. Rasheed, W. Khan, M. Imran, N. Ahmad, Y. Khan, M. Akram, and M. Sultana, "Leveraging artificial intelligence for advance data networking and cybersecurity," *Spectrum of Engineering Sciences*, vol. 4, no. 3, pp. 286-298, 2026.
- [31] Fahad Amin (2025). A Scalable Framework for Interpretable Binary Vulnerability Analysis Using Data Dependency Modeling. In *Proceedings of the International Conference on Artificial Intelligence and Cybersecurity (ICAIC 2025)*, 249-255.
- [32] F. Amin, "Binary Flaw Detection: A Security Analysis Paper," in *2025 International Conference on Advances in Machine Intelligence, and Cybersecurity Technologies (AMICT)*, Kota Kinabalu, Sabah, Malaysia, 2025, pp. 325-330, doi: 10.1109/AMICT65811.2025.11402666.

