EAI Endorsed Transactions
on Industrial Networks and Intelligent Systems
Research Article EAI.EU

Scalable object-relational modeling for synthesizing multi-format visual analytics of STIX-based cyber threat intelligence data

Muhammed Onur Kaya[1], Resul Das[1,2]

[1]Firat University, Technology Faculty, Department of Software Engineering, 23119 Elazig, Türkiye
[2]Edinburgh Napier University, School of Computing, Engineering & The Built Environment, Department of Cyber Security and System Engineering, Edinburgh, Scotland, United Kingdom

Abstract

The observed increase in cyber threat intelligence data, the diversity of sources, and relational complexity make it difficult for analysts to directly assess and interpret threat profiles from raw Structured Threat Information Expression (STIX) packages. This study utilises an object-relational model to transform all object and relationship types into a single unified representation and reconstruct the same graph model across three different visualisation software packages (plotly, matplotlib, pyvis). The proposed analytical framework generates extended threat models by combining multiple STIX datasets through entity extraction, creating observable entities from each dataset, and completing relationships among them; thus mapping relationship types to the operational meanings of attacks and enabling more comprehensive risk prioritisation. The case study is based on a public OASIS STIX example package. The enriched graph is dominated by relationship types such as uses, indicates, pattern-refers-to, and attributed-to, and the highest risk scores are assigned to the nodes labelled "Privilege Escalation" and "Ugly Gorilla". Scalability tests performed on 1000-node synthetic directed graphs revealed a processing time of 8.08 seconds, memory consumption of 124.3 MB, and a frame rate of 215.05 fps during interactive preview. These findings demonstrate that the proposed framework offers a unified and viable foundation for visual analysis, risk prioritisation, and scalable analytical reporting of STIX-based cyber threat intelligence.

Received on 24 April 2026; accepted on 15 June 2026; published on 18 June 2026

Keywords: STIX; Cyber Threat Intelligence; Graph Visualisation; Threat Graph; Scalability

doi:10.4108/eetinis.132.12774

1. Introduction

The global spread of the internet and the increasing complexity of network infrastructures have led to a significant increase in both the quality and quantity of cybersecurity threats. Today, network systems undertake critical functions at the corporate, industrial, and individual levels, and ensuring the security of these systems has become a strategic priority at national and international levels. Cyberattacks carried out in this environment exhibit patterns that are difficult to detect, analyse, and interpret due to factors such as high-volume traffic data, heterogeneous attack vectors, and Advanced Persistent Threat (APT)s [1]. Therefore, the effective use of Intrusion Detection System (IDS)/Intrusion Prevention System (IPS) is a critical requirement for monitoring network anomalies in real time and systematically modelling attack behaviours. On the other hand, the large volume and multi-dimensional nature of the cyber threat intelligence data that these systems must process create significant methodological difficulties in terms of both computational complexity and visualisation [2].

Cyber threat intelligence data is the raw or modelled documentation of cyberattacks targeting the IT networks of an organisation or institution [3]. One common way to define attack scenarios is the Structured Threat Information Expression (STIX)

*Corresponding author. E-mail: r.das@napier.ac.uk

structure [4]. STIX is a standard developed by MITRE Corporation (MITRE) to represent attack objects and the relationships between these objects. Scenarios defined with STIX mainly fall into two families: STIX 1.x scenarios are mostly in Extensible Markup Language (XML) file format; STIX 2.x scenarios can be in both JavaScript Object Notation (JSON) and XML file formats [5]. Data read from these files can be used to deduce objects and their relationships; scenarios are built upon these components [6]. Visualisation and analysis of these scenarios can be performed in Python and its related ecosystem [7].

1.1. Problem Statement

Scaled Cyber Threat Intelligence (CTI) data needs to be transformed into structures that analysts can examine. Since raw STIX packages contain numerous object types, cross-references, and sometimes indirect relational fields, simply opening the file is not enough to reveal the threat image; consistently resolving the relationships and reducing them to a graphical model is a separate engineering challenge. Static or non-interactive views limit exploration; examining the same scenario at different scales, comparing subgraphs, and producing reportable output all require an interactive and reproducible visualisation pipeline [8]. The inability to transfer STIX 2.x JSON content directly into graph libraries creates the need to parse objects and standardise the node–edge abstraction; otherwise, separate and error-prone custom transformations must be written for each tool. Furthermore, presenting file selection, preview, and export steps together within a desktop interface for both training and operational use is a practical requirement so that non-technical users can also follow the scenario. This study targets the following gaps: (i) reading STIX 2.x packages, (ii) separating STIX Domain Object (SDO) and STIX Relationship Object (SRO) objects, (iii) mapping nodes and edges to a common internal representation, (iv) rendering the same graph abstraction comparatively with matplotlib, plotly, and pyvis, and presenting it within a Tkinter-based layout [9].

1.2. Scope and Research Positioning

Representing STIX bundles as graphs is a well-established practice, and the present study does not claim to introduce a fundamentally new graph-theoretic formalism or a new visualisation library. The contribution is instead an integrated, reproducible STIX 2.x reference pipeline that (i) defines a single canonical node–edge abstraction for heterogeneous objects and relationships, (ii) performs STIX-compliant enrichment (merging multiple bundles, extracting observables from indicator patterns, and completing resolvable relationships), (iii) overlays operational detection semantics and risk-oriented ranking on the same graph, and (iv) renders the *same* intermediate graph comparatively across matplotlib, plotly, and pyvis within one workflow—addressing the practical need, identified in Table 1 and in the problem statement above, for side-by-side, multi-format analyst reporting rather than tool-specific ad hoc transforms. The term *object-relational* is used here in the sense of unifying SDO and SRO instances under one internal representation, not as a claim of a new database model.

1.3. Main Contributions

Within the framework of the problem outlined in the previous subsection, the aim is to provide an integrated solution that extends from the parsing of STIX 2.x data to multiple backends, rather than focusing solely on a single visualisation component. The contributions summarised below are positioned through both methodological clarity (clear pipeline and algorithmic definitions) and application-level reusability and analyst-focused interaction. The contribution is positioned as a reproducible reference pipeline and analyst-support workflow rather than a new threat-reasoning or learned detection-intelligence model. In summary, the original contributions of this study to the literature are as follows:

- An end-to-end visualisation and analysis pipeline focused on STIX 2.x is offered: object parsing, SDO/SRO separation, standard node-edge abstraction, and multiple visualisation backends (matplotlib/plotly/pyvis) are combined into a single reproducible workflow.
- The study proposes not only a visualisation of raw sample packages but also an enrichment step that preserves STIX compliance: combining multiple STIX sources, extracting observable entities (e.g., ipv4-addr/domain-name) from within the pattern, and performing relationship completion yields an extended threat diagram.
- The developed detection layer maps STIX relationship types to operational attack semantics (e.g., IOC Match, C2 Beaconing, Phishing Delivery, Privilege Escalation); thus, the graph is transformed from a purely descriptive output into a risk network that serves analyst prioritisation.
- With the integrated dashboard approach, relationship distribution, node type composition, type-to-type density matrix, and risk ranking are reported together on the same experimental plane; this design combines topological interpretation with decision-support metrics in a single STIX-based analytical framework.

Table 1. Summary and positioning of related CTI visualisation and graph-based studies

Work	Context / data	Method / tool	Main contribution	Distinction from this study
[10]	STIX 2.x visual analytics; STIX 2 JSON events	KAVAS framework; graph DB; web interface	Knowledge-assisted interactive CTI examination and enrichment for analysts	Depends on a persistent graph DB; no comparative matplotlib/plotly/pyvis export from one canonical graph G
[11]	STIX 2.x and dynamic graphs; OSINT, traffic, deep web	Dynamic graph structures; Python pipeline	End-to-end CTI visual analysis with temporal threat tracking	Emphasises evolution over time; no side-by-side static and interactive library benchmark
[12]	CSKG concepts and applications; multi-source cyber data	KG schema and construction review; 9 categories / 18 subcategories	Classification framework for CTI mining and attack/defence scenarios	Conceptual survey; no reproducible STIX 2.x Python visualisation pipeline
[1]	Honeynet attack visualisation; real attack records	Attack-graph model; Cyber Kill Chain alignment; GDS	Botnet detection and attacker-link analysis on real honeynet data	Does not use STIX as input; lacks a reproducible STIX-oriented Python workflow
[13]	Survey of cybersecurity knowledge graphs; multi-source data	Graph-based models; ontology; ML/inference	Holistic review of situational awareness, attack-path visualisation, and CKG inference	Survey-level synthesis; no applied parse-once/render-many STIX pipeline
[3]	Systematic CTI review; 294 studies (2019–2023)	SLR + PRISMA; knowledge base; detection models; dashboards	Multilayer organisational CTI framework for cyber resilience	Framework-oriented; no STIX bundle parsing or comparative graph visualisation
[14]	CKG construction; CTI, CVE, ATT&CK	Ontology- and NLP-based construction techniques	Comprehensive comparison of CKG construction approaches and scenarios	Technology survey; limited hands-on visualisation and implementation detail
[15]	SLR on KG and semantic-web tools in CTI	Ontology + KG + ML/DL integration review	Synthesis of interoperability solutions among security operations centres	Ontology/KG focus; no practical multi-backend STIX visualisation workflow
[16]	Hybrid deep-learning visual analytics; UNSW-NB15 traffic	GAT; SigmaJS; homogeneous/heterogeneous graphs	Joint attack prediction and interactive JavaScript-based graph presentation	Network-traffic graphs; does not ingest standard STIX 2.x JSON bundles
[17]	Strategic attack graphs from CTI reports; APT text	HAG framework; joint inference; hypergraph edges	Tactical concepts represented intuitively via hypergraph attack chains	Report-derived hypergraphs; no direct reproducible rendering of STIX 2.x bundle graphs
[18]	CTI knowledge-graph construction; DNRTI-STIX2, APT reports	TiKG: SecureBERT + BiLSTM + CRF extraction pipeline	Domain-focused entity/relation extraction with reduced terminology error propagation	Extraction-centric; does not visualise graphs directly from STIX 2.x JSON bundles
This study	STIX 2.x multi-library Python visualisation; OASIS public bundles	NetworkX / igraph / pyvis; matplotlib / plotly; Tkinter	SDO/SRO parsing, canonical node–edge model, STIX-compliant enrichment, detection overlay, and comparative multi-format reporting	Integrative reference pipeline: parse-once/render-many workflow with rule-based risk ranking and reported scalability

- The scalability assessment is systematically presented using measurements of time, memory, and interaction latency on synthetic directed graphs; thus, quantitatively demonstrating not only the visual accuracy of the method but also its practical applicability and computational cost.

The remainder of the paper is structured as follows: Section 2 summarizes related work in the literature and frames the position of the proposed approach against existing methods; Section 3 systematically presents the dataset, the libraries used, and the algorithmic components of the object parsing and visualisation pipeline; Section 4 reports the findings in the context of experimental results and discussion; and Section 5 summarizes the essence of the study and identifies directions for future research.

2. Literature Review

Numerous academic studies exist in the literature regarding the visualisation of cyber intelligence data. Table 1 summarises eleven representative studies together with the present work in terms of context, method, contribution, and distinction from the proposed pipeline. [10] proposed the KAVAS framework, which combines semi-structured threat intelligence data based on STIX 2 with an interactive visual analytics component; it presents a graph-oriented approach where security experts can examine and enrich threats. In another study, [11] presents an end-to-end pipeline using dynamic graph structures for analyzing cyber threat intelligence data through graph visualisation, and STIX 2 packages are used as the primary data source for this process.

[1] proposed an attack graph model based on node (entity) and edge (action) separation by aligning real honeynet attack data with the Cyber Kill Chain framework; they applied graph data science algorithms to

this model for botnet detection and uncovering attacker connections. While the approach makes significant contributions to practical attack analysis, it does not use the STIX standard as an input format and does not include a reproducible Python pipeline. In this vein, a more recent study by [16] proposed a hybrid approach that combines an Graph Attention Network (GAT)-based model with JavaScript-based visualisation tools to both predict and interactively visualise cyberattacks; in this study, heterogeneous network traffic data was processed by transforming it into homogeneous and heterogeneous graph structures. In the field of cybersecurity, knowledge graphs are assuming an increasingly central role. [13] conducted a comprehensive review of security knowledge graphs that represent cyber knowledge through a graph-based data model, demonstrating that such graphs offer holistic approaches for threat intelligence acquisition, network visualisation, and understanding attack paths. [14] contributed a comprehensive survey to the literature by addressing the core technologies and scenario-based applications of Cybersecurity Knowledge Graph (CKG) construction. [12] provided a general overview of the knowledge graph concept, its schema, and construction methods, and carried out a detailed evaluation of relevant datasets and open-source frameworks.

Work on deriving structured attack graphs from CTI reports also constitutes an important area of research. [17] proposed the Hyper Attack Graph (HAG) framework, which identifies entities and relationships in CTI reports using common inference techniques and transforms this information into a hypergraph structure to generate attack chains at a strategic level; it has been demonstrated that the edges of the hypergraph represent tactical concepts more intuitively. [18], on the other hand, has presented the Threat Intelligence Knowledge Graph Pipeline (TiKG) pipeline, which combines a domain-specific transformer model with an attention-based Bidirectional Long Short-Term Memory (BiLSTM) to extract entities and relationships from CTI reports; this has significantly reduced error propagation arising from the nuances of cybersecurity terminology.

Extensive survey studies on the extraction and sharing of threat intelligence have also made significant contributions to the literature. [3], meanwhile, conducted a systematic literature review focused on CTI and proposed a comprehensive framework comprising a knowledge base, detection models and visualisation dashboards for CTI implementations within organisations. In terms of semantic web tools and ontology-based approaches, [15] presents a review that systematically examines knowledge graphs and semantic web tools in the CTI field; it demonstrates how ontology- and Knowledge Graph (KG)-based solutions enhance interoperability between security operations centres.

Whilst previous studies have made significant progress, gaps remain in terms of combining lightweight Python frameworks, standard node and edge extraction from STIX 2.x, and side-by-side visualisation using common libraries (igraph, NetworkX, pyvis, matplotlib, plotly). The proposed approach focuses on these areas.

Relative to prior STIX-oriented visual analytics, the present pipeline differs along axes that Table 1 summarises only partially. KAVAS [10] couples STIX 2 ingestion with a graph database and web interface for collaborative enrichment, but does not offer comparative matplotlib/plotly/pyvis export from one canonical graph G. Dynamic-graph CTI pipelines [11] emphasise temporal evolution rather than side-by-side static and interactive library backends. Knowledge-graph and hypergraph attack-chain methods [13, 17] infer strategic relations from unstructured reports instead of directly rendering standard STIX 2.x bundles. Operational platforms MISP [19] and OpenCTI [20] prioritise feed sharing, correlation, and enterprise knowledge-graph exploration; they complement, rather than replace, a lightweight reproducible parse-once/render-many Python workflow with enrichment, rule-based detection overlay, and reported scalability. Global graph-analysis metrics in attack-graph studies [1, 16] address complementary analytical goals and are not the primary contribution of this work.

3. Materials and Methods

This section describes the implementation of the STIX 2.x visualisation pipeline. Unlike recent work on anomaly detection using Graph Convolutional Network (GCN)–Long Short-Term Memory (LSTM) hybrid architectures on graphs derived from dynamic traffic [21], this approach takes the STIX JSON package as input and defines the output in a way that serves analyst exploration via multi-visualisation backends. Figure 1 illustrates the main stages: file reading, object parsing, standardisation of nodes and edges (Algorithms 1–3), graph construction using igraph, NetworkX and pyvis.network, and visualisation using matplotlib, plotly and pyvis. The object list and parsing, along with the node-edge records, are formally specified as Equations (1) to (4) respectively. Figure 2, meanwhile, schematically illustrates how standard node and edge attributes are mapped to graph libraries and the visualisation layer by summarising the data mapping dimension of the same pipeline. The process involves, in sequence: reading the file; importing the JSON file generated by STIX 2.x; parsing the objects into relationship objects and node (domain) objects according to their types; definition in standard format using Algorithm 2 and Algorithm 3; and the addition of the defined objects to the graph using network

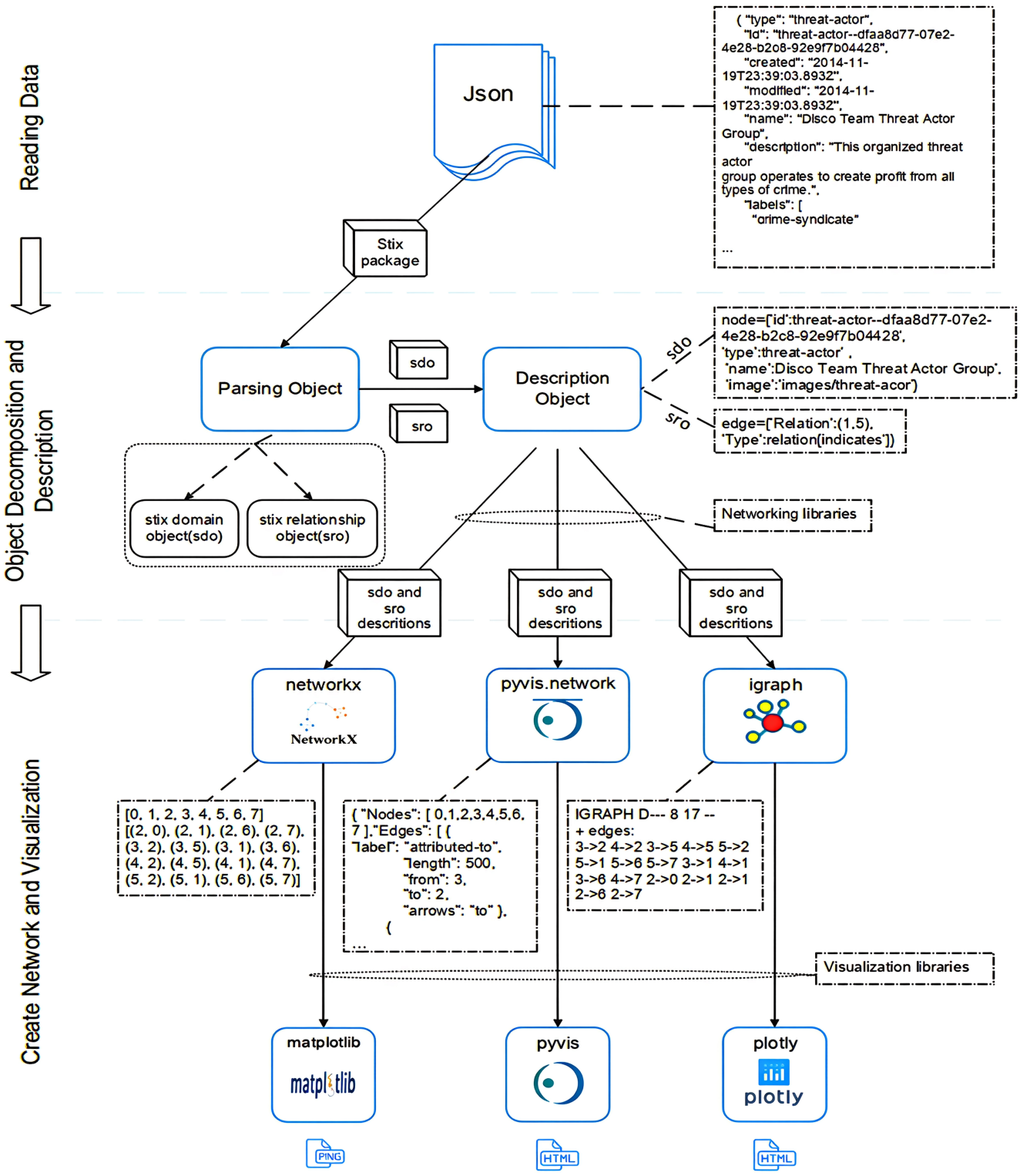


Figure 1. Flowchart of the processing steps for the implemented graph visualisation pipeline

generation libraries, followed by the visualisation of the resulting network data.

3.1. Datasets

The experimental evaluation, summarised in Section 4 is based on a threat report package taken from the official STIX 2.x JSON example repository published by the OASIS CTI Technical Committee [22]. The file path used (relative to the repository root) is examples/threat-reports/apt1.json; a working copy is provided under the name data/oasis_threat_report_apt1.json; the root structure consists of a 'type: bundle' and an 'objects' array. This package contains a total of 76 objects (30 relationships, 46 domain objects); the type distribution is summarised as follows: indicator, tool, attack-pattern, malware, threat-actor, identity, intrusion-set, report. Following parsing, the graph yields 46 nodes and, to the extent that the package can be resolved internally, 30 directed edges (all source_ref/target_ref endpoints point to objects present in the package). The data is public sample content and does not contain any confidential operational data. The Disco Team identifier in Table 2 is consistent with the identifier pattern used in the same OASIS sample tradition.

The list of objects is processed in accordance with the Algorithm 1–3; the matplotlib plots shown in the figures are based on a graph derived from this official package (which can be reproduced using the cti_figures_from_bundle.py script). Reading, processing and constructing a network from a STIX 2.x JSON file in Python is carried out using the steps summarised in Figure 1.

Table 2. Representative key–value slice for an identity object from common STIX 2.x sample bundles (summary)

Key	Value
type	identity
id	identity–733c5838-34d9-4fbf-949c-62aba761184c
created	2016-08-23T18:05:49.307Z
modified	2016-08-23T18:05:49.307Z
name	Disco Team
description	Organised threat actor crime syndicate
identity_class	organization
contact_information	sco-team@stealthemail.com

3.2. STIX Bundle Ingestion and Object Separation

Cyber intelligence data is typically available in XML format for STIX 1.x and in JSON format for STIX 2.x [4]. In this study, JSON files are read using Python's standard json module; the data is stored in memory as a dictionary. STIX 2.x scenario data arrives in a single bundle of the form $B = \langle T_B, \mathcal{O} \rangle$, where $T_B \equiv$ bundle [22]. When the file is read, the entire scenario is loaded into memory and the bundle objects are accessed:

$$\mathcal{O}(B) \triangleq B[\text{objects}], \qquad |\mathcal{O}| = |\mathcal{O}_{\text{dom}}| + |\mathcal{O}_{\text{rel}}| \tag{1}$$

where $\mathcal{O}(B)$ is the finite set containing all STIX objects in bundle B, and $|\cdot|$ denotes the cardinality. As shown in Equation (1), this arrangement is consistent with the STIX Bundle / objects component in Figure 2.

Objects must be categorised as SRO and SDO. Let $\tau : \mathcal{O} \to \mathcal{T}$ be the type access function; for each $o \in \mathcal{O}$, if the $\tau(o) = \text{type}(o)$ field is "relationship", the object is classified as a relationship (edge candidate), otherwise, it is classified as a domain object (node candidate):

$$\begin{aligned} \mathcal{O}_{\text{rel}} &\triangleq \left\{ o \in \mathcal{O} \mid \tau(o) = \text{relationship} \right\}, \\ \mathcal{O}_{\text{dom}} &\triangleq \mathcal{O} \setminus \mathcal{O}_{\text{rel}}, \\ \mathcal{O} &= \mathcal{O}_{\text{dom}} \dot{\cup} \mathcal{O}_{\text{rel}}. \end{aligned} \tag{2}$$

As shown in Equation (2), Algorithm 1 performs the same separation using list operations.

Algorithm 1 Splitting STIX objects into domain-object and relationship lists

Require: Bundle dictionary D in memory
Ensure: $\mathcal{O}_{\text{dom}}, \mathcal{O}_{\text{rel}}$ lists
1: $\mathcal{O} \leftarrow D[\text{objects}]$
2: $\mathcal{O}_{\text{rel}} \leftarrow \emptyset, \mathcal{O}_{\text{dom}} \leftarrow \emptyset$
3: **for** each $o \in \mathcal{O}$ **do**
4: **if** type(o) = relationship **then**
5: Add o to the $\mathcal{O}_{\text{rel}}$ list
6: **else**
7: Add o to the $\mathcal{O}_{\text{dom}}$ list
8: **end if**
9: **end for**

At the end of this step, SDO node candidates and SRO edge candidates are obtained as lists; Figure 2 illustrates these two aspects through the node and edge panels, respectively [23].

3.3. Canonical Node–Edge Abstraction and Graph Model

A standard node record is created for each domain object (Algorithm 2). Defining the function $\nu : \mathcal{O}_{\text{dom}} \to \mathcal{N}$, the record is modelled as a quadruple corresponding to the fields summarised on the node side in Figure 2:

$$\nu(o) = \left\langle \text{id}(o),\ \tau(o),\ \text{name}(o),\ \text{img}(o) \right\rangle, \quad o \in \mathcal{O}_{\text{dom}} \tag{3}$$

As shown in Equation (3), display naming for types such as marking-definition, sighting, and observed-data is handled specifically; the icon path is assigned via GetImagePath(name(o)).

Figure 2 schematically summarises how the standard node and edge records created after parsing objects in

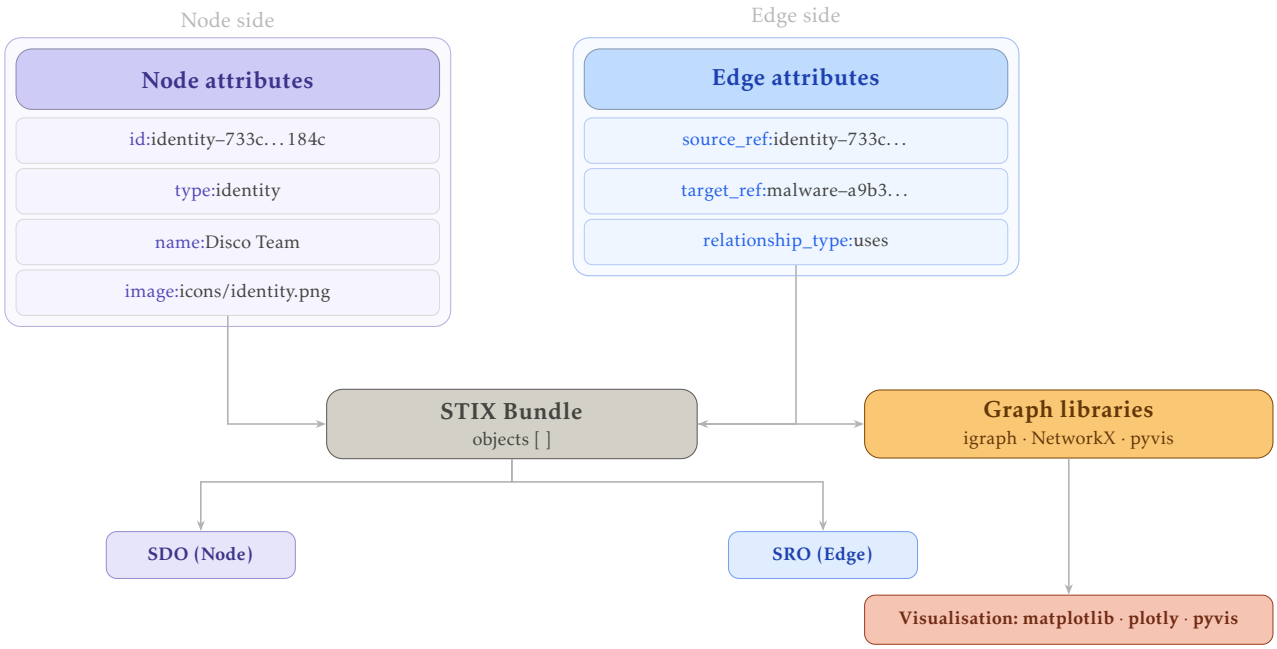


Figure 2. Standard node and edge attributes passed to graph libraries

Algorithm 2 Building the standard node list from STIX domain objects

Require: $\mathcal{O}_{\text{dom}}$ list
Ensure: node list $\mathcal{V}$
$\mathcal{V} \leftarrow \emptyset$
for each $o \in \mathcal{O}_{\text{dom}}$ **do**
  $v \leftarrow$ empty node record
  $\text{id}(v) \leftarrow \text{id}(o)$, $\text{type}(v) \leftarrow \text{type}(o)$
  if $\text{type}(o) \notin \{\text{marking-definition}, \text{sighting}\}$ **then**
    $\text{name}(v) \leftarrow \text{name}(o)$
  else
    $\text{name}(v) \leftarrow$ composite name appropriate for the type label
  end if
  $\text{image}(v) \leftarrow \text{GetImagePath}(\text{name}(v))$
  Add v to the $\mathcal{V}$ list
end for
return $\mathcal{V}$

the STIX package are mapped to graph libraries and visualisation tools: on the node side, id, type, name and image; on the edge side, source_ref, target_ref and relationship_type. The figure emphasises data mapping rather than reflecting the sequence of steps executed in the code line by line; formal definitions are provided via Equations (1) to (4), whilst the end-to-end process flow is presented via Figure 1 and Algorithm 1–3. Relationship objects point to source and target objects via source_ref and target_ref; these fields are located as edge attributes in Figure 2 [24]. Let $\phi : \text{id}(o) \mapsto \text{idx}_{\mathcal{V}}$ be the id-to-index mapping function, and let $\sigma(r)$, $\upsilon(r)$, and $\rho(r)$ denote source_ref, target_ref, and relationship_type, respectively. The standard edge triplet for each $r \in \mathcal{O}_{\text{rel}}$ is defined by the function $\varepsilon : \mathcal{O}_{\text{rel}} \to \mathcal{E}$ as:

$$\varepsilon(r) = \Big\langle \phi(\sigma(r)),\ \phi(\upsilon(r)),\ \rho(r) \Big\rangle \tag{4}$$

As shown in Equation (4), here $\sigma(r)$, $\upsilon(r)$, and $\rho(r)$ correspond to the source_ref, target_ref, and relationship_type fields, while ϕ corresponds to the graph node index associated with the id values of the nodes created in Equation (3). Algorithm 3 generates an edge list in the form of (4) by finding identity matches within $\mathcal{V}$. Separate rules and algorithms have also been implemented in the project for indirect relationships such as report, observation, and marking references (e.g., created_by_ref, object_marking_refs, object_refs, observed_data_refs).

The steps above formally define a directed labelled graph [25]. Let $\lambda : \mathcal{V} \cup \mathcal{E} \to \mathcal{T}$ be the type labeling function; the resulting graph is summarized as shown in Equation (5):

$$G = \big(\mathcal{V},\ \mathcal{E},\ \lambda\big), \quad \mathcal{V} = \Big\{ \upsilon(o) \mid o \in \mathcal{O}_{\text{dom}} \Big\}, \quad \mathcal{E} \subseteq \Big\{ \varepsilon(r) \mid r \in \mathcal{O}_{\text{rel}} \Big\} \tag{5}$$

While $|\mathcal{V}| = |\mathcal{O}_{\text{dom}}|$, it holds that $|\mathcal{E}| \le |\mathcal{O}_{\text{rel}}|$ because an edge is excluded if ϕ remains undefined. In the experimental dataset, $|\mathcal{V}| = 46$ and $|\mathcal{E}| = 30$ were obtained [22].

Algorithm 3 Building edges from STIX relationship objects

Require: $\mathcal{V}$ and $\mathcal{O}_{\text{rel}}$ lists
Ensure: edge list $\mathcal{E}$
1: $\mathcal{E} \leftarrow \emptyset$
2: **for** each $r \in \mathcal{O}_{\text{rel}}$ **do**
3: Find indices in $\mathcal{V}$ for $\phi(\sigma(r))$ and $\phi(\upsilon(r))$
4: **if** both indices are defined **then**
5: Add the triplet $\varepsilon(r)$ to the $\mathcal{E}$ list according to (4)
6: **end if**
7: **end for**
8: **return** $\mathcal{E}$

3.4. Libraries and Multi-Backend Visualisation

The graph visualisation approach was implemented using Python [26]. Python supports the dictionary data structure and offers flexibility in data processing and visualisation; the dictionary structure facilitates object identification and relationship maintenance within attack-scenario data. Among graph construction libraries compatible with Python 3, igraph, NetworkX and pyvis.network stand out [27]; the same logical graph can be stored in memory using different internal structures. Three open-source visualisation libraries (matplotlib, plotly, and pyvis) were used [28, 29]. Matplotlib provides stable static plots; plotly and pyvis offer JavaScript-based interactivity and two- or three-dimensional views. In the example application, Tkinter was used for desktop control; PyQt and Kivy are available as alternatives. Library selection was based on availability and widespread familiarity; version compatibility with Python 3 was verified.

Supporting three visualisation backends within one workflow is a deliberate methodological choice rather than redundant tooling: each backend addresses a distinct analyst deliverable—print-ready reporting (matplotlib), shareable browser exploration with hover semantics (plotly), and local cluster tracing with physics-based rearrangement (pyvis)—while all consume the same canonical node–edge graph G produced by Algorithms 1–3. Table 4 summarises measured export latency on the OASIS apt1.json case study and workflow-level proxies that quantify the efficiency gain of parse-once, render-many fan-out relative to maintaining separate STIX-to-graph transforms per library.

3.5. Operational Detection Mapping and Risk Ranking

The enriched graph $G = (\mathcal{V}, \mathcal{E}, \lambda)$ is further annotated with analyst-oriented detection labels on edges and a node-level risk score. Mapping is deterministic and rule-based: each directed edge $e = (u, v)$ with

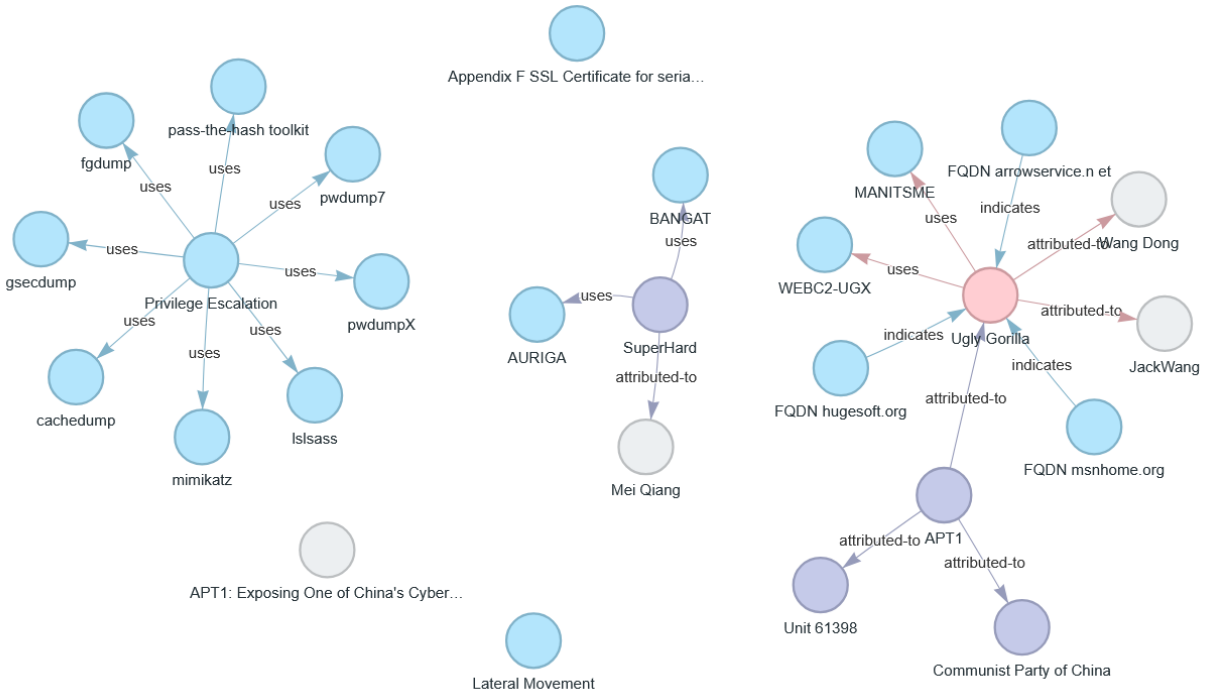


Figure 3. Interactive Pyvis view of the OASIS apt1.json graph

Table 3. Rule-based mapping of STIX edges to detection semantics

Condition on $\rho(e)$	Endpoint constraint	Label check (if any)	$\delta(e)$
`indicates`, `pattern-refers-to`	$\tau(v) \in$ {`ipv4-addr`, `domain-name`}	–	Indicator of Compromise (IOC) Match
`communicates-with`, `controls`	$\tau(v) \in$ {`infrastructure`, `ipv4-addr`, `domain-name`}	–	Command and Control (C2) Beaconing
`uses`	–	"phish" $\in$ name(u) $\cup$ name(v) or $\tau(u) =$ `campaign`	Phishing Delivery
`uses`	$\tau(u) \in$ {`malware`, `attack-pattern`}	"privilege" or "escalation" in labels	Privilege Escalation
`mitigates`	–	–	Mitigation Action
(other)	–	–	Context Link

relationship $\rho(e) = \lambda(e)$ and endpoint types $\tau(u), \tau(v)$ is assigned a detection class $\delta(e) \in \Delta$ according to Table 3. Rules combine STIX relationship semantics with endpoint object types; where several uses edges are indistinguishable by type alone, display-name substring checks (e.g. "privilege", "escalation", "phish") disambiguate attack-pattern labels such as Privilege Escalation or Phishing Delivery.

Let $\mathcal{H} \subset \mathcal{T}$ denote high-salience relationship types (uses, indicates, pattern-refers-to, communicates-with, controls, consists-of), $\mathcal{A} \subset \mathcal{T}$ threat-entity types (threat-actor, malware, attack-pattern), and $\mathcal{O} \subset \mathcal{T}$ observable/infrastructure types (ipv4-addr, domain-name, infrastructure). The edge weight is

$$w(e) = \omega_0 + \alpha\, \mathbb{I}\big[\rho(e) \in \mathcal{H}\big] + \beta\, \mathbb{I}\big[\tau(u) \in \mathcal{A}\big] + \gamma\, \mathbb{I}\big[\tau(v) \in \mathcal{O}\big] \quad (6)$$

with $(\omega_0, \alpha, \beta, \gamma) = (0.8, 1.3, 0.8, 1.0)$, emphasising attacker-to-observable chains over generic related-to links. The node risk score aggregates outbound and inbound edge salience:

$$R(n) = \sum_{e=(n,\cdot)\in\mathcal{E}} w(e) + \delta \sum_{e=(\cdot,n)\in\mathcal{E}} w(e), \qquad \delta = 0.35 \quad (7)$$

Nodes are ranked by descending $R(n)$. Equation (7) is a semi-local, additive score: outbound edges receive full weight (source-side attribution), inbound edges a discounted share (downstream observables). This design favours interpretability and alignment with STIX relationship semantics over global metrics such as PageRank or betweenness [1]; it is intended for analyst prioritisation, not as a learned detection model.

4. Experimental Results and Discussion

In data science and academic presentations, visualising information as a graph enhances clarity. In this study, the apt1.json package from the OASIS CTI Technical Committee example repository was used as the evaluation dataset [22]. After the objects were parsed using Algorithm 1, node records were created using Algorithm 2 and edge records using Algorithm 3; thus yielding a Figure 1-compliant JSON → node/edge abstraction → NetworkX graph chain. The experimental presentation proceeds in three steps: a representative view of the layered (multipartite) layout in a double-column width at the top of the same page, and immediately below it, a single-column Pyvis screenshot (Figures 3 and 4); followed by a collage of the actual apt1.json graph in three different static layouts using Matplotlib (Figure 5).

Figure 3 summarises how the same NetworkX abstraction is presented in the web-based layer; it appears on the same page as Figure 4, immediately below it in a single-column layout. The view is organised with several local clusters selected: for example, 'uses' relationships around privilege escalation tools, 'attributed-to' chains between malware and threat actor nodes, and 'indicates' arrows leading to infrastructure indicators are clearly distinguished. In the interactive version, node positions can be changed through user interaction; the aim is to enable rapid exploration and tracing of relationships prior to static reporting.

Figure 5 presents the study's key visual finding: when the layout algorithm is changed for the same data, the network structures that become visible differ markedly. The graph is a sparse directed network ($|V| = 46$, $|E| = 30$, density ≈ 0.0145) and exhibits a distinct weakly connected component structure (component sizes are 11, 9, 7, 4, 3, 2 and isolated nodes, respectively). In panel (a), the hub-centred spring layout clearly reveals edge clustering and local cores around high-degree nodes (e.g., the threat-actor and attack-pattern nodes at degree = 8). (b) The Kamada–Kawai panel presents the same edge set with a more balanced geometric distribution, thereby enhancing component separation and providing an intuitive sense of path length. (c) The trajectory panel, however, sacrifices topological detail to highlight the global structural summary; it may therefore be preferred for reporting or summary presentations. Reading the three panels together demonstrates that a single layout is insufficient; findings such as centrality, componentisation and relationship-type density must be evaluated in a layout-sensitive manner as complementary factors.

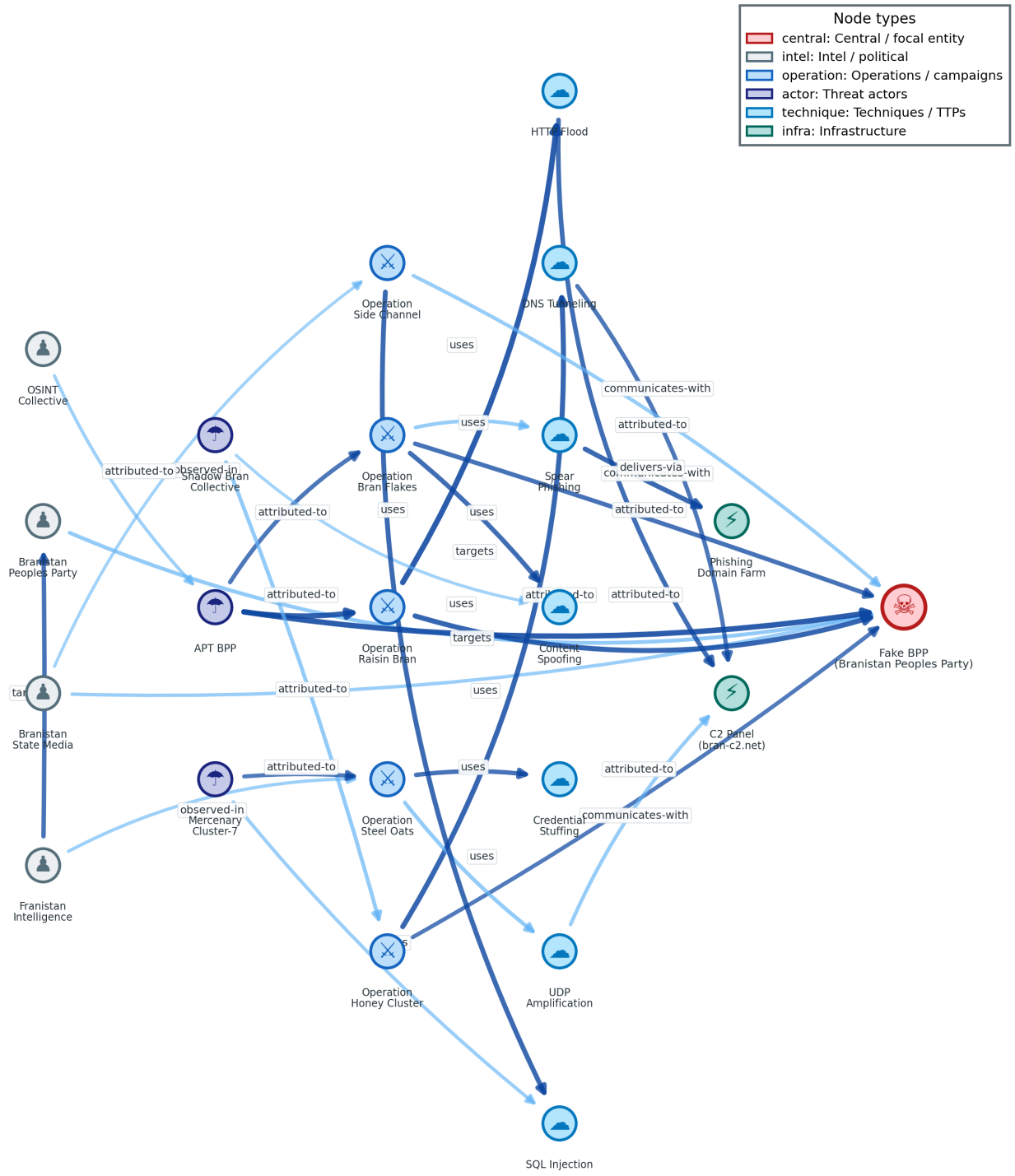
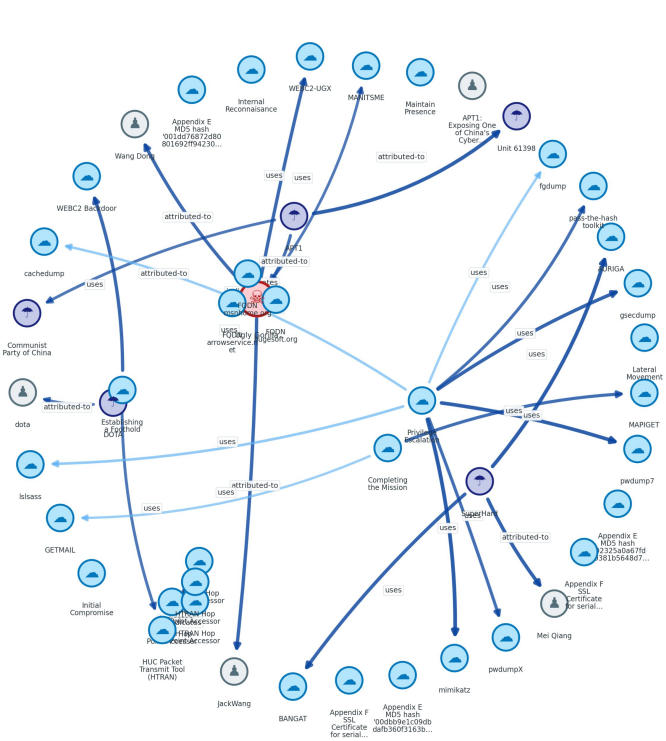
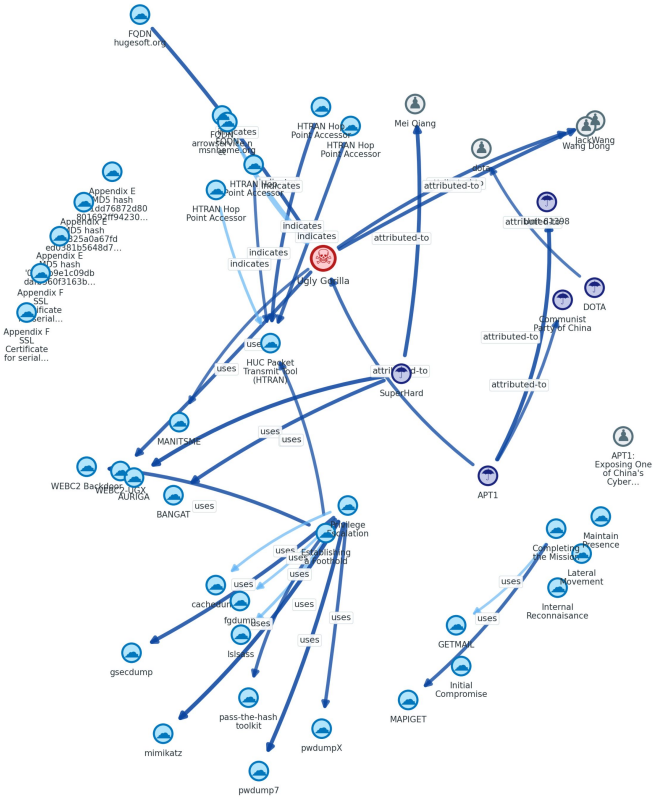


Figure 4. Representative user-interface mockup of the layered (multipartite) column layout and node-class coding

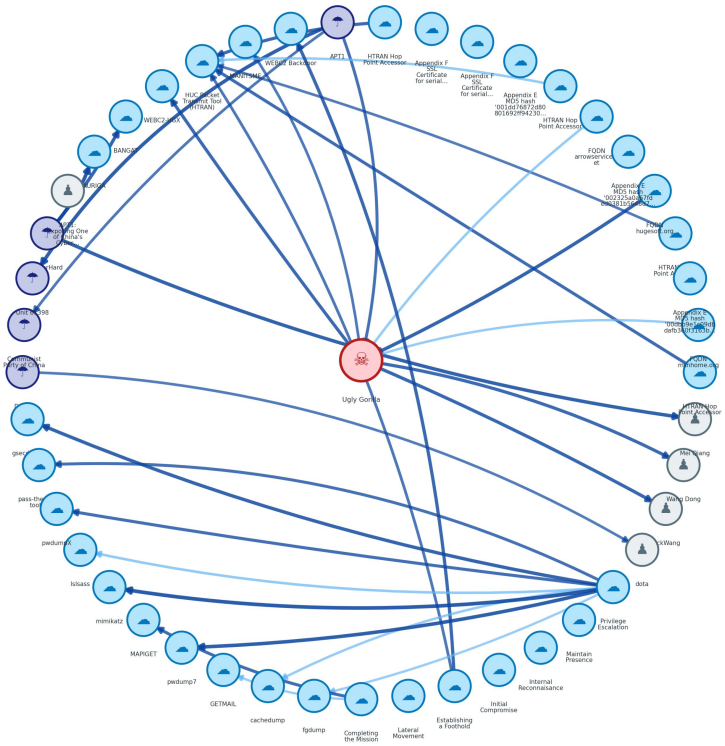
The presentation of these three visualisations together reflects the need for multi-scale interpretation commonly encountered in CTI analysis, rather than claiming a single 'correct' view: a force-directed layout for exploration, Kamada–Kawai for general shape comparison, and an orbit for a simplified summary. Interactive Pyvis and Plotly HyperText Markup Language (HTML) outputs support the same abstraction in the browser; the generation script is provided in supplementary/generate_interactive_figures.py. A static collage has been preferred in the main body of this article for print compatibility. The incremental export costs reported in Table 4 show that, once G is



(a) Force-directed layout via spring relaxation



(b) Kamada–Kawai graph embedding



(c) Single-ring orbit layout about the hub

Figure 5. OASIS apt1.json graph: three layouts of the same NetworkX abstraction

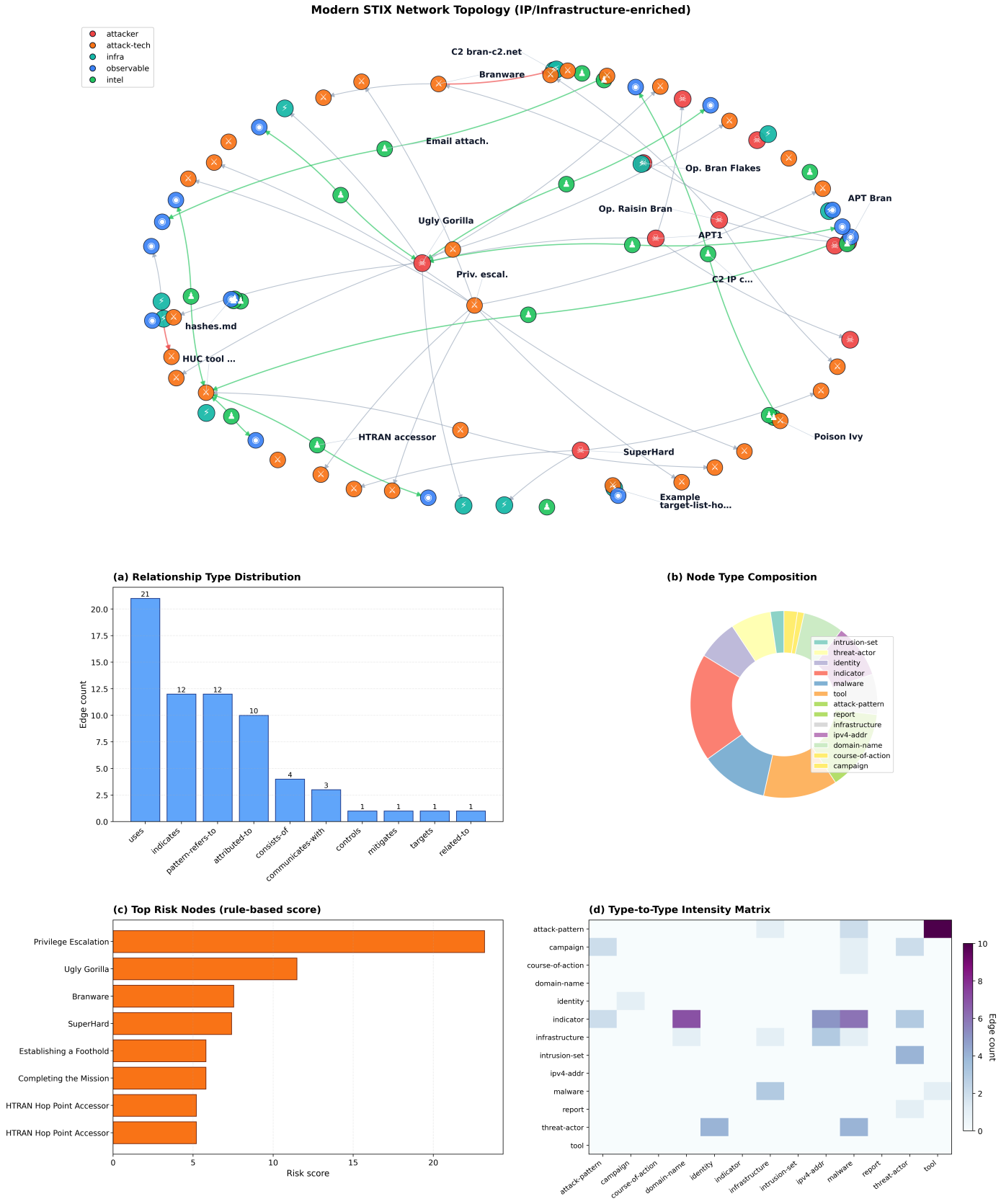


Figure 6. Combined STIX-enriched network topology and dashboard summary

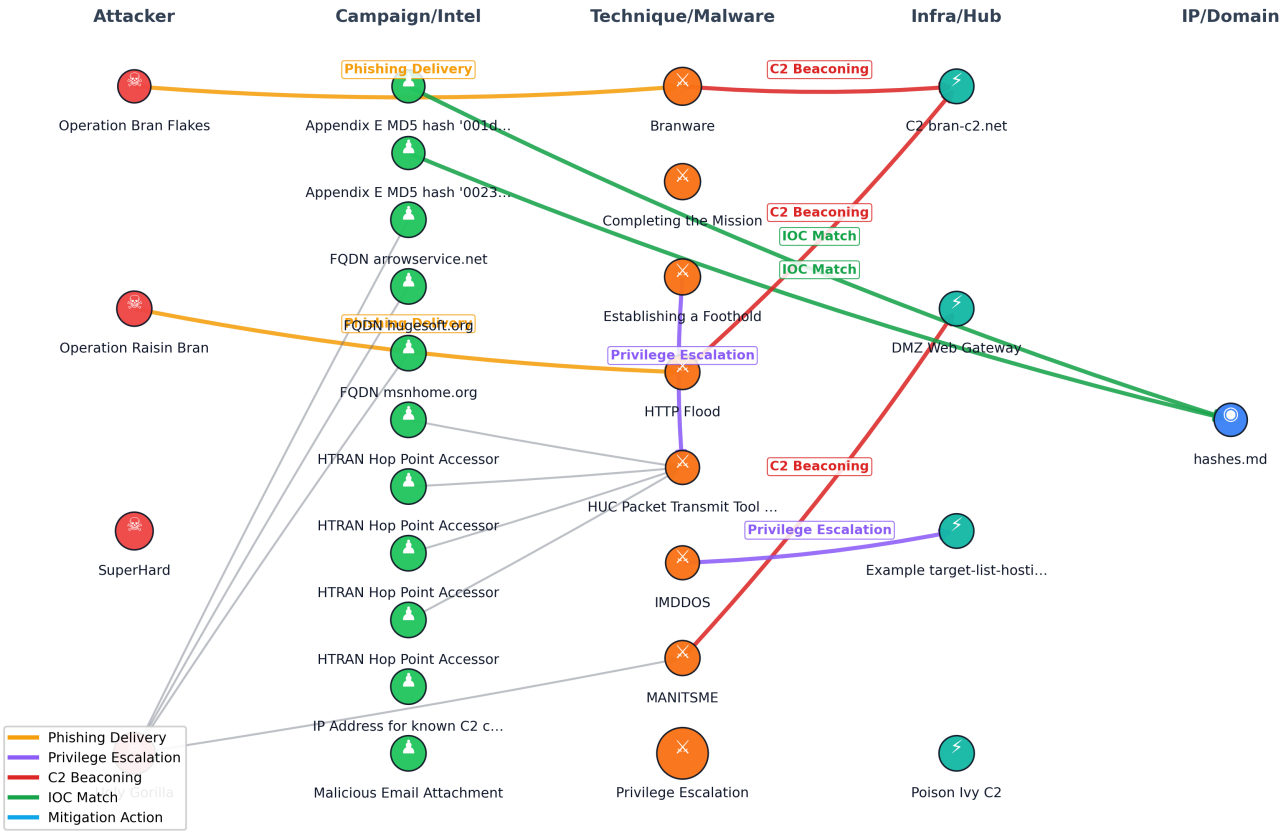


Figure 7. STIX-based attack-detection path view

Table 4. Multi-backend rendering performance (OASIS apt1.json; $|V|$=46, $|E|$=30).

Backend	Primary analyst task	Output	Export (s)	Interact.
matplotlib	Multi-layout static reporting	PNG/PDF	0.48	–
plotly	Browser zoom/hover exploration	HTML	0.15	✓
pyvis	Cluster tracing & drill-down	HTML	0.30	✓
Unified pipeline workflow proxies (single G, one STIX ingestion)				
STIX ingestions for a three-format report			1 (vs. 3 ad-hoc transforms)	
Time to first static view (parse + layout + matplotlib)			0.51	
Time to all three backend exports			0.96	
Preview frame time at $\|V\|$=1000			4.65 ms (215 FPS)	

built, additional interactive views can be generated in sub-second time (plotly: 0.15 s; pyvis: 0.30 s on apt1.json), whilst the preview Frames per second (FPS) values in Table 5 indicate that the analyst feedback loop remains responsive even at $|V|$=1000.

The numbers above (specifically $|V| = 46$ and $|E| = 30$ in Figures 3 and 5) are specific to the OASIS apt1.json file parsed from the public package. In contrast, the dashboard and detection-focused shapes in Figures 6 and 7 use a separate, enriched network built from a combined STIX package: the enrichment step combines or chains multiple STIX entries, instantiates additional observable entities, and implements relationship completion. Therefore, the node and edge totals are not comparable to the single-file apt1 baseline.

Figure 6 provides a quantitative summary of the structural and detection-oriented behaviour of the STIX-based enriched network model before benchmark

measurements. The upper panel shows the role-coded topology of the combined bundle ($|V| = 86$, $|E| = 66$ directed edges); the four labelled dashboard panels below report complementary statistics on the same graph. In subfigure (a), the vertical axis is *edge count* by STIX relationship type; uses (21), indicates (12), pattern-refers-to (12) and attributed-to (10) are the dominant classes. Subfigure (b) summarises node-type composition, showing that abstract threat entities and observables coexist in one graph (e.g., indicator (16), tool (11), malware (10), attack-pattern (10), IPv4-addr (8), infrastructure (6)). Subfigure (c) ranks nodes by rule-based $R(n)$ from Equations (7) and (6) (Section 3.5); Privilege Escalation (23.2) and Ugly Gorilla (11.48) receive the highest scores and emerge as priority points for attack-path analysis. Subfigure (d) reports the type-to-type intensity matrix, with colour mapped to edge count. Consequently, this figure combines an illustrative topology view with reproducible numeric summaries in a single pre-benchmark validation step.

Figure 7 transforms structural network information into operational detection semantics (Table 3) by providing an 'event-flow' view of the same enriched STIX graph. The layered layout facilitates tracing the attack path from the source entity to the observable endpoint, whilst edge classification quantitatively distinguishes the relative density of detection types: Context Link (8), C2 Beaconing (3), IOC Match (2), Phishing Delivery (2) and Privilege Escalation (2). This distribution demonstrates that the model not only maps the existence of a relationship but can also associate it with an analytical detection category. The fact that the nodes with the highest risk scores (particularly Privilege Escalation and the actor/malware core) are located in the early-to-mid layers of the chain indicates that 'pre-containment' points can be topologically identified for practical intervention. Consequently, this figure demonstrates that STIX-based visualisation can be used not only as an explanatory presentation tool but also for detection prioritisation and analyst guidance.

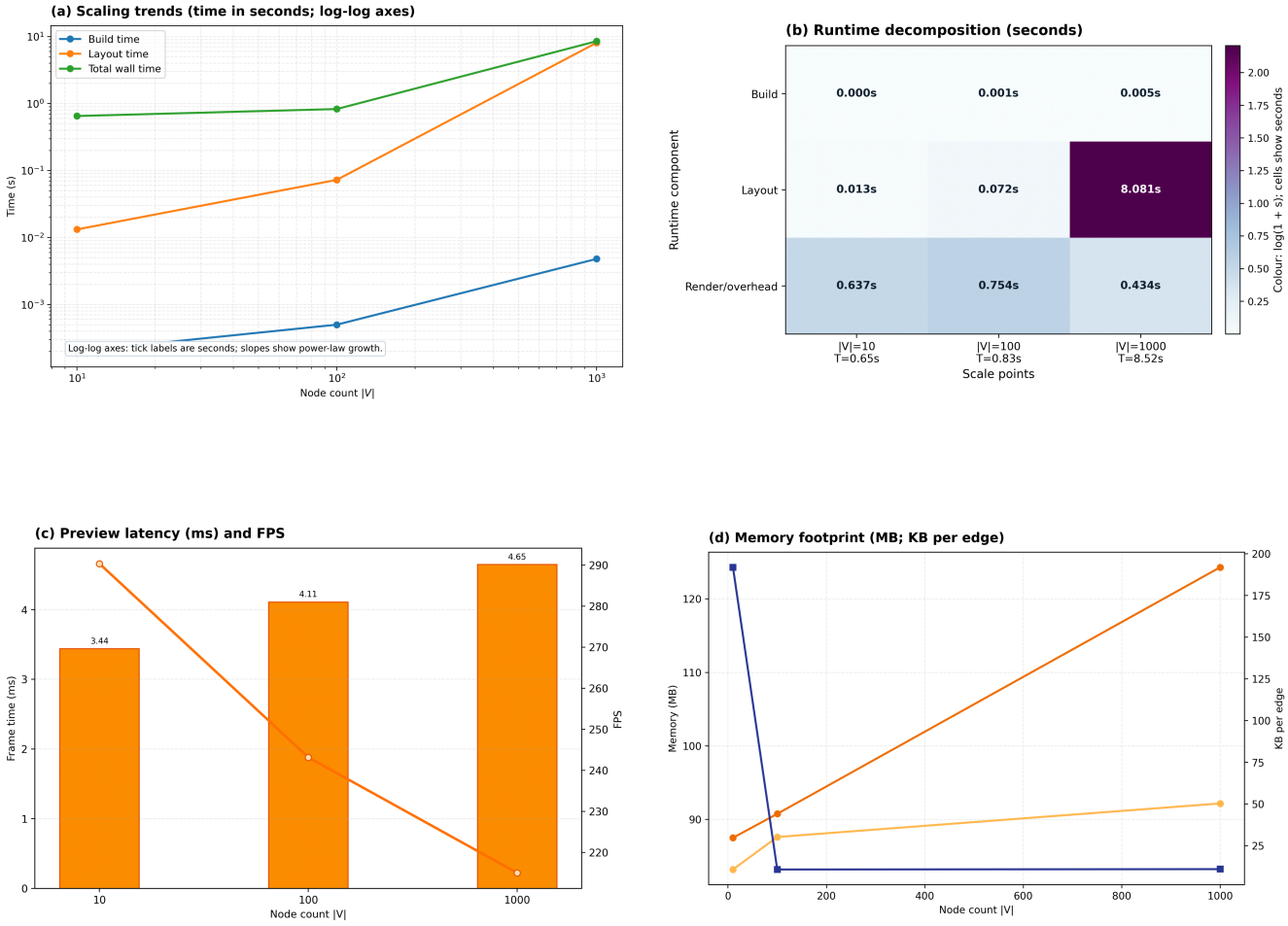


Figure 8. Scalability findings on synthetic directed graphs

Table 5. Scalability results on synthetic directed graphs

$\|V\|$	$\|E\|$	Memory (MB)	Layout (s)	Frame (ms)	FPS
10	23	87.5136	0.01	3.44	290.34
100	300	90.7742	0.07	4.11	243.18
1000	3000	124.3027	8.08	4.65	215.05

To report on scalability and runtime behaviour, directed synthetic graphs were generated to suit the NetworkX-based visualisation workflow used in this study. Placement cost was measured using the hub-centred spring_layout, and FPS was calculated from a rapid preview redraw cycle on a matplotlib Agg canvas rather than from file export (FPS $\approx 1/t_{\text{frame}}$). Therefore, the reported FPS values represent the real-time preview processing efficiency. Memory was measured as Resident set size (RSS) (MB) using psutil. The number of edges was generated such that the average output degree was close to three. Measurements were generated using benchmark_graph_pipeline.py, and the raw outputs were stored in the supplementary/performance_benchmark.json file. A numerical summary of the scalability results is provided in Table 5. These benchmarks characterise workflow behaviour on synthetic graphs up to $|V| = 1000$; they do not constitute validation on continuously ingested, production-scale STIX corpora, which we identify as a direction for future evaluation.

Figure 8 summarises the scalability measurements in Table 5 through four labelled subfigures. Subfigure (a) plots build, layout, and total wall time versus $|V|$ on *log-log axes*; the tick labels report *seconds*, whilst logarithmic spacing compresses the three benchmark scales ($|V| \in \{10, 100, 1000\}$) so that power-law runtime growth can be read as a near-linear segment. Layout cost increases super-linearly from 0.01 s to 8.08 s, dominating the wall-time curve. Subfigure (b) decomposes the same runs into Build, Layout, and Render/overhead components; annotated cell values remain in seconds and confirm that layout accounts for the bottleneck at $|V| = 1000$ (8.081 s versus ≤ 0.754 s for the other components). Subfigure (c) reports preview *frame time* (ms) and *FPS* jointly: frame time rises only from 3.44 ms to 4.65 ms whilst FPS decreases from 290.34 to 215.05, indicating that the interactive preview loop stays responsive at large scale. Subfigure (d) complements this with RSS memory (MB) and incremental KB per edge; absolute consumption increases from 87.5 MB to 124.3 MB, whilst per-edge overhead stabilises after the smallest graph, suggesting amortisation of fixed startup cost. Consequently, Fig. 8 provides a focused, unit-labelled view of computational cost, interaction latency, and memory efficiency on synthetic graphs.

5. Conclusion

In this study, a comprehensive visualisation approach has been developed to analyse STIX 2.x-based cyber threat intelligence data in a more understandable, comparable, and reproducible manner. The proposed framework progresses through a single workflow, from the extraction of raw data to its presentation across different visualisation layers; this enables the analyst to evaluate the same data from different perspectives. The results demonstrate that the method not only generates visual representations but also more clearly highlights attack relationships and key points for investigation. Experimental findings confirm that the approach is also sustainable in terms of practical application. Although processing costs increase as scale grows, interactive usage has been maintained, and memory consumption has remained within manageable limits. Furthermore, integrating relationship-type semantics (e.g., uses, indicates, attributed-to) into a single graph abstraction has enhanced analytical consistency and ensured that detection-focused interpretations can be reproduced across interfaces. Similarly, the fact that the latency–FPS balance remains within a narrow band demonstrates that the analyst feedback loop can be sustained below operational thresholds during the visual exploration phase. In this respect, the study provides a reliable foundation for CTI analysis at both the methodological and application levels. Future work aims to extend this framework to real-time decision-support scenarios by incorporating temporal event streams and automatic alarm correlation.

In summary, the study advances a reproducible STIX 2.x visual analytics workflow that combines canonical parsing, STIX-compliant enrichment, operational detection semantics, comparative multi-backend rendering (matplotlib/plotly/pyvis), and reported scalability characteristics. It does not propose a new STIX standard, a novel layout algorithm, or a new plotting library; the comparative use of existing libraries is an integrative design choice within the pipeline. The risk ranking is rule-based and intended for analyst prioritisation and interpretability rather than as a learned detection model; its parameters may require adaptation across organisations. Formal evaluation of analyst performance is left as future work.

Abbreviations

APT Advanced Persistent Threat

BiLSTM Bidirectional Long Short-Term Memory

C2 Command and Control

CKG Cybersecurity Knowledge Graph

CTI Cyber Threat Intelligence
FPS Frames per second
GAT Graph Attention Network
GCN Graph Convolutional Network
HAG Hyper Attack Graph
HTML HyperText Markup Language
IDS Intrusion Detection System
IOC Indicator of Compromise
IPS Intrusion Prevention System
JSON JavaScript Object Notation
KG Knowledge Graph
LSTM Long Short-Term Memory
MITRE MITRE Corporation
RSS Resident set size
SDO STIX Domain Object
SRO STIX Relationship Object
STIX Structured Threat Information Expression
TiKG Threat Intelligence Knowledge Graph Pipeline
XML Extensible Markup Language

Acknowledgement

This study was supported by the Scientific and Technological Research Council of Turkey - Science Fellowships and Grant Programs Department (TÜBİTAK-BIDEB) through the BİDEB 2219 International Postdoctoral Research Scholarship Program, under the project "Development of New GNN-Based Approaches for Graph Visualisation of Cyber Threat Intelligence Data", conducted at the Department of Cybersecurity and Systems Engineering, School of Computing, Engineering and the Built Environment, Edinburgh Napier University, Scotland. Professor Resul Das gratefully acknowledges TÜBİTAK–BİDEB for its financial support.

References

[1] M. Rabzelj, C. Bohak, L. Š. Južnič, A. Kos, and U. Sedlar, "Cyberattack graph modeling for visual analytics," *IEEE Access*, vol. 11, pp. 86 910–86 944, 2023.

[2] N. Sun, M. Ding, J. Jiang, W. Xu, X. Mo, Y. Tai, and J. Zhang, "Cyber threat intelligence mining for proactive cybersecurity defense: A survey and new perspectives," *IEEE Communications Surveys & Tutorials*, vol. 25, no. 3, pp. 1748–1774, 2023.

[3] S. Saeed, S. A. Suayyid, M. S. Al-Ghamdi, H. Al-Muhaisen, and A. M. Almuhaideb, "A systematic literature review on cyber threat intelligence for organizational cybersecurity resilience," *Sensors*, vol. 23, no. 16, p. 7273, 2023.

[4] R. M. Czekster, R. Metere, and C. Morisset, "Incorporating cyber threat intelligence into complex cyber-physical systems: A stix model for active buildings," *Applied Sciences*, vol. 12, no. 10, p. 5005, 2022.

[5] A. Ramsdale, S. Shiaeles, and N. Kolokotronis, "A comparative analysis of cyber-threat intelligence sources, formats and languages," *Electronics*, vol. 9, no. 5, 2020.

[6] J. Hamza, S. Felix, V. Kunčak, I. Nussbaumer, and F. Schramka, "From verified scala to stix file system embedded code using stainless," in *NASA Formal Methods Symposium*. Springer, 2022, pp. 393–410.

[7] L. J. Borges Amaro, B. W. Percilio Azevedo, F. L. Lopes de Mendonca, W. F. Giozza, R. d. O. Albuquerque, and L. J. García Villalba, "Methodological framework to collect, process, analyze and visualize cyber threat intelligence data," *Applied Sciences*, vol. 12, no. 3, 2022.

[8] M. O. Kaya, M. Ozdem, and R. Das, "A novel approach for graph-based real-time anomaly detection from dynamic network data listened by Wireshark," *EAI Endorsed Transactions on Industrial Networks and Intelligent Systems*, vol. 12, no. 2, 2025.

[9] A. H. Sial, S. Y. S. Rashdi, and A. H. Khan, "Comparative analysis of data visualization libraries matplotlib and seaborn in python," *International Journal*, vol. 10, no. 1, pp. 277–281, 2021.

[10] F. Böhm, F. Menges, and G. Pernul, "Graph-based visual analytics for cyber threat intelligence," *Cybersecurity*, vol. 1, no. 1, p. 16, 2018.

[11] M. Sülü and R. Daş, "Graph visualization of cyber threat intelligence data for analysis of cyber attacks," *Balkan Journal of Electrical and Computer Engineering*, vol. 10, no. 3, pp. 300–306, 2022.

[12] K. Liu, F. Wang, Z. Ding, S. Liang, Z. Yu, and Y. Zhou, "Recent progress of using knowledge graph for cybersecurity," *Electronics*, vol. 11, no. 15, p. 2287, 2022.

[13] L. F. Sikos, "Cybersecurity knowledge graphs," *Knowledge and Information Systems*, vol. 65, pp. 3511–3531, 2023.

[14] X. Zhao, R. Jiang, Y. Han, A. Li, and Z. Peng, "A survey on cybersecurity knowledge graph construction," *Computers & Security*, vol. 136, p. 103523, 2024.

[15] C. Bratsas, E. K. Anastasiadis, A. K. Angelidis, L. Ioannidis, R. Kotsakis, and S. Ougiaroglou, "Knowledge graphs and semantic web tools in cyber threat intelligence: A systematic literature review," *Journal of Cybersecurity and Privacy*, vol. 4, no. 3, pp. 518–545, 2024.

[16] M. Soylu and R. Das, "Prediction and graph visualization of cyber attacks using graph attention networks," *Computers & Security*, vol. 157, p. 104534, 2025.

[17] J. Jia, L. Yang, Y. Wang, and A. Sang, "Hyper attack graph: Constructing a hypergraph for cyber threat intelligence analysis," *Computers & Security*, vol. 149, p. 104194, 2025.

[18] I. Mouiche and S. Saad, "Entity and relation extractions for threat intelligence knowledge graphs," *Computers & Security*, vol. 148, p. 104120, 2025.

[19] MISP Project, "MISP – threat intelligence and sharing platform," https://www.misp-project.org/, 2024, accessed: 2026-04-19.

[20] Filigran, "OpenCTI – open cyber threat intelligence platform," https://www.opencti.io/, 2024, accessed:

2026-04-19.

[21] M. O. Kaya, M. Ozdem, and R. Das, "A new hybrid approach combining GCN and LSTM for real-time anomaly detection from dynamic computer network data," *Computer Networks*, vol. 268, p. 111372, 2025.

[22] OASIS Cyber Threat Intelligence (CTI) TC, "OASIS TC Open Repository: Non-normative Schemas and Examples for STIX 2," https://github.com/oasis-open/cti-stix2-json-schemas, 2023, threat-report examples located at examples/threat-reports/apt1.json. License: BSD-3-Clause. Accessed: Apr. 2025.

[23] L. Yan, Y. Mei, H. Ma, and M. Zhang, "Evolutionary web service composition: A graph-based memetic algorithm," in *2016 IEEE congress on evolutionary computation (CEC)*. IEEE, 2016, pp. 201–208.

[24] C. Chen, J. Li, H.-Y. Zhou, X. Han, Y. Huang, X. Ding, and Y. Yu, "Relation matters: Foreground-aware graph-based relational reasoning for domain adaptive object detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 3, pp. 3677–3694, 2023.

[25] R. Das and M. Soylu, "A key review on graph data science: The power of graphs in scientific studies," *Chemometrics and Intelligent Laboratory Systems*, vol. 240, p. 104896, 2023.

[26] M. Z. Al-Taie and S. Kadry, *Python for graph and network analysis*. Springer, 2017.

[27] G. Hutson and M. Jackson, *Graph Data Modeling in Python: A practical guide to curating, analyzing, and modeling data with graphs*. Packt Publishing Ltd, 2023.

[28] S. S. Qudratovich, "Data visualization in python," *Eurasian Journal of Mathematical Theory and Computer Sciences*, vol. 4, no. 10, pp. 15–22, 2024.

[29] X. Zhu, Z. Li, X. Wang, X. Jiang, P. Sun, X. Wang, Y. Xiao, and N. J. Yuan, "Multi-modal knowledge graph construction and application: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 36, no. 2, pp. 715–735, 2022.