GPML: Graph Processing for Machine Learning

Majed Jaber[1,2], Julien Michel[1,2], Nicolas Boutry[2], and Pierre Parrend[1,2]

[1] EPITA, Laboratoire de Recherche de l'EPITA (LRE), 14-16 Rue Voltaire, 94270 Le Kremlin Bicêtre {julien.michel, nicolas.boutry, pierre.parrend}@epita.fr
[2] ICube UMR7357, Université de Strasbourg, CNRS, F-67000 Strasbourg, France
majed.jaber@etu.unistra.fr

Abstract. The dramatic increase of complex, multi-step, and rapidly evolving attacks in dynamic networks involves advanced cyber-threat detectors. The GPML (Graph Processing for Machine Learning) library addresses this need by transforming raw network traffic traces into graph representations, enabling advanced insights into network behaviors. The library provides tools to detect anomalies in interaction and community shifts in dynamic networks. GPML supports community and spectral metrics extraction, enhancing both real-time detection and historical forensics analysis. This library supports modern cybersecurity challenges with a robust, graph-based approach.

Keywords: Graph processing · Machine Learning · Python-based library · Spectral graph analysis · Graph communities · Attack detection

1 Metadata

Nr.	Code metadata description	Please fill in this column
C1	Current code version	1.0.0
C2	Permanent link to code/repository used for this code version	https://github.com/lre-security-systems-team/gpml
C3	Permanent link to Reproducible Capsule	https://www.kaggle.com/code/majedjaber/gpml-reproducible-capsule
C4	Legal Code License	ISC https://opensource.org/license/isc-license-txt
C5	Code versioning system used	git
C6	Software code languages, tools, and services used	python
C7	Compilation requirements, operating environments & dependencies	https://github.com/lre-security-systems-team/gpml/blob/main/requirements.txt
C8	If available Link to developer documentation/manual	https://github.com/lre-security-systems-team/gpml/blob/main/README.md
C9	Support email for questions	{julien.michel,majed.jaber}@epita.fr

Table 1. METADATA

2 Motivation and significance

In today's digital landscape, network security faces growing challenges due to the increasing complexity of cyber-threats [12]. Attackers constantly improve their

techniques, targeting vulnerabilities across interconnected devices, systems, and services. Traditional security measures often struggle to keep up, as they primarily rely on signature-based detection [5] or rule-based methods [10], which may not capture emerging threats in network traffic. To effectively monitor, analyze, and secure network environments, new approaches are required—approaches that can handle large-scale and dynamic data from highly interconnected environments while addressing the alert fatigue in security operations [11].

Two main approaches emerge from the literature: graph analytics, or complex network analysis, which characterizes the connectivity properties on the graph itself, and embedding analysis, which extracts information about node surroundings for each node. Graph analytics leverages Community, Spectral, or Complex Network information to quantify the relative connectivity of node groups, the structure of the connections between nodes, or the position wrt. to whole network. Embedding analysis typically rely on Graph Neural Networks (GNN) and their variants. In all cases, graph-based models represent entities as nodes and their interactions as edges and enable effective analysis of network traffic. Graph Neural Networks (GNNs) extend this by learning at the node, edge, and graph levels [13]. While traditional GNNs rely on node features and topology [6], recent models integrate edge features to better capture interaction-specific information such as packet volume or connection type. Edge-aware variants like E-GraphSage [7] and NE-GConv [1] incorporate these features during aggregation, improving the precision of edge-level predictions. NE-GConv performs binary edge classification, whereas E-GraphSage supports both binary and multi-class outputs.

Dynamic Graph Community (DGC) metrics significantly enhance detection performance in network traffic classification tasks. Enriching the baseline feature set, it captures dynamic interaction patterns in network data. The spectral method in GPML library ($SPEC - TRA$) analyzes graph states over time using time windows and classifies attacks based on aggregated behaviors. While GNNs aim to identify specific malicious edges, $SPECTRA$ detects broader anomalous evolutions over multiple attack categories.

GPML library integrates both community and spectral analytics to leverage a graph-based approach to network security analysis. This library transforms raw network traffic data into graph representations [3]. This approach not only enhances the detection of known issues but also reveals hidden or emerging patterns within complex networks. By identifying interconnections through communities and spectral analysis, and monitoring their changes over time, it becomes possible to detect unusual connectivity patterns, isolate suspicious nodes, and proactively respond to potential threats. Tracking both static and dynamic network metrics enables both real-time and historical analyses are critical. GPML library builds on established methods in network graph analysis and leverages widely used libraries like NetworkX [2] for graph operations and Pandas [8] for data handling.

In future studies, the GPML library could serve as a foundational tool for developing AI-driven solutions to detect sophisticated threats.

3 Software description

3.1 Supported metrics

The graph community metrics are calculated from a graph community partition at time t, and their dynamicity is calculated from the difference between time $t+1$ and t. Let V_t be the number of node of a community at time t. **Stability**= $\frac{|V_t \cap V_{t+1}| - |(V_t \cap \bar{V}_{t+1}) \cup (V_{t+1} \cap \bar{V}_t)|}{|V_t \cup V_{t+1}|}$, is a ratio of similarity between two consecutive states of a community. **Density** in a community is the probability for a node to be adjacent to any given node in the community, **Conductance** is the proportion of communications pointing outside the community and **Degree** are the number of edges going out of a node. Refer to [14] for their definition.
The spectral metrics are derived from the spectrum Λ_t of the Laplacian matrix at time t, for more details you can refer to our work about detecting attacks using spectral graph analysis [3]. We denote by $\Lambda\_t[i]$ the i^{th} eigenvalue, $i \in [1, n]$, sorted in increasing order, and by $\mathcal{Z}(t)$ the multiplicity of zero in Λ_t. **Connectedness**=$(\exp(1/\mathcal{Z}(t) - 1))$, measures interconnectivity in the network. Let $(\mathcal{N})$ be the number of network devices (e.g. switches and servers). **Flooding**=$((\frac{1}{\mathcal{N}} \sum_{i=\mathcal{Z}(t)+1}^{\mathcal{Z}(t)+\mathcal{N}} \Lambda\_t[i]) - 1)$ and **Wiriness**=$(\frac{1}{\mathcal{N}} \sum_{i=n-\mathcal{N}+1}^{n} \Lambda\_t[i])$,

3.2 Software architecture

The library is organized into three main components, as illustrated in Fig 1. The hierarchy starts with the root *gpml* directory, branching down into three primary sub-directories: *data_preparation*, *metrics*, and *visualization*.

- The *data_preparation* directory is responsible for preparing data, starting with CSV files. It contains three classes: *data_frame_handler.py*, *graph_extractor.py* and *time_series_extractor.py*
- The second directory, *metrics*, is central to our methodology, and computes spectral metrics in *spectral_metrics.py* as well as community metrics in *graph_community.py*.
- The final directory, *visualization*, focuses on presenting graphs of nodes and edges for user interpretation, using network traffic CSV files as input. Visualization can be done through *plot.py*, which extracts and plots graphs using the *networkx* library, or *graphviz.py*, which enhances representation and provides interactivity for HTML web versions.

3.3 Software Implementation

The library is structured under its main directory as follows:

- gpml - Contains the core processing modules for two primary methodologies: community and spectral graph detection, along with components for graph processing and visualization. This is the central part of the library, offering diverse functionalities that aid in attack classification and dataset exploration.

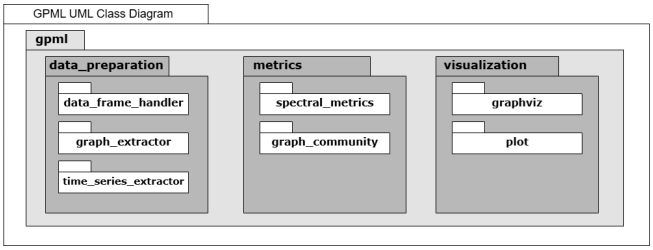


Fig. 1. UML diagram showing the structure of the GPML library

- data - Includes various datasets in CSV format.
- doc - Holds the library documentation, detailing the dataset functionalities and usage.
- test - Contains test cases for regression, providing examples that can be adapted to similar datasets using the specified constraints and parameters.

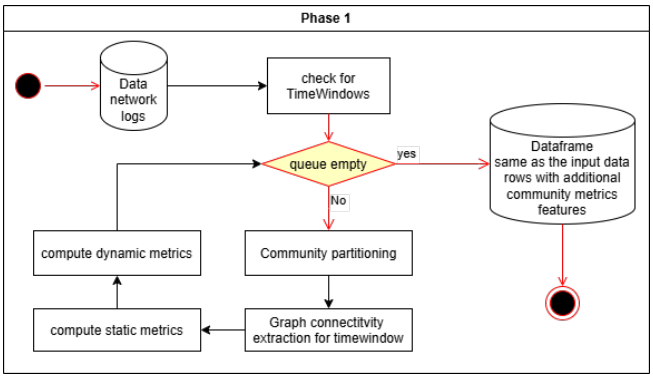


Fig. 2. UML workflow diagram community strategy

3.4 Software functionalities

The library functionalities can be divided into three main parts:

- Extracting community graph features,
- Extracting spectral graph features,
- Plotting connectivity graphs.

Additionally, because the library requires correct inputs to function normally, you must make sure your dataset contains the following common features that exist in every traffic network data:

– Timestamp for the arriving packets.
– Source and destination IP addresses.
– For spectral metrics extraction, you need in addition to the above, the total number of packets, size of bytes and the rate of packets. These features exists in every traffic network data logs.

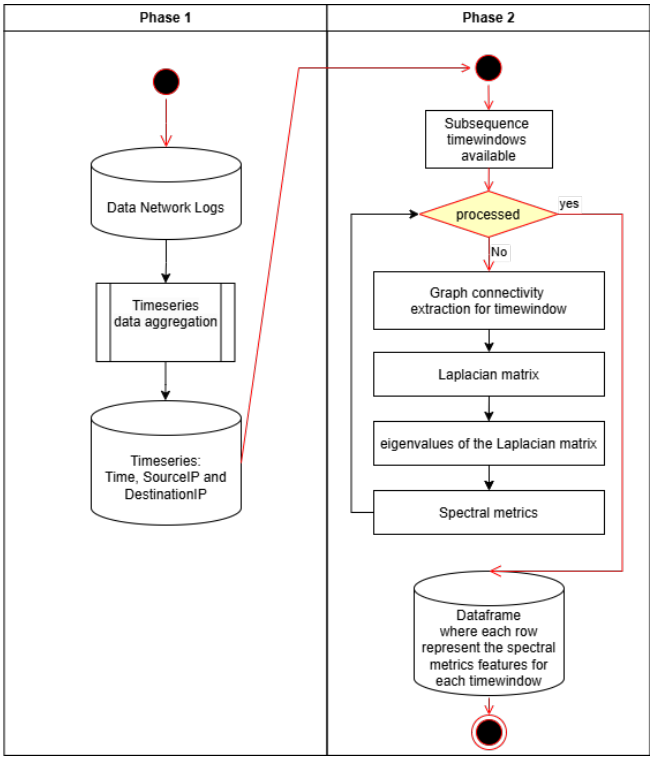


Fig. 3. UML workflow diagram for spectral graph strategy

The library provides a set of functionalities that help to add new features for better predictions. The main functionalities are:

```
time_series_extractor(df,stime,time_unit,features_list,
    sortby_list,groupby_list,aggregation_dict)
```

Listing 1.1. Extraction of time series

In the *time_series_extractor*, a time interval of $t = 1s$ is applied to the dataframe *df* provided by the user. Within this interval, the features are sorted according to *sortby_list* and grouped based on the features specified in the *groupby_list*.

The remaining features, such as packets, bytes and rates are aggregated using the functions defined in the *aggregation_dict* such as mean, avg and max/min functions. This transformation converts the dataset into a time series format, typically reducing the number of rows compared to the original dataset.

```
insert_graph_community_metrics(dataframe,time_interval,
    date_time,edge_source,edge_dest,label,name,date_timestamp
    , community_strategy,continuity)
```

Listing 1.2. Insertion of graph community metrics

The *insert_metrics_to_dataframe* function take a *pandas DataFrame* and for a given time windows and community partitioning strategy as the parameters *time_interval* and *community_strategy* will produce the corresponding community metrics and insert them back to the *DataFrame* as columns. The user needs to specify the column used for the time as *date_time* and the columns used for the nodes as lists in the *edge_source* and *edge_dest* parameters. A column from the *DataFrame* has to be set to *label* for the edges and a *name* for a suffix of the new columns in the returned *DataFrame*. A *continuity* parameter has been added to choose if the user allows holes in the timeline of the data. The input *DataFrame* is divided in time windows, for each time windows primarily, three underlying method will be called by this function: *gc_metrics_first_order(G)* which calculates metrics by one travel of the graph *G*, *gc_metrics_second_order(forder_metrics_c, forder_metrics_g)* which calculates metrics by one travel over the first order metrics, and *propagate_communities(g1, g2, center, center_t)* which creates the correspondence of communities from two graph at consecutive time windows. Dynamic community metrics are then computed from those metrics and propagated through communities; all of them are added to the output *DataFrame*.

```
spectral_metrics_extractor(ts,stime,saddr,daddr,pkts,
bytes_size,rate,lbl_category)
```

Listing 1.3. Extraction of spectral metrics

In the *spectral_metrics_extractor*, the parameters are provided by the user; however, constructing weighted edges within the network graph extracted over each time window requires selecting a specific feature. For this purpose, *pkts*, *bytes_size*, and *rate* are passed as inputs to weigh the graph during processing across three distinct topologies within each time window. Spectral metrics are then computed for each time window at the midpoint and the end. The output of this function is a new dataframe where each row represents a time window, including its corresponding common features, spectral features, and the associated label.

```
print_graph(dataset,graph_type,label,src_addr,dst_addr, sport
    ,dport,url,title,attack_name,src_mac,dst_mac)
```

Listing 1.4. Print graph

The *print_graph* function is designed to display the graph in two formats: a non-interactive format using *networkx* and an interactive *HTML* format that allows user interactions.

4 Illustrative examples

4.1 Code snippets

– Community metrics example

```
from datetime import date, timedelta, datetime
from gpml.data_preparation import data_frame_handler as
    ins_data
community_df = ins_data.insert_graph_community_metrics(df,
     timedelta(minutes=5),'Date time',['Source IP'], ['
    Destination IP'],'Label', 'ip5',date_timestamp=False,
    community_strategy='louvain',continuity=True)
```

Listing 1.5. Insertion of graph community metrics - Example

– Spectral metrics example
- Parameters:

```
features = ['stime', 'datetime', 'saddr', 'daddr', 'sport'
    , 'dport', 'pkts', 'bytes', 'rate', 'attack', '
    category', 'subcategory', 'weight', 'dur', 'mean', '
    sum', 'min', 'max','spkts', 'dpkts', 'srate', 'drate']
sortby_list = ['stime']
groupby_list = ['stime', 'datetime', 'saddr', 'daddr']
aggregation_dict = { 'pkts': 'sum', 'bytes': 'sum', '
    attack': 'first', 'category': 'first', 'subcategory':
    'first', 'rate': 'mean', 'dur': 'mean', 'mean': 'mean'
    , 'sum': 'mean', 'min': 'mean', 'max': 'mean', 'spkts'
    : 'mean', 'srate': 'mean', 'drate': 'mean', 'weight':
    'sum'
}
```

Listing 1.6. Spectral metrics - parameters

-Timeseries extraction:

```
ts = time_series_extractor(df, 'stime', 's', features_list
    , sortby_list, groupby_list, aggregation_dict)
```

Listing 1.7. Extraction of time series - Example

-Extracting spectral metrics:

```
spectral_metrics_df = spectral_metrics_extractor(ts, stime
    , saddr, daddr, pkts, bytes, rate, lbl_category)
print(res)
```

Listing 1.8. Extraction of spectral metrics - Example

– Graph visualization example

```
import pandas as pd
df = pd.read_csv('data/ton_iot/ransomware-normal.csv')
print_graph(df, 'ip', 'label', 'src_ip', 'dst_ip',
    src_port', 'dst_port')
```

Listing 1.9. Upload data - Example

The output file is saved in */graph_representation/* directory as an *.html* extention file as shown in Fig. 4

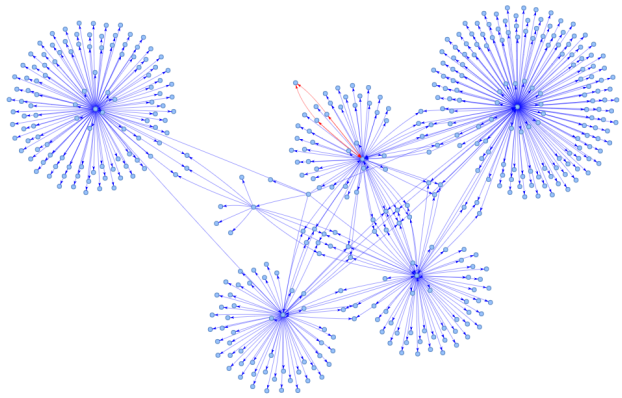
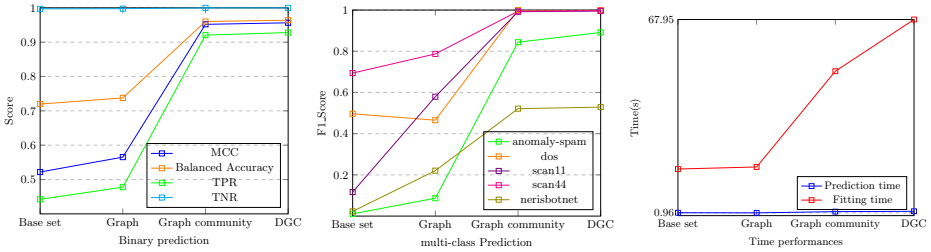


Fig. 4. Ransomware attack in Ton-IoT dataset presented via HTML using graphviz function that exist in GPML library

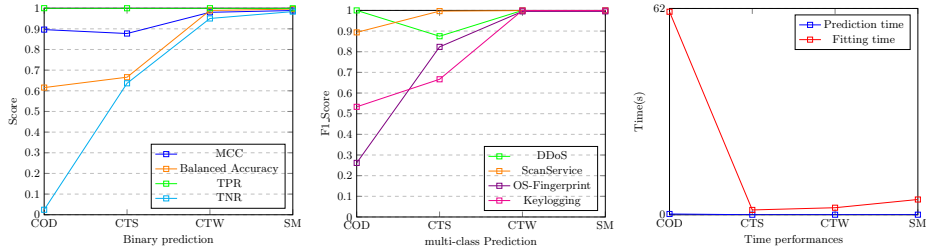
4.2 Evaluation and effectiveness

Fig. 5. Comparison of graph community approaches with baseline on UGR16 dataset with 5-folds evaluation using XGboost. Base set is original dataset feature space, the other one are the same dataset enriched with incrementally: graph metrics, graph community metrics and dynamic graphe community metrics.



The evaluation highlights the effectiveness of both community and spectral approaches in detecting network threats. The community-based method, referred to as DGC (Dynamic Graph Community) as shown in Fig 5, demonstrates improved metrics on the UGR16 dataset. For binary predictions, DGC shows Matthews Correlation Coefficient (MCC) of 0.96, Balanced Accuracy of 0.96 and True Positive Rate (TPR) of 0.93, surpassing the baseline with MCC of 0.52 and TPR of 0.44 , as well as graph and graph community models on a 5-folds evaluation. In multi-class predictions, DGC achieves high F1-scores, notably 0.89 for anomaly-spam and 0.99 for both scan44 and scan11 attacks, against respectively 0.01 for anomaly-spam, 0.69 for scan44 and 0.11 for scan11 in the baseline. Time performance analysis reveals prediction times from around 1s to 1.5s, though fitting times increase from 16.15s to 67.95s due to model complexity.

Fig. 6. Comparison of spectral graph approaches with baseline on Botnet dataset



We adopt four distinct approaches to classify network traffic data. The first approach, *Classification on Original Data-logs (COD)*, uses the raw dataset with minimal filtering, excluding only time and sequence-related features to avoid bias from temporal patterns. The second approach, *Classification on Time-Series (CTS)*, introduces a time-series transform mechanism to segment traffic logs data into series of data and extract basic quantitative features like packet counts and rates, aiming to capture evolving traffic behaviors without relying on topological structures. The third approach, *Classification on Time-Windowing (CTW)*, enhances CTS by aggregating data within each time window into new datasets labeled based on the presence of attacks, improving temporal context while still focusing on quantitative attributes. The fourth and final approach, *Spectral Metrics (SM)*, constructs graphs within each time window and splits them into two sub-windows to observe structural changes. It then extracts spectral metrics from Laplacian matrices of these subgraphs, providing topological insights and enabling a graph-based representation of dynamic network behavior for classification. These approaches are detailed in our previous work [3] for further details.

The spectral approach, represented as SM (Spectral Metrics) as shown in Fig 6, proves highly effective on the Botnet dataset. For binary predictions, SM reaches near-perfect metrics, including an MCC of 0.91 out of approximate of 0.9 for the COD, Balanced Accuracy of 0.97 out of 0.61 for COD, TPR of 0.93, and TNR of 0.99 out 0.01 for COD. In multi-class predictions, SM achieves excellent F1-scores, such as 0.99 for ScanService and 0.91 for DDoS attacks. Despite higher fitting times at 62s for COD, prediction times remain efficient across all configurations.

A comparison evaluation between $SPECTRA$, E-GraphSage and EN-GConv is performed for Botnet [4] and TonIoT [9] IoT datasets. The results for binary are shown in tables 2, and 3. The results for multi-classification are shown in Appendix in tables 4, 5. $SPECTRA$ significantly outperforms GCN for BotNet IoT in binary and multi-class detection for all attacks and outperforms them for multi-class detection in 7 cases out of 9 (when considering balanced accuracy) or 8 out of 9 (for F1-score) for TonIoT. For binary classification, it is competitive but second to E-GraphsSage.

Table 2. Comparison study between E-GraphSage, EN-GConv and SPECTRA over Botnet IoT dataset for binary classification

	E-GraphSage	EN-GConv	SPECTRA
F1 Score	0.9888	0.3322	**0.9944**
Balanced Acc	0.9796	0.8817	**0.9971**
MCC	0.7772	0.3892	**0.9936**
ADR (%)	97.86	97.14	**99.52**
Precision	**0.9993**	0.2004	0.9936

DGC and SM address advanced network security challenges through complementary strengths. DGC excels in analyzing large-scale networks by identifying clusters based on community-driven patterns, which helps detect anomalies like insider threats or advanced persistent threats. On the other hand, SM leverages mathematical rigor to uncover structural insights, identifying hidden substructures such as covert communication channels or botnets.

5 Impact

The software opens up avenues for addressing new research questions throughout the pipeline for attack detection. One area of exploration involves graph-based models. Researchers can compare various graph structures and complex network metrics, evaluating their utility in the context of cybersecurity. Additionally, a comparison between graph convolutional models (GCNs), where learning operates directly on graph data, and graph-derived features, where learning operates on tabular data, can reveal insights into their detection capabilities, time performance, complexity, and support for parallelization. A promising direction involves exploring how traditional graph metrics and knowledge graphs complement each other to improve detection methods. Traditional metrics—such as degree centrality, betweenness centrality, closeness, and clustering coefficient—help identify structural patterns, including influential nodes or anomalies. Knowledge graphs add contextual semantics, supporting accurate classification of entities and relationships. Combining both approaches strengthens the detection of complex threats and irregular behaviors.

For attack detectors, the software facilitates characterizing detectors for specific types of attacks, defining weak signal analysis to improve detection capabilities, and advancing feature engineering tailored to attack scenarios. Explainability remains a critical focus, providing transparency and interpretability in the detection process.

The software significantly enhances the pursuit of existing research questions in several ways. It enables the automated extraction of derived features, streamlining the process of preparing data for machine learning models. Similarly, it automates the computation of graph metrics, reducing manual effort and increasing efficiency. Moreover, the systematized visualization and evaluation framework provided by the software improves the ability to assess and interpret machine learning approaches for attack detection, enabling more robust and reproducible research outcomes.

6 Conclusions

The GPML (Graph Processing for Machine Learning) library provides a robust and scalable solution for addressing the challenges posed by advanced cyberthreats. By leveraging graph-based methodologies, it enables the detection of

complex attack patterns, community dynamics, and emerging anomalies in network environments. The library integrates community and spectral graph analysis with powerful Python tools, offering advanced functionalities for temporal graph processing, feature extraction, and visualization. These capabilities not only enhance threat detection but also support deeper exploration of network behaviors, making GPML a valuable resource for researchers and practitioners in cybersecurity. Future work can expand its use for real-time applications and AI-driven threat detection solutions, further advancing network security practices.

Acknowledgements

The authors acknowledge the support of the Région Grand-Est and École Pour l'Informatique et les Techniques Avancées (EPITA) for the joint funding of the XDGMed Project.

References

1. Tanzeela Altaf, Xu Wang, Wei Ni, Ren Ping Liu, and Robin Braun. Ne-gconv: A lightweight node edge graph convolutional network for intrusion detection. *Computers & Security*, 130, 2023.
2. Aric Hagberg and Drew Conway. Networkx: Network analysis with python. *URL: https://networkx.github.io*, 2020.
3. Majed Jaber, Nicolas Boutry, and Pierre Parrend. Graph-based spectral analysis for detecting cyber attacks. In *Proceedings of the 19th International Conference on Availability, Reliability and Security*, pages 1–14, 2024.
4. Nickolaos Koroniotis, Nour Moustafa, Elena Sitnikova, and Benjamin Turnbull. Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-iot dataset. *Future Generation Computer Systems*, 100:779–796, 2019.
5. Parameshwar Reddy Kothamali and Subrata Banik. Limitations of signature-based threat detection. *Revista de Inteligencia Artificial en Medicina*, 13(1):381–391, 2022.
6. Jielun Liu, Ghim Ping Ong, and Xiqun Chen. Graphsage-based traffic speed forecasting for segment network with sparse data. *IEEE Transactions on Intelligent Transportation Systems*, 23(3):1755–1766, 2020.
7. Wai Weng Lo, Siamak Layeghy, Mohanad Sarhan, Marcus Gallagher, and Marius Portmann. E-graphsage: A graph neural network based intrusion detection system for iot. In *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*, pages 1–9. IEEE, 2022.
8. Wes McKinney et al. pandas: a foundational python library for data analysis and statistics. *Python for high performance and scientific computing*, 14(9):1–9, 2011.
9. Nour Moustafa. A new distributed architecture for evaluating ai-based security systems at the edge: Network ton_iot datasets. *Sustainable Cities and Society*, 72, 2021.

10. Iqbal H Sarker, Helge Janicke, Mohamed Amine Ferrag, and Alsharif Abuadbba. Multi-aspect rule-based ai: Methods, taxonomy, challenges and directions toward automation, intelligence and transparent cybersecurity modeling for critical infrastructures. *Internet of Things*, 2024.

11. Shahroz Tariq, Mohan Baruwal Chhetri, Surya Nepal, and Cecile Paris. Alert fatigue in security operations centres: Research challenges and opportunities. *ACM Computing Surveys*, 2025.

12. M Uma and Ganapathi Padmavathi. A survey on various cyber attacks and their classification. *Int. J. Netw. Secur.*, 15(5):390–396, 2013.

13. Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

14. Jaewon Yang and Jure Leskovec. Defining and evaluating network communities based on ground-truth. In *Proceedings of the ACM SIGKDD Workshop on Mining Data Semantics*, MDS '12, New York, NY, USA, 2012. Association for Computing Machinery.

Table 3. Comparison study between E-GraphSage, EN-GConv and SPECTRA over TonIoT dataset for binary classification

	E-GraphSage	EN-GConv	SPECTRA
F1 Score	0.966	**0.9991**	0.9977
Balanced Acc	**0.9652**	0.8741	0.9493
MCC	**0.9623**	0.1765	0.9252
ADR (%)	93.43	99.82	**99.87**
Precision	**0.9999**	**0.9999**	0.9968

Table 4. Comparison study between E-GraphSage and SPECTRA over TonIoT dataset for multi-class classification

		E-GraphSage	SPECTRA
DDoS	F1 Score	0.9823	**0.9944**
	Balanced Acc	90.839	**0.9999**
	MCC	0.9760	**0.9944**
	ADR (%)	0.9694	**100**
	Precision	**0.9956**	0.9889
DoS	F1 Score	0.7304	**1**
	Balanced Acc	0.9220	**1**
	MCC	0.7006	**1**
	ADR (%)	0.9608	**100**
	Precision	0.5892	**1**
Scanning	F1 Score	0.8549	**1**
	Balanced Acc	0.8755	**1**
	MCC	0.8125	**1**
	ADR (%)	0.7584	**100**
	Precision	0.9795	**1**
Ransomware	F1 Score	0.9410	**0.9649**
	Balanced Acc	**0.9907**	0.9793
	MCC	90.395	**0.9648**
	ADR (%)	**0.9855**	95.88
	Precision	0.9003	**0.9711**
SQL Injection	F1 Score	0.8282	**0.9933**
	Balanced Acc	0.9420	**0.9962**
	MCC	0.8264	**0.9933**
	ADR (%)	0.8894	**99.24**
	Precision	0.7749	**0.9943**
Password	F1 Score	0.9115	**0.9869**
	Balanced Acc	0.9437	**0.9952**
	MCC	0.9062	**0.9867**
	ADR (%)	0.8915	**99.07**
	Precision	0.9324	**0.9832**
XSS	F1 Score	**0.9463**	0.9285
	Balanced Acc	**0.959**	0.9521
	MCC	**0.9412**	0.9287
	ADR (%)	**0.9208**	90.43
	Precision	**0.9732**	0.9541
Backdoor	F1 Score	0.0787	**0.9944**
	Balanced Acc	0.5226	**0.9972**
	MCC	0.0839	**0.9942**
	ADR (%)	0.0506	**99.46**
	Precision	0.1771	**0.9942**
MitM	F1 Score	0.1752	**0.9924**
	Balanced Acc	0.935	**0.9925**
	MCC	0.2909	**0.9925**
	ADR (%)	0.8743	**98.5**
	Precision	0.0973	**1**

Table 5. Comparison study between E-GraphSage and SPECTRA over Botnet IoT dataset for multi-class classification

	%	E-GraphSage	SPECTRA
DDoS	F1 Score	0.9999	**1**
	Balanced Acc	0.9997	**1**
	MCC	0.9995	**1**
	ADR (%)	100	**100**
	Precision	0.9999	**1**
ScanService	F1 Score	0.9321	**0.9986**
	Balanced Acc	0.9374	**0.9817**
	MCC	0.9267	**0.9697**
	ADR (%)	87.52	**99.89**
	Precision	0.9968	**0.9982**
OS Fingerprint	F1 Score	0.7926	**0.9953**
	Balanced Acc	0.9865	**0.9953**
	MCC	0.8025	**0.9952**
	ADR (%)	98.69	**99.07**
	Precision	0.6623	**1**
Keylogging	F1 Score	0.8224	**1**
	Balanced Acc	1	**1**
	MCC	0.8357	**1**
	ADR (%)	100	**100**
	Precision	0.6984	**1**