ARES: Anomaly Recognition Model For Edge Streams

Simone Mungari
University of Calabria
ICAR-CNR
Revelis s.r.l.
Rende, Italy
simone.mungari@unical.it

Albert Bifet
Artificial Intelligence Institute, University of Waikato
LTCI, Télécom Paris, Institut Polytechnique de Paris
Hamilton, New Zealand
abifet@waikato.ac.nz

Giuseppe Manco
ICAR-CNR
Rende, Italy
giuseppe.manco@icar.cnr.it

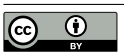
Bernhard Pfahringer
Artificial Intelligence Institute, University of Waikato, NZ
Hamilton, New Zealand
bernhard@waikato.ac.nz

Abstract

Many real-world scenarios involving streaming information can be represented as temporal graphs, where data flows through dynamic changes in edges over time. Anomaly detection in this context has the objective of identifying unusual temporal connections within the graph structure. Detecting edge anomalies in real time is crucial for mitigating potential risks. Unlike traditional anomaly detection, this task is particularly challenging due to concept drifts, large data volumes, and the need for real-time response. To face these challenges, we introduce ARES, an unsupervised anomaly detection framework for edge streams. ARES combines Graph Neural Networks (GNNs) for feature extraction with Half-Space Trees (HST) for anomaly scoring. GNNs capture both spike and burst anomalous behaviors within streams by embedding node and edge properties in a latent space, while HST partitions this space to isolate anomalies efficiently. ARES operates in an unsupervised way without the need for prior data labeling. To further validate its detection capabilities, we additionally incorporate a simple yet effective supervised thresholding mechanism. This approach leverages statistical dispersion among anomaly scores to determine the optimal threshold using a minimal set of labeled data, ensuring adaptability across different domains. We validate ARES through extensive evaluations across several real-world cyber-attack scenarios, comparing its performance against existing methods while analyzing its space and time complexity. The code used to perform the experiments is publicly available at https://github.com/AnomalyRecognitionModelForEdgeStreams/ARES.

CCS Concepts

• **Computing methodologies** → **Anomaly detection**; *Learning latent representations*; Classification and regression trees; • **Information systems** → **Data streaming**.



KDD '26, Jeju Island, Republic of Korea

ACM ISBN 979-8-4007-2258-5/2026/08
https://doi.org/10.1145/3770854.3780242

Keywords

Anomaly Detection, Edge Stream, Dynamic Graphs, Machine Learning, Cyber-Security

ACM Reference Format:
Simone Mungari, Albert Bifet, Giuseppe Manco, and Bernhard Pfahringer. 2026. ARES: Anomaly Recognition Model For Edge Streams. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.1 (KDD '26), August 09–13, 2026, Jeju Island, Republic of Korea.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3770854.3780242

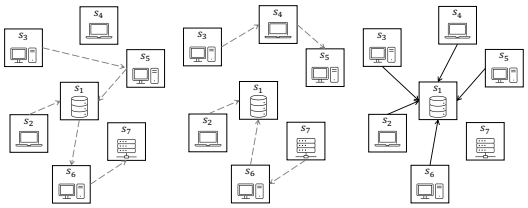
1 Introduction

In the context of cyber-security, anomaly detection is one of the most important tasks, whose aim is to identify potential cyber-attacks [29, 45] such as SQL Injections, Botnets, Distributed Denial of Service (DDoS) attacks [1, 32], as well as financial frauds [2]. Most of these threats can be represented as temporal connections, i.e., temporal edges, between endpoints. Automatically recognizing anomalous edges in data streaming domains is hence fundamental for the early detection of noxious activities.

Edge anomalies can be generally categorized into "burst" or "spike" (which can be further divided in more specific sub-categories [10]), based on whether they involve multiple nodes and connections or represent an anomaly relative to a specific connection. A noteworthy example of burst anomaly is the classical DDoS scenario, described in Figure 1, where a peer (s_1, representing a hosting website) is targeted by other nodes in a network, who massively connect to the peer within a specific timeframe. Early recognition of such abnormal connections is crucial for avoiding potential cyber-attacks and ensuring a reliable quality of service.

Discovering potential harmful activity within the flow of such temporal interactions poses several challenges. First, the streaming nature of the underlying data requires the capability of dealing with changes in the underlying edge distributions (described as concept drifts in the literature) or the rise of new anomalies. Additionally, data streaming scenarios are inherently hard to cope with due to the massive amount of data to manage, and by contrast they require the capability of classifying edges in real-time (or near real-time).

The recent literature has paid significant attention to the challenge of identifying anomalies in edge streams. Supervised approaches, such as those in [50, 53], face two main limitations. First,



(a) T1 (b) T2 (c) T3

Figure 1: Connections among endpoints over time (T1, T2, T3). Fig (a) and (b) depict the typical behavior of connections among endpoints. Fig (c) shows a significant increase of connections from $s_2, ..., s_6$ to s_1.

they are restricted to detecting previously known anomalies. Second, they rely on labeled data, which can be difficult to obtain in real-world operational scenarios. In contrast, semi-supervised and unsupervised algorithms [5–8, 8, 10, 14, 24, 26, 33, 46, 51] address these limitations by reducing the need for labeled examples. The approaches in [14, 33] focus on spike anomalies and tend to overlook anomalies originated by coordinated behavior involving multiple edges. On the other hand, other methods focus on burstiness [5, 7, 8, 10] and disregard anomalies involving sparsely-connected graph components. Another significant challenge with most unsupervised methods is that they do not provide a clear distinction between normal and anomalous edges. Instead, they generate a range of anomaly scores that must be compared to domain-specific thresholds to assess how abnormal the data is. The choice of these thresholds is critical, as even small adjustments can have a major impact on detection performance.

In this paper, we introduce ARES (Anomaly Recognition Model for Edge Streams), an unsupervised anomaly detection model for edge streams. First, ARES automatically extracts and maps node and edge features from the stream to a latent space. To do so, the model employs a Graph Neural Network (GNN), which is the state-of-the-art approach for graph representation learning tasks. Second, for assigning an anomaly score for each edge, we endow ARES with a tree-based model named Half-Space Trees (HST). As such, ARES randomly divides the latent space and separates normal from suspicious edges, according to their representations. The degree of anomaly in a component is revealed through the geometrical embeddings, enabling the representation of both spiking and bursty behaviors in terms of geometrical proximity. HST further enhances the model by efficiently exploring the latent space and isolating anomalies through random partitioning, thus enabling the detection of a broad range of anomaly types. Our framework integrates a simple GNN to minimize computational overhead while delegating stream processing to the HST. This design ensures efficient anomaly detection by balancing the strengths of both components—leveraging the GNN for feature extraction and the HST for handling dynamic edge streams. More importantly, we provide both theoretical and empirical evidence that ARES offers an efficient processing of the data stream, as it complexity is only affected by the computational complexity of the GNN component.

As an additional aspect that we investigate, we further propose a simple yet effective supervised thresholding mechanism that can be combined with the ARES scoring framework. Specifically, we propose a domain-agnostic method that leverages the Gini index—a measure of statistical dispersion among anomaly scores—to strongly separate true anomalies effectively. Although the method requires a minimal amount of supervision, it intrinsically explores the distribution of the anomaly scores against true labels, thus identifying the optimal cut-off in an effective way.

We empirically demonstrate the capabilities of ARES by performing an extensive evaluation on seven different real-world scenarios, which represent different types of cyber-attacks. Our analysis also includes a comparison with existing methods and considers theoretical aspects such as space and time complexities. To summarize, our contributions are the following.

- We devise ARES, a simple yet effective model for identifying anomalies in edge streaming data that combines Graph Neural Networks and Half-Space Trees.
- We show both theoretically and empirically that ARES is amenable for efficient edge stream processing and can effectively generate meaningful node and edge embeddings that consistently uncover edge anomalies.
- We additionally equip ARES with an algorithmic validation-based supervised method for choosing the best threshold, which can be used in real-world scenarios for discriminating between normal and abnormal activities.
- Through an extensive evaluation, we show that ARES outperforms SOTA methods using seven different real-world scenarios.

The rest of the paper is organized as follows. Section 2 discusses the contributions of the current literature in the area of graph anomaly detection. Section 3 formalizes the problem and provides a technical description of the proposed solution. Theoretical insights and model properties are discussed in Section 4. The experimental evaluation is described in Section 5. Finally, Section 6 concludes the paper and set pointers for further research.

2 Related work

Our contribution is placed in the general area of Anomaly Detection. Our focus is on anomalies occuring within graph representations and their evolution in a streaming scenario. In this perspective, we identify two main topics related to our work, namely graph/node anomalies and edge anomalies.

Graph and Node Anomaly Detection. The problem of detecting anomalous nodes, or even anomalous graph configuration, has been extensively studied in the current literature, given the importance of accounting for inter-dependencies during the anomaly detection process, which can be also exploited for describing and explaining the inherent anomalies. Approaches to graph anomaly detection focus on both static and dynamic aspects [34], and consider structure, local and global features [3]. Graph iForest [49] adapts the Isolation Forest algorithm for identifying anomalous graphs in static scenarios. Isolation Forest (IF) [26] is a popular tree-based method for anomaly detection on tabular data. It isolates anomalies by dividing the feature space into sub-spaces. Particular emphasis is also given by explainability and interpretability [27].

Recent studies employ advanced techniques based on Neural Networks for discovering anomalies. Deep representation learning techniques provide expressive representations facilitating the task of dividing anomalies from normal data [28], at the expense of interpretability. [46] combines Isolation Forests with Neural Networks (NNs). In particular, the author propose to initialize the weights of an ensemble of NNs, exploiting Isolation Forest to identify anomalous data points based on the embeddings produced by the former. Graph Neural Networks [44] have recently gained particular attention [11, 21, 28, 42]. The authors of [50] present a framework for detecting malwares, exploiting both Graph Neural Networks and Recurrent Neural Networks in order to capture temporal and structural correlations. [53] focuses on the detection the anomalous nodes which are the source of potential harmful connections within a network.

Edge Anomaly Detection. The detection of anomalous edges in both static and dynamic scenarios seeks to uncover unexpected connections at different levels: locally, where a single edge between two endpoints is deemed anomalous, and globally, where multiple connections within a group of nodes raise suspicion. [48] introduced NetWalk, which detects node and edge anomalies in temporal networks by clustering their corresponding embeddings. The anomaly score for a data point (edge or node) is determined by its proximity to the nearest cluster center. AddGraph [51] combines Graph Convolutional Network (GCN) with attention methods and employs a semi-supervised training procedure to cope with the scarcity of labelled anomalies.

In the edge streaming context, PENminer [5] is a framework for studying persistent patterns in evolving networks by extending the concept of temporal motifs, i.e., recurrent patterns in temporal edges. F-Fade [10] models the distribution of edge frequency over time and identifies anomalies by assessing the likelihood of the observed frequency of incoming interactions. In a similar vein, SedanSpot [14] introduces a PageRank-like algorithm that exploits the Marginal Proximity Increase metric to compute the anomaly score, which is based on edge frequencies. SLADE-H [24] extend the Temporal Graph Networks architecture proposed by [36] for detecting anomalous edges, employing node memories for tracking the graph evolution along the stream. The proposed model assigns anomaly scores by measuring the cosine similarity between the node's current and previous memory vectors. MIDAS [6, 7] exploits count-min sketches (CMS, specific data structures tailored for counting node degrees) to identify anomalies. AnoGraph [8] employs an enhanced version of CMS (high-order count-min sketches, H-CMS). These approaches are particularly effective in detecting global (bursting) features and instead may struggle in scenarios where anomalies are not related to edge frequencies.

Compared to the aforementioned approaches, our approach, similarly to [24], relies on the capability of GNNs to automatically extract topological and semantic features of the underlying edge stream. This allows us to boost the detection both at a local and a global level, as explained in the next sections.

3 Anomaly Recognition for Edge Streams

We begin our treatment by fixing some notation and setting the main problem statement. Let $\mathcal{G} = \{e_1, e_2, ..., \}$ be a (potentially infinite) stream of edges representing connections between entities. We denote as V_t and E_t the set of all the nodes and edges comprised within the stream until time t, respectively. Each e_t is a tuple in the form $e_t = (s, d, t)$ which occurs at time t, and where $s, d \in V_t$ are the source and the destination node, respectively. Let $X_t \in \mathbb{R}^{|V_t| \times K}$ be the feature matrix associated with nodes, and $G_t = \langle V_t, E_t, X_t \rangle$ be the temporal directed multigraph representing the evolution within $\mathcal{G}$ up to time t. We assume that nodes exhibit at least one feature. Scenarios where no explicit features are available are modeled by considering the node identifiers as the only available feature. This modeling choice is crucial for devising meaningful representations of structural properties, as explained in the following section.

Each edge e_t can be characterized by a binary label $l^{e_t} \in \{0, 1\}$. In particular, $l^{e_t} = 1$ denotes that e_t can be considered anomalous, whereas $l^{e_t} = 0$ denotes normality.

Our objective is to estimate the probability $P(l^{e_t} = 1 | e_t, G_t)$, given the edge $e_t \in \mathcal{G}$ and the temporal graph $G_t = \langle V_t, E_t, X_t \rangle$ which encodes the evolution of $\mathcal{G}$. In practice, the research question we are interested in addressing is: how likely is it to consider $e_t \in \mathcal{G}$ anomalous, given its features and the current history G_t?

To tackle this problem, we leverage GNN embeddings and Half-Space Trees. Figure 2 represents a graphical overview of the framework. The core idea is to utilize a Half-Space Tree model to distinguish anomalous edges from normal ones in $\mathcal{G}$. By assuming that elements to be analyzed can be mapped into an appropriate representation space, Half-Space Trees provide an efficient partitioning of such space, thus enabling a dynamic yet fast and accurate identification of isolated data points. To ensure a semantically consistent representation of the evolving graph components, we use a Graph Neural Network to generate embeddings for both nodes and edges. Thus, the two main components of our framework are:

(1) *Embedding generator*, which maps edges and nodes into a latent space.
(2) *Anomaly scorer*, which computes anomaly scores for edges $e_t = (s, d, t)$ by leveraging the latent representation of the edge (e_t), along with the embeddings of the source (s) and destination (d) nodes.

3.1 Embedding generator

To map edges and nodes to a latent space by generating meaningful embeddings that translate structural properties into spatial features, we utilize Graph Neural Networks, which have been proven effective for learning tasks involving graphs [44]. In particular, we devise a Graph Autoencoder (GAE) [22, 35, 43, 52] architecture for learning the latent representations of nodes. A graph autoencoder is an encoder-decoder model $\tilde{G} = D(E(G))$, where G is a graph and $E(G) \in \mathbb{R}^{n \times d}$ is the encoding of all nodes $V \in G$. It compresses the original data into latent vectors ($E(G)$) and reconstructs the original data from these vectors ($D(E(G))$), such that $G \approx \tilde{G}$. Both the encoder and decoder are GNNs. We denote them as $E \equiv \text{GNN}^{enc}$ and $D \equiv \text{GNN}^{dec}$, respectively. For our purposes, we adopt GraphSAGE [20] to define the architecture of the GNN components.

After training the GAE, we can generate node embeddings using only the encoder component GNN^{enc}. Specifically, we compute $h_t^v \in \mathbb{R}^d$, the embedding of the node v at time t, and $h_t^e \in \mathbb{R}^d$,

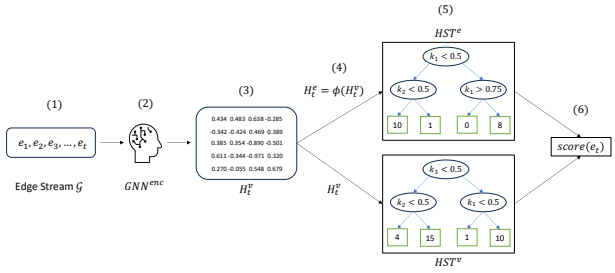


Figure 2: Framework overview. Given a streaming of edges denoted as $\mathcal{G}$ (1), we use a pre-trained Graph Neural Network, GNN^{enc} (2), to derive a set of node embeddings, represented as H_t^v (3). We also generate a set of edge embeddings denoted as H_t^e (4). These embeddings serve as inputs to two Half-Space Trees, designated as HST^v and HST^e, which encode unexpectedness for both nodes and edges according to their positioning in the latent space (5). The final anomaly score (6) is hence computed by combining information from these trees both at a node and an edge level.

which represents the embedding of the edge e. The embedding h_t^v is obtained directly from GNN^{enc}. The edge embedding h_t^e is obtained by combining the embeddings of the source and destination nodes. For this, we adopt two alternative strategies, inspired by prior work [18]. Given an edge $e_t = (s, d, t)$ at time t, the first alternative consists in averaging the embeddings of both the source and destination nodes,

$$h_t^{e_t} = \phi_1(h_t^s, h_t^d) = \frac{h_t^s + h_t^d}{2} \tag{1}$$

Alternatively, we can compute the difference between the node embeddings, capturing directional information through the asymmetry of this difference:

$$h_t^{e_t} = \phi_2(h_t^s, h_t^d) = h_t^d - h_t^s \tag{2}$$

3.2 Anomaly Scoring

To efficiently explore the resulting latent space, we adopt Half-Space Trees (HST)[41], a robust tree ensemble framework designed for anomaly detection in streaming environments. HST belongs to the broader family of isolation-based anomaly detection methods. We chose HST for anomaly scoring due to its demonstrated effectiveness and reliability in streaming data scenarios. Like Isolation Forest[26], HST identifies anomalies by recursively partitioning the data. However, unlike Isolation Forest, HST is specifically tailored for real-time applications. One of its key strengths is the use of a sliding window mechanism, which enables the model to adapt dynamically to concept drift and evolving data distributions. Another advantage is that HST does not require a dedicated training phase, in contrast to many other isolation-based techniques [37], which helps reduce computational overhead. Overall, isolation-based methods like HST offer several benefits: (i) low computational and memory requirements, (ii) scalability to large datasets, and (iii) the ability to detect both global and local anomalies [37].

Technically, HST builds each tree iteratively by randomly choosing a feature and a splitting value to partition the space. Each leaf node in the tree keeps a count of the number of data instances that fall within its corresponding region. Data points that appear in sparsely populated subregions are then considered anomalous. More precisely, for a generic input X, HST computes the anomaly score by combining the anomaly scores exhibited by each tree in the ensemble. Each score considers both the sparsity and the extention of the subregion where X occurs:

$$\text{HST}(X) = \frac{1}{Z}\text{Node}.r \times 2^{\text{Node}.h} \tag{3}$$

Here, Node.r is the counter and Node.h is the depth level of the leaf node where X is located (i.e., the number of inequality constraints that border the subspace). Z is a normalization constant. The counts within leaf nodes are updated dynamically as data streams in, using a window-based strategy to track changes over time.

Within ARES, we exploit two HST instances, namely HST^v and HST^e, to combine anomalies both at node and edge levels. The final score for a given edge $e_t = (s, d, t)$ is then obtained through a weighted average:

$$\begin{aligned}\text{score}(e_t) =& w_s\text{HST}^v(h_t^s) + w_d\text{HST}^v(h_t^d) \\ &+ w_e\text{HST}^e(h_t^{e_t})\end{aligned} \tag{4}$$

The (normalized) weights w_1, w_2, and w_3 reflect the contributions of the source, destination node and edge embeddings to the overall anomaly score. Within our framework, score(e_t) encodes the probability $P(l^{e_t} = 1 | e_t, G_t)$.

3.3 Processing Edge Streams

For a given stream $\mathcal{G}$, the framework starts by training the GAE on an initial snapshot G_t of the edge stream. Next, we freeze the neural network's weights and exploit GNN^{enc} for the embeddings throughout $\mathcal{G}$. For this, we build and initialize the internal data structures of both HST^v and HST^e, using a subset of the data used for training the GAE.

Algorithm 1 shows the main pipeline of the framework. The algorithm receives as input the pre-trained GNN^{enc}, initialized HSTs, and the data stream $\mathcal{G}$. The main flow hence consists in incrementally considering new edges $e_t \in \mathcal{G}$ (lines 5-18). For each (batch of) new edges, the current graph G_t selected, (line 6) and the GNN^{enc} component generates the embeddings for the source and destination nodes ($s, d \in e_t$) (line 10). These embeddings are hence combined (line 11) to generate the embedding $h_t^{e_t}$ (through either Equation 1 or Equation 2). The anomaly score $score^{e_t}$ is hence computed (line 12).

An important consideration concerning the embedding is that restricting the training of GNN^{enc} on only the initial snapshot G_t of the edge stream does not inherently limit the representational power of the GNN, a design choice also adopted in prior approaches [12]. This is because the underlying temporal graph G_t is continuously updated as new edges arrive. As the graph evolves, it dynamically integrates new structural and relational information, allowing node and edge embeddings to reflect the latest changes. Consequently, even without retraining, the model remains capable of capturing emerging patterns and dependencies within the graph.

To compute anomaly scores, we introduce two variants within our framework: ARES-Static and ARES-Dynamic. In ARES-Static, the internal structures of the HSTs remain fixed throughout the

data stream, meaning that the counters inside the leaf nodes are not updated after initialization. In contrast, ARES-Dynamic allows for continuous updates to the HSTs, enabling them to adapt to changes in the data stream.

As noted in [41], HSTs perform well when anomalies in the data stream are relatively sparse. However, their effectiveness diminishes when a large number of anomalies occur, as this can distort the counts within the tree ensemble. ARES-Static mitigates this issue by preventing continuous updates and instead relying on a fixed subsample of the data. Within the algorithm, line 13 specifies where an update may occur when the ARES-Dynamic variant is employed.

In the algorithm, we also incorporate a *cache* memory of predefined size C to expedite the computation of anomaly scores for edges $e_t = (s, d, t)$ that have previously appeared. Specifically, if there exists an earlier occurrence $e_{t'} = (s, d, t') \in \mathcal{G}$ with $t' < t$, the cache prevents redundant recalculations. This optimization is particularly beneficial in scenarios such as burst traffic or denial-of-service (DoS) attacks, where a peer repeatedly attempts to connect to the same device within a short timeframe. To manage stored entries efficiently, the cache can implement various update policies, including FIFO (First-In, First-Out), LRU (Least Recently Used), or full replacement.

Algorithm 1 Streaming pipeline

1: **Input:** pre-trained GNN^{enc}, HST^{e}, HST^{v}, $\mathcal{G}$, cache size C
2: **Output:** *Scores*
3: Cache $\leftarrow$ {}
4: *Scores* $\leftarrow$ []
5: **while** new edge $e_t = (s, d, t)$ occurs in $\mathcal{G}$ **do**
6: $G_t \leftarrow$ Update graph G_{t-1}
7: **if** $(s, d) \in$ Cache **then**
8: $\text{score}^{e_t} \leftarrow \text{Cache}[(s, d)]$
9: **else**
10: $h^s, h^d \leftarrow \text{GNN}^{enc}(G_t)$
11: $h^{e_t} \leftarrow \phi(h^s, h^d)$ {Equations 1/2}
12: $\text{score}^{e_t} \leftarrow score(e_t)$ {Equation 4}
13: Update HST^e, HST^v
14: $\text{Cache}[(s, d)] \leftarrow score^{e_t}$
15: **end if**
16: **if** |Cache| $> C$ **then**
17: Update *Cache*
18: **end if**
19: *Scores*.insert($score^{e_t}$)
20: **end while**

3.4 Validation-based Adaptive thresholding

Algorithm 1 enables the computation of the anomaly score for each edge in the stream, within a range from 0 (unlikely) to 1 (very likely). For the choice of the appropriate threshold τ, we devise a validation-based strategy that relies on a validation set $\{(e_{t_1}, l_1), \ldots, (e_{t_n}, l_n)\}$, where anomaly labels are known for each edge. For each of these edges we can compute the anomaly score by selecting the associated temporal graph and computing the scores provided by the GNN embeddings in the HSTs computed by algorithm 1. The result is hence a set of pairs $D = \{(\text{score}^{e_{t_1}}, l_1), ..., (\text{score}^{e_{t_n}}, l_n)\}$. The optimal threshold can be obtained by considering the partition that provides the minimal variance in the labels obtained by the two resulting subsets. For this, we exploit the Gini Diversity Index [9]. Let D_1, D_2 be two partitions of D, where $D_1 = \{(s, l) \in D | s \leq \tau\}$ and $D^2 = D - D_1$. We then compute the optimal threshold τ^* by minimizing the formula

$$\tau^* = \arg\min_{\tau} \frac{|D_1|}{|D|} \times \text{Gini}(D_1) + \frac{|D_2|}{|D|} \times \text{Gini}(D_2), \tag{5}$$

where $\text{Gini}(S) = 1 - (p_+^2 + p_-^2)$ and p_+ (resp. p_-) is the fraction of positive (resp. negative) labels in S.

4 Analysis

ARES relies on theoretical and empirical guarantees provided by its two main components, namely GNN and HST. First, as explained in Section 3, GNN^{enc} exploits a GraphSAGE model. The latter has been shown to provide meaningful representations of nodes and edges that capture the underlying graph structure [20]. This is also true when the feature matrix X_t is naively represented as a set of unique identifiers [20, Corollary 2]. As a consequence, equipping ARES with this architecture ensures that GNN^{enc} learns local graph structures and generates meaningful node and edge embeddings.

A second aspects is concerned with the ability of Half-Space Trees models to produce proper anomaly scores. As already mentioned above, HST assumes that anomalies are rare within the dataset, as empirically shown in [41]. Consequently, we expect ARES to work well in situations where the graph streams exhibits a limited amount of anomalies. Surprisingly, the framework produces high-quality results even when this property does not occur in the data, as we will show later in Section 5.

We next analyze the computational complexity of the proposed approach. For this, we observe that the two main components of the primary loop in Algorithm 1 (GNN encoding and HST exploitation and update) are executed sequentially. Let G_t contain n nodes, and assume GraphSAGE is configured with κ layers, a recursive neighborhood sampling of r neighbors, and a batch size of s. According to [44], the time and space complexity of the GraphSAGE model are $O(r^\kappa n K^2)$ and $O(sr^\kappa K + \kappa K^2)$, respectively. In the space complexity expression, the term $sr^\kappa K$ accounts for storing the embeddings of sampled neighbor nodes across all layers, while κK^2 represents the storage required for the model parameters.

Notably, within algorithm 1, GraphSAGE is used to embed only two individual nodes. The final embedding of a single node only needs to consider its r neighbors. Since each layer aggregates and concatenates information from neighboring nodes, these dependencies propagate backward through multiple layers. Consequently, with an optimized implementation, the time complexity for computing the embedding of a single node is $O(r^\kappa K^2)$, which directly corresponds to the computational cost of step 10 in the algorithm.

The complexity of the HST algorithm primarily depends on the maximum tree height h, the number of trees β, and the window size ψ. As discussed in [41], the time complexity is $O(\beta(h + \psi))$ and the space complexity is $O(\beta 2^h)$.

By combining these results, the overall complexity for each iteration of Algorithm 1 is $O(r^\kappa K^2 + \beta(h+\psi))$ in time and $O(sr^\kappa K + \kappa K^2 + \beta 2^h)$ in space. Assuming that h, ψ, β, κ, s, r, and K are bounded, the

overall complexity is determined solely by the number of nodes in the current graph G_t.

In practice, by assuming r and k bounded, the complexity of ARES is constant for each edge, offering a crucial feature in the edge streaming context. This property ensures that the model can efficiently handle large and continuously growing graphs without an exponential increase in computational cost. This theoretical advantage is further empirically validated through experimental results, as demonstrated in Section 5.

5 Experiments

We evaluate the capabilities of the proposed framework through a series of experiments aimed at answering the following research questions:

RQ1: Can ARES be used to recognize anomalies in edge streams? How does it compare to SOTA models?
RQ2: How effective is the adaptive thresholding method in discriminating between normal and anomalous edges?
RQ3: How does ARES behave when facing different categories of anomalies?
RQ4: How do the components of ARES affect its capability to detect anomalies? How robust is the framework?

We discuss the results for both variants of the proposed approach, namely ARES-Static and ARES-Dynamic. Our implementation is developed using both PyTorch Geometric [15] and River [30] frameworks. All experiments were performed on an NVIDIA DGX equipped with 4 V100 GPUs. The code used to perform the experiments is publicly available[1]. We provide further experiments and details in the Online Appendix [2].

5.1 Experimental Setting

Competitors. We compare our results with three state-of-the-art models, discussed in the following.

- MIDAS [7] exploits count-min sketches [13] and a chi-squared test to evaluate the degree of abnormality for each edge. We consider two further variants of MIDAS: MIDAS-R [6] that incorporates temporal and spatial relations, and MIDAS-F, which improves MIDAS-R by filtering anomalies to prevent negatively altering the internal structures.
- AnoGraph [8] expands the concept of count-min sketches to strengthen the ability of representing multidimensional inputs. In our experiments, we used the two variants focusing on edge anomalies: AnoEdge-G and AnoEdge-L.
- SLADE-H [24] employs a Temporal Graph Neural Network architecture for detecting anomalous edges. The model assigns anomaly scores by measuring the cosine similarity between the node's current and previous memory vectors, which are used to track the nodes' evolution.

To ensure a fair comparison, we used the source code provided by the authors and performed a tuning of the hyperparameters according to the findings described in the original papers.

[1]https://github.com/AnomalyRecognitionModelForEdgeStreams/ARES
[2]https://github.com/AnomalyRecognitionModelForEdgeStreams/ARES/blob/main/KDD_26_Ago_ARES_online_appendix.pdf

Metrics. We report two widely used metrics in anomaly detection: the Area Under the ROC Curve (ROC-AUC) and Average Precision (AP). The ROC-AUC measures the model's ability to distinguish between classes by computing the true positive rate (TPR) against the false positive rate (FPR) across different threshold values. Similarly, the AP summarizes the precision-recall curve by calculating precision and recall at various thresholds, providing insight into the model's performance in imbalanced scenarios.

In addition, to answer **RQ2** and evaluate the model capabilities under the thresholding mechanism, we report the F1-Score and Balanced Accuracy (B-Accuracy) metrics. Both metrics are calculated using the threshold determined by the adaptive thresholding methods described in section 3.4. For the competitors, we evaluate multiple threshold values and report the results for the threshold that yields the highest F1-Score on the validation set.

Datasets. Our evaluation is based on the following datasets:

- DARPA [25], one of the most widely used benchmarks, featuring various categories of cyber threats, including Denial-of-Service, Remote-to-Local, User-to-Root, and Surveillance attacks.
- ISCX2012[39] and CIC-IDS2017[38], provided by the University of New Brunswick (UNB), Canada. ISCX2012 includes network traffic for protocols such as HTTP, SMTP, SSH, IMAP, POP3, and FTP, while CIC-IDS2017 covers a range of attack types, including DoS, DDoS, Brute Force, XSS, SQL Injection, Infiltration, Port Scan, and Botnet.
- UNSW-NB15 [31], which consists of network traffic including both real and synthetic attack activities.
- CTU-13 [16], a botnet traffic dataset containing 13 distinct attack scenarios. We excluded scenarios with fewer than 1 million edges or that closely resembled other attack types. The final selected scenarios are 1, 10, and 13.

Table 8 (Appendix) summarizes the statistics of the seven real-world datasets. Each dataset is temporally split: 60% for training, 20% for validation, and 20% for testing. The training set is used to train the GAE and initialize HST^e and HST^v, and similarly for training SLADE-H and initializing MIDAS/AnoGraph structures.

We use the anomaly scores on the validation set for tuning the hyper-parameters. In ARES, the edge representation (Eq.1 vs. Eq.2) is also selected via validation. Final results are computed on the test set using Algorithm 1.

5.2 Results

Recognizing anomalies. To answer **RQ1** and compare the results of ARES against the competitors, we repeat the experiments using nine different seeds. The resulting metrics are compared using the Wilcoxon test with a confidence level of 95%. Table 1 report the comparisons. Bold and underlined values denote best and second-best, respectively. Statistical ties are represented by multiple bold/underlined values.

ARES demonstrates prominent results in terms of ROC-AUC and AP, exhibiting its capabilities in separating anomalies from normal edges. The only exception is on CIC-IDS2017 where MIDAS-F shows better scores both in terms of AUC and AP. The row relative to "Performance Gain" quantifies the relative improvement of ARES by displaying the percentage difference between its best results and those achieved by competing methods, offering a direct comparison

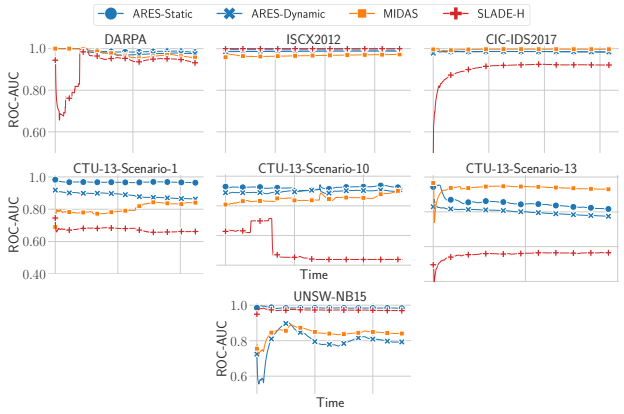


Figure 3: Comparisons of ROC-AUC scores over time among ARES-Static, ARES-Dynamic, MIDAS, and SLADE-H. The best variant of MIDAS is take into account for each dataset, as reported in Table 1

of performance improvement. In general, ARES consistently outperforms all the competitors, achieving improvements up to +12.8% on Scenario 1 and +23.5% on Scenario 10 in terms of ROC-AUC and AP, respectively.

Robustness to concept drifts. To further validate ARES, we assess its performance over time using the AUC-ROC metric, as illustrated in Figure 3. This evaluation highlights the model's robustness in the presence of concept drift—an inherent challenge in streaming scenarios. The results demonstrate that ARES consistently maintains stable performance, outperforming both MIDAS and SLADE-H. We omit results for AnoEdge, as its performance is significantly lower in comparison.

The robustness of ARES stems from the synergy of ARES 's core components. As previously discussed, the graph G_t is continuously updated, enabling the GNN to generate node and edge embeddings that adapt to the evolving data stream. Simultaneously, HST incrementally updates its internal structures with incoming edges, ensuring that anomaly scores are reflective of the most recent data.

Discerning between normal and abnormal. The behavior of the model under the the adaptive thresholding method (and hence the answer to **RQ2**) can be observed through the values of F1, B-Accuracy and Precision/Recall reported in Table 2. We observe robust results on both the static and dynamic versions of the model. ARES-Dynamic exhibits minor capabilities in terms of recall, but overall both methods are stable in detecting anomalies. Notably, the proposed method provides a simple yet effective alternative to baseline approaches, which lack an algorithmic mechanism for computing thresholds.

Detecting different cyber-attacks. To investigate RQ3, we evaluate the ability of ARES in detecting anomalies relative to specific attack scenarios. First, notice that the CTU-13 dataset variants already contain single different scenarios, referring to Click-fraud, UDP DDoS, and Port-scan attacks respectively. And in fact, as shown in Table 1, ARES outperforms its competitors in detecting such attacks.

We further conduct an in-depth analysis of accuracy scores for specific attack types in the DARPA and UNSW-NB15 (Table 3) datasets, both of which encompass diverse anomaly patterns, including bursty and spike anomalies. In both cases, ARES exhibits an accuracy remains consistent across all available attack types.

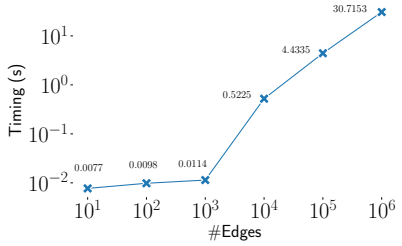


Figure 4: ARES-Static timings on CIC-IDS2017 dataset as the number of edges increases

Timings. Table 4 presents the running times for all datasets. Compared to rule-based models, ARES has higher computational costs due to the integration of the GNN and HST architectures. However, unlike SLADE-H, the proposed model leverages a lightweight yet powerful GNN, leading to significant computational speed-ups. This efficiency gain makes ARES a well-balanced solution, striking an optimal tradeoff between accuracy and performance. The tradeoff in computational cost is justified by the significant accuracy improvements ARES offers across diverse datasets. The model consistently outperforms MIDAS and AnoEdge in 6 out of 7 datasets in both AUC and AP metrics, highlighting its superior capability to detect subtle and complex anomalies in evolving graph structures. In applications where detection performance is critical, the additional computational overhead becomes a justified and worthwhile investment. Figure 4 presents a scalability analysis for increasing edge stream sizes. The running times exhibit linear growth as the number of edges increases. Notably, an overhead emerges when the number of edges exceeds 10^3, due to the GNN^{enc} component's need to process more batches and large adjacency matrices. Notwithstanding, the empirical result confirms the theoretical assessment discussed in Section 4.

Ablation study and Statistical robustness. To answer **RQ4**, we conduct an ablation study on the two core components of ARES: Half-Space Trees (HST) and GraphSAGE embeddings. First, we evaluate the contribution of HST by comparing it with RRCF [19] and a baseline GNN Encoder-Decoder (GNN-RecErr), which uses the reconstruction error as the anomaly score after previous work [28]. As shown in Table 5, HST provides significantly better accuracy. Notably, it also outperforms RRCF in inference speed (see Table 3 in the Online Appendix). Second, we assess the impact of the embedding model by testing different GNNs with HST. GraphSAGE consistently achieves the best performance, outperforming both GCN and GAT (Table 5). These results confirm the importance of high-quality graph embeddings and demonstrate the strong synergy between GraphSAGE and HST within ARES. Due to space limitations, we

Model	DARPA		UNSW-NB15		ISCX2012		CIC-IDS2017		CTU-13					
									Scenario 1		Scenario 10		Scenario 13	
	ROC-AUC	AP	ROC-AUC	AP	ROC-AUC	AP	ROC-AUC	AP	ROC-AUC	AP	ROC-AUC	AP	ROC-AUC	AP
MIDAS-F	0.957 ± 0.004	0.977 ± 0.002	0.793 ± 0.001	0.337 ± 0.001	0.972 ± 0.002	0.275 ± 0.015	**0.997 ± 0.000**	**0.968 ± 0.012**	0.841 ± 0.041	0.077 ± 0.015	0.911 ± 0.022	0.477 ± 0.074	**0.929 ± 0.051**	0.131 ± 0.049
MIDAS-R	0.840 ± 0.003	0.921 ± 0.001	0.840 ± 0.002	0.469 ± 0.001	0.771 ± 0.021	0.067 ± 0.006	0.791 ± 0.006	0.130 ± 0.004	0.618 ± 0.061	0.031 ± 0.007	0.294 ± 0.000	0.085 ± 0.000	0.342 ± 0.010	0.012 ± 0.001
MIDAS	0.781 ± 0.006	0.889 ± 0.003	0.815 ± 0.005	0.444 ± 0.002	0.597 ± 0.001	0.033 ± 0.001	0.544 ± 0.027	0.071 ± 0.005	0.523 ± 0.007	0.017 ± 0.001	0.305 ± 0.002	0.087 ± 0.002	0.389 ± 0.001	0.013 ± 0.000
AnoEdge-G	0.895 ± 0.009	0.948 ± 0.004	0.756 ± 0.012	0.423 ± 0.013	0.867 ± 0.014	0.097 ± 0.004	0.971 ± 0.010	0.586 ± 0.088	0.436 ± 0.006	0.015 ± 0.001	0.661 ± 0.000	0.153 ± 0.001	0.294 ± 0.011	0.011 ± 0.001
AnoEdge-L	0.909 ± 0.006	0.953 ± 0.003	0.752 ± 0.019	0.437 ± 0.031	0.893 ± 0.025	0.113 ± 0.026	0.980 ± 0.005	0.627 ± 0.066	0.484 ± 0.017	0.041 ± 0.003	0.696 ± 0.016	0.180 ± 0.017	0.315 ± 0.021	0.013 ± 0.001
SLADE-H	0.931 ± 0.042	0.958 ± 0.031	0.971 ± 0.021	0.867 ± 0.096	**0.999 ± 0.001**	**0.975 ± 0.021**	0.921 ± 0.096	0.413 ± 0.201	0.662 ± 0.178	0.035 ± 0.019	0.438 ± 0.216	0.114 ± 0.041	0.565 ± 0.077	0.019 ± 0.005
ARES-Static	**0.985 ± 0.008**	**0.991 ± 0.005**	**0.985 ± 0.002**	**0.892 ± 0.014**	**0.999 ± 0.000**	0.942 ± 0.007	0.984 ± 0.001	0.806 ± 0.025	**0.969 ± 0.004**	**0.291 ± 0.014**	**0.956 ± 0.019**	**0.712 ± 0.086**	**0.948 ± 0.049**	**0.249 ± 0.091**
ARES-Dynamic	**0.982 ± 0.005**	**0.990 ± 0.002**	0.876 ± 0.002	0.486 ± 0.008	**0.999 ± 0.000**	0.961 ± 0.017	0.984 ± 0.001	0.816 ± 0.009	0.863 ± 0.103	0.090 ± 0.049	0.940 ± 0.015	**0.631 ± 0.064**	0.853 ± 0.039	0.094 ± 0.041
Performance Gain (%)	+2.8	+1.4	+1.4	+2.5	0.0	−1.4	−1.3	−15.2	+12.8	+21.4	+4.5	+23.5	+1.9	+11.8

Table 1: Evaluation of ARES **'s performance against competitors in identifying anomalies across different datasets.**

Model	DARPA		UNSW-NB15		ISCX2012		CIC-IDS2017		CTU-13					
									Scenario 1		Scenario 10		Scenario 13	
	F1-Score	B-Accuracy	F1-Score	B-Accuracy	F1-Score	B-Accuracy	F1-Score	B-Accuracy	F1-Score	B-Accuracy	F1-Score	B-Accuracy	F1-Score	B-Accuracy
MIDAS-F	0.804 ± 0.003	0.595 ± 0.008	0.636 ± 0.004	0.834 ± 0.005	0.485 ± 0.030	0.974 ± 0.003	0.797 ± 0.019	0.980 ± 0.002	0.051 ± 0.003	0.630 ± 0.040	0.291 ± 0.010	0.667 ± 0.015	0.045 ± 0.009	0.639 ± 0.090
MIDAS-R	0.815 ± 0.002	0.835 ± 0.004	0.599 ± 0.000	0.799 ± 0.001	0.116 ± 0.058	0.573 ± 0.048	0.309 ± 0.004	0.832 ± 0.003	0.047 ± 0.000	0.619 ± 0.000	0.061 ± 0.113	0.285 ± 0.187	0.033 ± 0.000	0.518 ± 0.000
MIDAS	0.741 ± 0.006	0.754 ± 0.012	0.547 ± 0.000	0.737 ± 0.000	0.054 ± 0.000	0.553 ± 0.004	0.174 ± 0.005	0.619 ± 0.005	0.047 ± 0.000	0.622 ± 0.000	0.268 ± 0.003	0.625 ± 0.007	0.037 ± 0.000	0.583 ± 0.000
AnoEdge-G	0.809 ± 0.000	0.831 ± 0.002	0.530 ± 0.032	0.739 ± 0.021	0.056 ± 0.002	0.587 ± 0.016	0.644 ± 0.047	0.943 ± 0.016	0.034 ± 0.001	0.491 ± 0.005	0.220 ± 0.002	0.515 ± 0.007	0.032 ± 0.000	0.502 ± 0.001
AnoEdge-L	0.805 ± 0.000	0.835 ± 0.001	0.573 ± 0.022	0.773 ± 0.015	0.087 ± 0.009	0.738 ± 0.035	0.734 ± 0.040	0.942 ± 0.020	0.040 ± 0.003	0.520 ± 0.009	0.331 ± 0.012	0.679 ± 0.018	0.006 ± 0.001	0.404 ± 0.05
SLADE-H	0.844 ± 0.051	0.702 ± 0.146	0.827 ± 0.112	0.897 ± 0.076	0.556 ± 0.492	0.778 ± 0.247	0.587 ± 0.221	0.881 ± 0.162	0.096 ± 0.037	0.769 ± 0.147	0.095 ± 0.172	0.485 ± 0.174	0.060 ± 0.009	0.739 ± 0.058
ARES-Static	**0.970 ± 0.005**	**0.968 ± 0.001**	**0.939 ± 0.002**	**0.984 ± 0.001**	**0.964 ± 0.012**	**0.999 ± 0.000**	**0.894 ± 0.005**	**0.991 ± 0.000**	**0.334 ± 0.089**	**0.957 ± 0.022**	**0.711 ± 0.142**	**0.849 ± 0.095**	**0.343 ± 0.194**	**0.867 ± 0.203**
ARES-Dynamic	0.962 ± 0.006	**0.962 ± 0.007**	0.622 ± 0.055	0.815 ± 0.067	0.437 ± 0.482	0.722 ± 0.247	**0.899 ± 0.005**	**0.999 ± 0.000**	0.104 ± 0.109	0.579 ± 0.121	**0.612 ± 0.092**	**0.838 ± 0.061**	0.127 ± 0.110	0.585 ± 0.084
Performance Gain (%)	+12.6	+13.3	+11.2	+8.7	+40.8	+2.5	+10.2	+1.9	+23.8	+18.8	+4.5	+38.0	+28.3	+12.8

Table 2: Evaluation of ARES **with thresholding discrimination.** ARES **with adaptive thresholding is compared against the optimal threshold devised for each competitor.**

Dataset	Cyber-Attack	ROC-AUC	AP
DARPA	Denial-of-Service	0.989 ± 0.008	0.993 ± 0.004
	Surveillance/Probing	0.998 ± 0.001	0.999 ± 0.001
	Remote-to-Local	0.972 ± 0.012	0.982 ± 0.008
	User-to-Root	1.000 ± 0.000	1.000 ± 0.000
UNSW-NB15	Exploits	0.984 ± 0.001	0.828 ± 0.012
	Reconnaissance	0.984 ± 0.001	0.851 ± 0.014
	DoS	0.987 ± 0.002	0.786 ± 0.022
	Shellcode	0.980 ± 0.002	0.840 ± 0.017
	Fuzzers	0.984 ± 0.002	0.870 ± 0.014
	Worms	0.991 ± 0.004	0.942 ± 0.030
	Backdoors	0.980 ± 0.003	0.705 ± 0.024

Table 3: Anomaly detection performance (ROC-AUC and AP) for specific cyber-attack types in the DARPA and UNSW-NB15 datasets.

Model	DARPA	UNSW-NB15	ISCX2012	CIC-IDS2017	CTU-13 Scenario 1	CTU-13 Scenario 10	CTU-13 Scenario 13
MIDAS-F	0.39	0.43	1.26	0.58	17.02	8.32	12.24
MIDAS-R	0.21	0.10	0.21	0.42	3.69	1.24	1.89
MIDAS	0.03	0.02	0.02	0.06	0.24	0.08	0.11
AnoEdge-G	39.41	23.17	13.62	96.55	31.51	11.87	24.09
AnoEdge-L	0.33	0.32	0.22	0.64	2.29	0.71	0.93
SLADE-H	763.09	189.63	29.83	1539.85	505.57	67.07	230.36
ARES-Static	7.83	5.40	14.13	51.36	59.52	44.85	108.26
ARES-Dynamic	10.84	6.11	6.76	90.76	74.33	51.08	70.20

Table 4: Running times (in seconds) across the used datasets

only provide the results for the CTU-13 Scenario 1 dataset. More details on the ablation study are available in Appendix C.

We also measure the statistical robustness of our findings. Table 6 presents the ROC-AUC and AP performance of ARES compared to competing models, on different subsamples of the DARPA dataset. The latter is particularly challenging as it comprises a large number (59.9%) of edge anomalies. For each seed used, we uniformly sampled 50% of the test set and computed the anomaly detection scores. As we can see, the results are consistent and stable along all runs,

Model	ROC-AUC	AP
GNN-RecErr	0.394 ± 0.000	0.022 ± 0.000
GCN+HST	0.707 ± 0.173	0.066 ± 0.032
GAT+HST	0.654 ± 0.009	0.034 ± 0.005
GraphSAGE+RRCF	0.804 ± 0.064	0.072 ± 0.025
ARES	**0.969 ± 0.004**	**0.291 ± 0.014**

Table 5: Ablation study on the CTU-13 Scenario 1 dataset demonstrates the effectiveness of combining GraphSAGE embeddings with Half-Space Trees.

witnessing the robustness of the proposed model in handling such a challenging dataset. Experiments concerning the other datasets are in Appendix C.

Model	ROC-AUC	AP
MIDAS-F	0.961 ± 0.005	0.979 ± 0.003
AnoEdge-L	0.910 ± 0.007	0.955 ± 0.003
SLADE-H	0.900 ± 0.094	**0.912 ± 0.091**
ARES-Static	**0.991 ± 0.005**	**0.994 ± 0.003**
ARES-Dynamic	**0.989 ± 0.007**	**0.994 ± 0.003**

Table 6: Comparison of ARES **against competitors by subsampling the dataset DARPA**

6 Conclusion

We proposed ARES, an unsupervised anomaly detection method that combines Graph Neural Networks (GNNs) with Half-Space Trees to analyze edge streams efficiently, ensuring constant-time

processing for each incoming edge. The model is equipped with a simple yet effective supervised algorithmic mechanism for determining a threshold to differentiate between normal and anomalous edges. Extensive empirical evaluation demonstrates its competitive advantage over state-of-the-art models.

There are several potential directions for improving this framework, which should be explored in future research. One key area is the development of more complex continual learning procedure to better update the GNN, which would enhance the quality of the embeddings over time. Additionally, methods to reduce computational overhead by accelerating GNN calculations should be considered. Another area for improvement is in the proposed thresholding method. Currently, ARES uses a supervised approach, but future work should explore unsupervised techniques that can still capture diversity in the distribution of anomaly scores [4, 17, 23, 40, 47].

Acknowledgments

This work has been partially funded by (i) Project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union – NextGenerationEU; (ii) MUR on D.M. 352/2022, PNRR Ricerca, CUP H23C22000550005.

References

[1] Mohiuddin Ahmed, Abdun Naser Mahmood, and Jiankun Hu. 2016. A survey of network anomaly detection techniques. *J. Netw. Comput. Appl.* 60 (2016), 19–31.

[2] Mohiuddin Ahmed, Abdun Naser Mahmood, and Md. Rafiqul Islam. 2016. A survey of anomaly detection techniques in financial domain. *Future Gener. Comput. Syst.* 55 (2016), 278–288.

[3] Leman Akoglu, Hanghang Tong, and Danai Koutra. 2015. Graph based anomaly detection and description: a survey. *Data Min. Knowl. Discov.* 29, 3 (2015), 626–688.

[4] Muhammad Qasim Ali, Ehab Al-Shaer, Hassan Khan, and Syed Ali Khayam. 2013. Automated Anomaly Detector Adaptation using Adaptive Threshold Tuning. *ACM Trans. Inf. Syst. Secur.* 15, 4 (2013), 17:1–17:30.

[5] Caleb Belth, Xinyi Zheng, and Danai Koutra. 2020. Mining Persistent Activity in Continually Evolving Networks. In *Procs. of the 26th ACM SIGKDD Conf.* KDD 2020, 934–944.

[6] Siddharth Bhatia, Bryan Hooi, Minji Yoon, Kijung Shin, and Christos Faloutsos. 2020. Midas: Microcluster-Based Detector of Anomalies in Edge Streams. In *Procs of the 34th AAAI Conf. (AAAI 2020)*. 3242–3249.

[7] Siddharth Bhatia, Rui Liu, Bryan Hooi, Minji Yoon, et al. 2022. Real-Time Anomaly Detection in Edge Streams. *ACM Trans. Knowl. Discov. Data* 16, 4 (2022), 75:1–75:22.

[8] Siddharth Bhatia, Mohit Wadhwa, Kenji Kawaguchi, et al. 2023. Sketch-Based Anomaly Detection in Streaming Graphs. In *Procs. of the 29th ACM SIGKDD Conf. (KDD 2023)*. 93–104.

[9] Leo Breiman, Jerome Friedman, Charles J. Stone, and R.A. Olshen. 1984. *Classification and Regression Trees.* Chapman and Hall/CRC.

[10] Yen-Yu Chang, Pan Li, Rok Sosic, M. H. Afifi, Marco Schweighauser, and Jure Leskovec. 2021. F-FADE: Frequency Factorization for Anomaly Detection in Edge Streams. In *Procs. of the 14th ACM WSDM Conf. (WSDM 2021).*

[11] Anshika Chaudhary, Himangi Mittal, and Anuja Arora. 2019. Anomaly Detection using Graph Neural Networks. In *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*. 346–350.

[12] Erica Coppolillo, Simone Mungari, and et al. Ettore Ritacco. 2025. Algorithmic Drift: A simulation framework to study the effects of recommender systems on user preferences. *Inf. Process. Manag.* 62, 5 (2025), 104125. https://doi.org/10.1016/j.ipm.2025.104125

[13] Graham Cormode and S. Muthukrishnan. 2005. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms* 55, 1 (2005), 58–75.

[14] Dhivya Eswaran and Christos Faloutsos. 2018. SedanSpot: Detecting Anomalies in Edge Streams. In *Procs. of the IEEE ICDM Conf. (ICDM 2018)*. 953–958.

[15] Matthias Fey and Jan Eric Lenssen. 2019. Fast Graph Representation Learning with PyTorch Geometric. *CoRR* abs/1903.02428 (2019).

[16] Sebastián García, Martin Grill, Jan Stiborek, and Alejandro Zunino. 2014. An empirical comparison of botnet detection methods. *Comput. Secur.* 45 (2014), 100–123.

[17] Amin Ghafouri, Waseem Abbas, Aron Laszka, Yevgeniy Vorobeychik, and Xenofon D. Koutsoukos. 2016. Optimal Thresholds for Anomaly-Based Intrusion Detection in Dynamical Environments. In *Procs. of the 7th GameSec Conf. (Lecture Notes in Computer Science, Vol. 9996)*. 415–434.

[18] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*. ACM, 855–864. https://doi.org/10.1145/2939672.2939754

[19] Sudipto Guha, Nina Mishra, Gourav Roy, and Okke Schrijvers. 2016. Robust Random Cut Forest Based Anomaly Detection on Streams. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016 (JMLR Workshop and Conference Proceedings, Vol. 48)*. 2712–2721.

[20] William L. Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *Procs. of the 30th NeurIP Conf. (NeurIPS 2017)*. 1024–1034. https://proceedings.neurips.cc/paper/2017/hash/5dd9db5e033da9c6fb5ba83c7a7ebea9-Abstract.html

[21] Hwan Kim, Byung Suk Lee, Won-Yong Shin, and Sungsu Lim. 2022. Graph Anomaly Detection With Graph Neural Networks: Current Status and Challenges. *IEEE Access* 10 (2022), 111820–111829.

[22] Thomas N. Kipf and Max Welling. 2016. Variational Graph Auto-Encoders. *CoRR* (2016). http://arxiv.org/abs/1611.07308

[23] Adrian Komadina, Mislav Martinic, Stjepan Gros, and Zeljka Mihajlovic. 2024. Comparing Threshold Selection Methods for Network Anomaly Detection. *IEEE Access* 12 (2024), 124943–124973.

[24] Jongha Lee, Sunwoo Kim, and Kijung Shin. 2024. SLADE: Detecting Dynamic Anomalies in Edge Streams without Labels via Self-Supervised Learning. In *Procs. of the 30th ACM SIGKDD Conf. (KDD 2024)*. 1506–1517.

[25] Richard Lippmann, Robert K. Cunningham, David J. Fried, et al. 1999. Results of the DARPA 1998 Offline Intrusion Detection Evaluation. In *Recent Advances in Intrusion Detection, Second International Workshop, RAID 1999, West Lafayette, Indiana, USA, September 7-9, 1999.*

[26] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2012. Isolation-Based Anomaly Detection. *ACM Trans. Knowl. Discov. Data* (2012). doi:10.1145/2133360.2133363

[27] Yixin Liu, Kaize Ding, Qinghua Lu, et al. 2023. Towards Self-Interpretable Graph-Level Anomaly Detection. In *Procs. of the 36th NeurIPS Conf. (NeurIPS 2023)*. http://papers.nips.cc/paper_files/paper/2023/hash/1c6f06863df46de009a7a41b41c95cad-Abstract-Conference.html

[28] Xiaoxiao Ma, Jia Wu, Shan Xue, et al. 2023. A Comprehensive Survey on Graph Anomaly Detection With Deep Learning. *IEEE Trans. Knowl. Data Eng.* 35, 12 (2023), 12012–12038.

[29] Matthew V. Mahoney. 2003. Network Traffic Anomaly Detection Based on Packet Bytes. In *Procs. of the ACM SAC Conf.* SAC Conf., 346–350.

[30] Jacob Montiel, Max Halford, Saulo Martiello Mastelini, et al. 2021. River: machine learning for streaming data in Python. *J. Mach. Learn. Res.* 22 (2021), 110:1–110:8.

[31] Nour Moustafa and Jill Slay. 2015. UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In *Procs of MilCIS Conf. (MilCIS 2015)*. 1–6.

[32] Evangelos E. Papalexakis, Alex Beutel, and Peter Steenkiste. 2012. Network Anomaly Detection Using Co-clustering. In *Procs. of the ASONAM Conf. (ASONAM 2012)*.

[33] Stephen Ranshous, Steve Harenberg, Kshitij Sharma, and Nagiza F. Samatova. 2016. A Scalable Approach for Outlier Detection in Edge Streams Using Sketch-based Approximations. In *Procs. of the 2016 SIAM SDM Conf. (SDM 2016)*. 189–197.

[34] Stephen Ranshous, Shitian Shen, Danai Koutra, Steve Harenberg, Christos Faloutsos, and Nagiza F. Samatova. 2015. Anomaly detection in dynamic networks: a survey. *WIREs Computational Statistics* 7, 3 (2015), 223–247.

[35] Amani Abou Rida, Rabih Amhaz, and Pierre Parrend. 2021. Evaluation of Anomaly Detection for Cybersecurity Using Inductive Node Embedding with Convolutional Graph Neural Networks. In *Procs. of the 10th COMPLEX NETWORKS Conf. (COMPLEX NETWORKS 2021, Vol. 1016)*. 563–574.

[36] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. 2020. Temporal Graph Networks for Deep Learning on Dynamic Graphs. In *ICML 2020 Workshop on Graph Representation Learning.*

[37] Durgesh Samariya and Amit Thakkar. 2023. A comprehensive survey of anomaly detection algorithms. *Annals of Data Science* 10, 3 (2023), 829–850.

[38] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani. 2018. Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. In *Procs. of the 4th ICISSP Conf. (ICISSP 2018)*. 108–116.

[39] Ali Shiravi, Hadi Shiravi, Mahbod Tavallaee, and Ali A. Ghorbani. 2012. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *Comput. Secur.* 31, 3 (2012), 357–374.

[40] Alban Siffer, Pierre-Alain Fouque, Alexandre Termier, and Christine Largouët. 2017. Anomaly Detection in Streams with Extreme Value Theory. In *Procs. of the 23rd ACM SIGKDD Conf. (KDD 2017)*. 1067–1075.

[41] Swee Chuan Tan, Kai Ming Ting, and Fei Tony Liu. 2011. Fast Anomaly Detection for Streaming Data. In *Procs. of the 22nd IJCAI Conf. (IJCAI 2011)*. 1511–1516.

[42] Jianheng Tang, Jiajin Li, Ziqi Gao, and Jia Li. 2022. Rethinking Graph Neural Networks for Anomaly Detection. In *Procs. of the 39th ICML Conf.* 21076–21089.

Dataset	$\|V\|$	$\|E\|$	% anomalies
DARPA	25K	4.5M	59.9%
UNSW-NB15	50	2.5M	12.8%
ISCX2012	31K	1.1M	4.2%
CIC-IDS2017	33K	7.8M	7.4%
CTU-13 Scenario 1	607K	2.8M	1.5%
CTU-13 Scenario 10	198K	1.3M	8.1%
CTU-13 Scenario 13	316K	1.9M	2.1%

Table 8: Summary statistics on the datasets.

[43] Chun Wang, Shirui Pan, Guodong Long, et al. 2017. MGAE: Marginalized Graph Autoencoder for Graph Clustering. In *Procs. of the ACM CIKM Conf. (CIKM 2017)*. 889–898.

[44] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. 2021. A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems* 32, 1 (2021), 4–24.

[45] Miao Xie, Song Han, Biming Tian, and Sazia Parvin. 2011. Anomaly detection in wireless sensor networks: A survey. *J. Netw. Comput. Appl.* (2011).

[46] Hongzuo Xu, Guansong Pang, Yijie Wang, and Yongjun Wang. 2023. Deep Isolation Forest for Anomaly Detection. *IEEE Trans. Knowl. Data Eng.* 35, 12 (2023), 12591–12604.

[47] Xue Yang, Enda Howley, and Michael Schukat. 2023. ADT: Agent-based Dynamic Thresholding for Anomaly Detection. *CoRR* abs/2312.01488 (2023).

[48] Wenchao Yu, Wei Cheng, Charu C. Aggarwal, et al. 2018. NetWalk: A Flexible Deep Embedding Approach for Anomaly Detection in Dynamic Networks. In *Procs. of the 24th ACM SIGKDD (KDD 2018)*. 2672–2681.

[49] Daniele Zambon, Lorenzo Livi, and Cesare Alippi. 2022. Graph iForest: Isolation of anomalous and outlier graphs. In *Procs. of the IJCNN Conf. (IJCNN 2022)*. 1–8.

[50] Zikai Zhang, Yidong Li, Wei Wang, Haifeng Song, and Hairong Dong. 2022. Malware detection with dynamic evolving graph convolutional networks. *Int. J. Intell. Syst.* 37, 10 (2022), 7261–7280.

[51] Li Zheng, Zhenpeng Li, Jian Li, et al. 2019. AddGraph: Anomaly Detection in Dynamic Graph Using Attention-based Temporal GCN. In *Procs. of the 28th IJCAI Conf. (IJCAI 2019)*. 4419–4425.

[52] Dali Zhu, Yuchen Ma, and Yinlong Liu. 2020. Anomaly Detection with Deep Graph Autoencoders on Attributed Networks. In *Procs. of the IEEE ISCC Conf. (ISCC 2020)*. 1–6.

[53] Francesco Zola, Lander Segurola-Gil, Jan Lukas Bruse, Mikel Galar, and Raul Orduna Urrutia. 2022. Network traffic analysis through node behaviour classification: a graph-based approach with temporal dissection and data-level preprocessing. *Comput. Secur.* 115 (2022), 102632. doi:10.1016/J.COSE.2022.102632

A Training Details and Sensitivity Analysis

GNN hyperparameters (layers, neurons, lr) are tuned to ensure meaningful node embeddings, thus the chosen configuration minimizes the loss. After integrating the GNN with HST, we perform a second tuning phase. The list of hyperparameters is shown in Table 7.

GNN Training	
Layers	2–6
Channels	Hidden: 16; Output: 4, 8, 16
Learning Rate	0.001
Sampling	Uniform
Aggregation	Mean
Anomaly Scoring	
Edge Embedding	Eq. 1, Eq. 2
Weights	(1.0, 0.0, 0.0), (0.33, 0.33, 0.33)
Cache Size	64, 32, 16, 8, 4
Trees	8, 16, 32, 64
Depth	3, 6, 9, 12
HST Window	8, 64, 512, 1024, 2048, 4096, 8192, 10^5, 10^6

Table 7: Hyperparameters and settings used for GNN training and anomaly scoring.

Further, Table 8 reports some statistics about the seven real-world datasets used for the experiments

In Tables 9 and 10 we report the AUC and AP by varying the number of trees and the depth for each tree, respectively. Generally, more trees and more depth improve robustness by reducing variance, but the optimal number is dataset-dependent.

Further, weights in Eq. 4 and the edge embedding strategy (Eq. 1 and Eq. 2) are also part of the hyperparameter tuning process. To provide further clarity, we include an experiment analyzing their impact on model performance in Tables 11 and 12.

B Further Experiments

To strengthen and broaden our experimental evaluation, we provide two additional experiments using non-cybersecurity related datasets, namely Bitcoin-Alpha and Bitcoin-OTC. These datasets represent financial trust networks, where nodes correspond to users and edges indicate trust scores assigned from one user to another. In this setting, abnormality is determined by the likelihood that a user is suspicious based on the proportion of low trust scores they receive. This approach differs significantly from the scenarios considered in our experiments, which are rooted in cybersecurity contexts where anomalies come from malicious activities such as cyber-attacks. For consistency and comparison, we adopted the same experimental setup as SLADE, including temporal splits and abnormality labeling, allowing us to directly reference their reported results.

In Table 13, we present a comparative analysis of ARES (in both static and dynamic configurations) against SLADE and other baseline methods such as AnoEdge-l and MIDAS. SLADE consistently achieves the highest AUC, while the proposed model outperforms all baselines, including SLADE, in terms of average precision (AP). These additional experiments highlight the potentiality and generality of the proposed method.

We also assess the statistical robustness of our framework by evaluating its performance under varying data conditions. Table 1 in the Online Appendix reports the ROC-AUC and AP scores for ARES and baseline models across several datasets, where we uniformly sample %50 of the test set. Notably, ARES demonstrates greater stability, maintaining high detection performance with minimal degradation, whereas baseline methods often exhibit larger performance drops or increased variability. This highlights the robustness of ARES not only in full-data scenarios but also under constrained or noisy data conditions.

C Ablation Study

Half-Space Trees contribution. HST provides (i) a simple but effective way of detecting anomalies, and (ii) it adapts to concept drift by continuously updating statistics, capturing data distribution shifts. Nonetheless, HST is not directly applicable to graph data, making an isolated ablation unfeasible. The GNN, used solely for node embeddings, can reflect drift when an increase in the reconstruction error occurs, offering a potential proxy for change detection. While not directly explored in our experiments, we acknowledge this potential and plan to incorporate continual learning methods to improve adaptability in future work.

Trees	DARPA		UNSW-NB15		ISCX2012		CIC-IDS2017		CTU-13					
									Scenario 1		Scenario 10		Scenario 13	
	ROC-AUC	AP	ROC-AUC	AP	ROC-AUC	AP	ROC-AUC	AP	ROC-AUC	AP	ROC-AUC	AP	ROC-AUC	AP
8	0.985 ± 0.008	0.991 ± 0.005	0.985 ± 0.002	0.891 ± 0.014	0.992 ± 0.009	0.701 ± 0.247	0.984 ± 0.001	0.806 ± 0.025	0.969 ± 0.004	0.290 ± 0.015	0.938 ± 0.029	0.661 ± 0.101	0.668 ± 0.267	0.118 ± 0.126
16	0.988 ± 0.003	0.993 ± 0.001	0.983 ± 0.001	0.883 ± 0.011	0.996 ± 0.005	0.833 ± 0.189	0.985 ± 0.001	0.804 ± 0.031	0.968 ± 0.004	0.280 ± 0.028	0.958 ± 0.020	0.728 ± 0.099	0.794 ± 0.216	0.161 ± 0.139
32	0.990 ± 0.002	0.994 ± 0.001	0.983 ± 0.001	0.883 ± 0.011	0.999 ± 0.001	0.935 ± 0.029	0.985 ± 0.001	0.781 ± 0.038	0.967 ± 0.001	0.246 ± 0.023	0.962 ± 0.011	0.659 ± 0.069	0.859 ± 0.185	0.173 ± 0.105
64	0.991 ± 0.000	0.994 ± 0.000	0.984 ± 0.001	0.889 ± 0.009	0.999 ± 0.001	0.944 ± 0.007	0.985 ± 0.001	0.766 ± 0.033	0.966 ± 0.000	0.228 ± 0.021	0.961 ± 0.007	0.640 ± 0.033	0.925 ± 0.116	0.244 ± 0.107

Table 9: Sensitivity analysis by varying the number of trees.

Depths	DARPA		UNSW-NB15		ISCX2012		CIC-IDS2017		CTU-13					
									Scenario 1		Scenario 10		Scenario 13	
	ROC-AUC	AP	ROC-AUC	AP	ROC-AUC	AP	ROC-AUC	AP	ROC-AUC	AP	ROC-AUC	AP	ROC-AUC	AP
3	0.985 ± 0.008	0.991 ± 0.005	0.984 ± 0.003	0.889 ± 0.018	0.999 ± 0.000	0.944 ± 0.007	0.984 ± 0.001	0.806 ± 0.025	0.966 ± 0.004	0.260 ± 0.024	0.941 ± 0.076	0.752 ± 0.225	0.925 ± 0.116	0.244 ± 0.107
6	0.986 ± 0.004	0.992 ± 0.002	0.983 ± 0.002	0.882 ± 0.012	0.999 ± 0.000	0.949 ± 0.006	0.985 ± 0.001	0.810 ± 0.027	0.969 ± 0.004	0.290 ± 0.015	0.942 ± 0.040	0.660 ± 0.133	0.637 ± 0.188	0.064 ± 0.075
9	0.987 ± 0.004	0.993 ± 0.002	0.984 ± 0.002	0.880 ± 0.016	0.999 ± 0.000	0.951 ± 0.007	0.984 ± 0.001	0.817 ± 0.011	0.970 ± 0.006	0.292 ± 0.037	0.957 ± 0.020	0.712 ± 0.094	0.503 ± 0.011	0.027 ± 0.003
12	0.988 ± 0.005	0.994 ± 0.002	0.985 ± 0.002	0.891 ± 0.014	0.999 ± 0.000	0.951 ± 0.004	0.985 ± 0.001	0.805 ± 0.032	0.970 ± 0.004	0.290 ± 0.027	0.949 ± 0.023	0.659 ± 0.131	0.499 ± 0.013	0.024 ± 0.003

Table 10: Sensitivity analysis by varying the tree depth.

Weights	DARPA		UNSW-NB15		ISCX2012		CIC-IDS2017		CTU-13					
									Scenario 1		Scenario 10		Scenario 13	
	ROC-AUC	AP	ROC-AUC	AP	ROC-AUC	AP	ROC-AUC	AP	ROC-AUC	AP	ROC-AUC	AP	ROC-AUC	AP
0.0, 0.0, 1.0	0.985 ± 0.008	0.991 ± 0.005	0.985 ± 0.002	0.892 ± 0.014	0.999 ± 0.000	0.942 ± 0.007	0.984 ± 0.001	0.806 ± 0.025	0.969 ± 0.004	0.291 ± 0.014	0.956 ± 0.019	0.712 ± 0.086	0.948 ± 0.049	0.249 ± 0.091
0.33, 0.33, 0.33	0.977 ± 0.007	0.991 ± 0.003	0.657 ± 0.255	0.433 ± 0.242	0.931 ± 0.005	0.255 ± 0.012	0.966 ± 0.080	0.856 ± 0.244	0.964 ± 0.003	0.267 ± 0.015	0.985 ± 0.004	0.787 ± 0.047	0.520 ± 0.005	0.023 ± 0.004

Table 11: Sensitivity analysis by varying nodes and edge weights for anomaly scoring.

Edge Embeddings	DARPA		UNSW-NB15		ISCX2012		CIC-IDS2017		CTU-13					
									Scenario 1		Scenario 10		Scenario 13	
	ROC-AUC	AP	ROC-AUC	AP	ROC-AUC	AP	ROC-AUC	AP	ROC-AUC	AP	ROC-AUC	AP	ROC-AUC	AP
Eq.1	0.985 ± 0.008	0.991 ± 0.005	0.109 ± 0.004	0.149 ± 0.005	0.431 ± 0.270	0.116 ± 0.210	0.984 ± 0.001	0.806 ± 0.025	0.969 ± 0.004	0.290 ± 0.015	0.770 ± 0.106	0.391 ± 0.110	0.469 ± 0.014	0.023 ± 0.003
Eq.2	0.844 ± 0.048	0.866 ± 0.055	0.985 ± 0.002	0.891 ± 0.014	0.999 ± 0.000	0.942 ± 0.007	0.981 ± 0.004	0.777 ± 0.038	0.886 ± 0.052	0.084 ± 0.025	0.957 ± 0.020	0.712 ± 0.094	0.925 ± 0.116	0.244 ± 0.107

Table 12: Sensitivity analysis by varying the edge embedding strategy.

Model	Bitcoin Alpha		Bitcoin OTC	
	ROC-AUC	AP	ROC-AUC	AP
MIDAS-F	0.646 ± 0.011	0.131 ± 0.001	0.622 ± 0.021	0.161 ± 0.000
AnoEdge-L	0.625 ± 0.002	0.125 ± 0.011	0.661 ± 0.19	0.170 ± 0.001
SLADE-H	**0.769 ± 0.004**	0.149 ± 0.001	**0.772 ± 0.003**	0.205 ± 0.001
ARES-Static	0.729 ± 0.004	0.161 ± 0.003	0.757 ± 0.013	**0.264 ± 0.026**
ARES-Dynamic	0.731 ± 0.004	**0.167 ± 0.007**	0.592 ± 0.040	0.111 ± 0.012

Table 13: Performance comparison between ARES **and baseline methods on the Bitcoin Alpha and Bitcoin OTC datasets.**

That said, we compare ARES with a GNN-only variant using reconstruction error as the anomaly score following prior work [28] (GNN-RecErr) and an ablation study comparing HST with RRCF [19] (GraphSage+RRCF). Results (AUC and AP) reported in Table 2 (see Online Appendix) demonstrate ARES significantly outperforms the baselines.

Notably, as previously mentioned, HST consistently outperforms the more recent RRCF, achieving superior accuracy while being 2x to 9x faster in inference time across multiple datasets as depicted in Table 3 in the Online Appendix. These results further validate HST as a robust and efficient choice for real-time anomaly detection in streaming graph environments.

Different Node/Edge embedders. GraphSAGE has been shown to produce meaningful representations of nodes and edges that effectively capture the underlying graph structure. We conducted further experiments comparing ARES (GraphSAGE + HST) with GCN+HST and GAT+HST described in Table 2 (Online Appendix).

D Thresholding mechanism limitations

Additional observations can also be made on the adaptive thresholding method, whose sensitivity to overfitting depends on the choice of the validation set. In our experiments, we partition the data according to the temporal dimension. In presence of dramatic concept drifts, this can result in a poor choice of the optimal thresholds, as witnessed by the unstable results in terms of F1 and B-Accuracy. For example, the validation set for ISCX2012 comprises 99.962% normal edges and only 0.038% anomalies, whereas the test set consists of 97.6% normal edges and 2.4% anomalous edges. In such, computing a threshold according to a validation set which exhibits a different distribution of anomalies may result in unstable results.