Visual Graph Representations for Supply Chain Risk Detection: A Novel Study on npm Packages

Paulo Vitor C. Lima[1], **Silvio E. Quincozes**[1,2], **Marcelo Z. do Nascimento**[1]

[1]PPGCO – Universidade Federal de Uberlândia (UFU)
[2]Horizon AI Labs/PPGES – Universidade Federal do Pampa (UNIPAMPA)
{paulo.lima1,marcelo.nascimento}@ufu.br
silvioquincozes@unipampa.edu.br

Abstract. *Identifying structural signals in software supply chain dependencies remains an open research challenge. We propose an exploratory approach: representing dependency graphs as adjacency-matrix images, making computer vision tools, texture analysis, Fourier transforms, spatial statistics, directly applicable. For 30 npm packages (15 with documented CVEs, 15 with no documented CVEs in the same period), we convert each dependency subgraph into a normalised* 64×64 *image and extract 16 visual features (LBP, GLCM, Fourier, basic stats). In this exploratory sample, six visual features show nominal group separation (*$p < 0.05$*), while none of the nine classical graph metrics does. LBP features are the most discriminative, reaching a medium effect size (*$|d| = 0.740$*) under spectral node ordering.*

1. Introduction

Package ecosystems such as npm host millions of reusable libraries connected by declared dependencies, forming directed graphs in which a single package may sit in the dependency tree of thousands of downstream applications. The 2021 compromise of *ua-parser-js*, which distributed versions containing malicious code, rapidly exposed downstream consumers through normal dependency installation and update workflows, simply because the package was trusted [Google 2021, Ladisa et al. 2023].

Current defences are reactive: `npm audit` and similar tools fire only after a vulnerability advisory is disclosed, by which point the exposure window may already have been open for days or weeks. Incidents involving widely adopted utilities such as *lodash* and *axios* show that foundational packages with large transitive reach are particularly consequential targets [Zimmermann et al. 2019, Decan et al. 2018]. Whether the dependency graph already carries detectable structural signals before any disclosure is an open question.

Prior graph-based work characterises risk through scalar metrics, degree, density, centrality, transitive closure size [Zimmermann et al. 2019, Decan et al. 2018], but these summaries discard the spatial arrangement of dependencies. The adjacency matrix of a directed graph, normalised and resized, is a greyscale image: hubs appear as filled rows, clusters as dense blocks. Nataraj et al. [Nataraj et al. 2011] exploited this idea for malware classification; geometric deep learning [Bronstein et al. 2017] formalises the graph-image correspondence. No prior work applies visual texture descriptors to dependency graphs for security.

This paper investigates whether such features can statistically discriminate npm packages with documented CVEs from those without. For 30 packages, 15 with documented vulnerabilities and 15 no-CVE controls, we convert each two-hop dependency subgraph into a 64×64 greyscale image and compare 16 visual descriptors against 9 classical graph metrics. Contributions: (i) a reproducible pipeline with three node-ordering strategies; (ii) an exploratory empirical comparison on real npm data; and (iii) open code and data, released upon acceptance.

2. Background and Related Work

Software supply chain security has been studied from the perspectives of attack taxonomy, dependency risk, and vulnerability propagation. Ladisa et al. [Ladisa et al. 2023] systematise open-source supply chain attacks, typosquatting, dependency confusion, account compromise, and malicious injection. Zimmermann et al. [Zimmermann et al. 2019] show that a small number of npm packages account for disproportionate transitive risk, while Decan et al. [Decan et al. 2018] analyse how vulnerabilities spread through dependency networks over time. Williams et al. [Williams et al. 2025] identify early structural detection as an open challenge. These works motivate graph-based analysis but rely on scalar metrics and do not explore visual representations of dependency structures.

The closest methodological precedent is visual malware analysis: Nataraj et al. [Nataraj et al. 2011] rendered binary executables as greyscale images and classified them with texture features without disassembly. The formal link between graphs and image-like domains comes from geometric deep learning [Bronstein et al. 2017], which establishes correspondences between graph convolution and 2D image convolution. Adjacency matrix visualisation has been used for human interpretation in bioinformatics and social network analysis, but not for automated feature extraction in a security context. This work explores that gap.

3. Proposed Methodology

The pipeline runs in six sequential stages, as shown in Figure 1: package group definition, subgraph construction, node ordering, graph-to-image conversion, visual feature extraction, and statistical comparison.

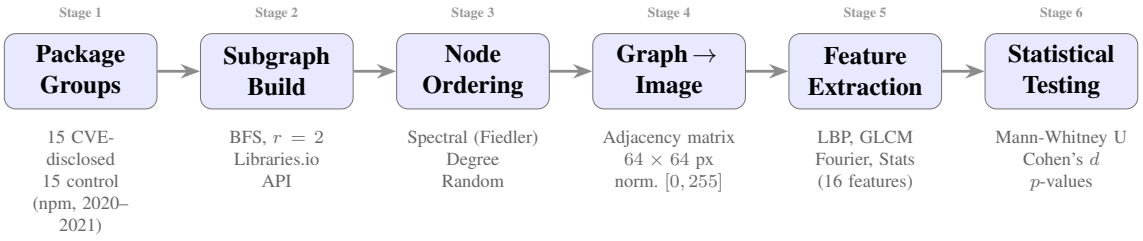

Figure 1. Proposed pipeline for visual supply chain risk detection. Six stages transform real npm package dependency data into statistical evidence about group differences between vulnerability-disclosed and control packages.

3.1. Package Groups

We define two balanced groups, each composed of 15 npm packages selected from real incidents documented in the OSV database [Google 2021].

The **vulnerability-disclosed group** includes packages associated with publicly documented vulnerabilities reported between 2020 and 2021, covering different vulnerability classes. These include prototype pollution vulnerabilities, such as *lodash* (CVE-2021-23337) and *y18n* (CVE-2020-7774); Regular Expression Denial-of-Service vulnerabilities, including *axios* (CVE-2021-3749), *glob-parent* (CVE-2020-28469), and six additional packages; supply chain injection, represented by *ua-parser-js* (GHSA-pjwm-rvh2-c87w); and arbitrary code execution, as observed in *underscore* (CVE-2021-23358).

The **control group** (*no-CVE controls*) consists of popular npm packages with more than one million weekly downloads and no documented CVEs in the same period; the absence of a CVE reflects no publicly disclosed vulnerability in the observation window, not an inherent safety guarantee. This group includes *chalk*, *commander*, *debug*, *express*, *semver*, *uuid*, and nine additional packages selected by ecosystem relevance and popularity.

3.2. Subgraph Construction

For each package p, a local dependency subgraph G_p is built by breadth-first search (BFS) up to radius $r = 2$ hops using the Libraries.io API [Libraries.io 2021]. The choice of $r = 2$ is deliberate: one hop captures direct dependencies, and two hops adds the transitive neighbourhood where supply chain risk typically propagates before reaching the application layer [Decan et al. 2018]. Data collection is rate-limited to 60 requests per minute with automatic exponential-backoff on errors, as required by the API's free tier.

The resulting directed graph has edges pointing from a package to each of its declared dependencies; isolated nodes, packages with no recorded dependencies at the collected version, are kept. Subgraphs were built using the latest version of each package available in the Libraries.io API at the time of data collection; for vulnerability-disclosed packages this corresponds to a post-disclosure snapshot rather than the exact vulnerable release. Results should therefore be interpreted as structural characterization of the dependency neighbourhood, not as prospective pre-disclosure detection.

3.3. Node Ordering and Graph-to-Image Conversion

Given G_p, we sort its nodes using one of three strategies, build the $n \times n$ adjacency matrix A_p in that order, resize to 64×64 pixels with bilinear interpolation, and normalise pixel values to $[0, 255]$. The fixed resolution ensures every package produces a same-size image regardless of subgraph size.

Node ordering is the most consequential methodological choice in the pipeline. The three strategies are: Spectral (Fiedler vector): since the dependency graph is directed, we first symmetrize the adjacency matrix ($A_s = \min(A + A^\top, 1)$) and then compute the graph Laplacian $L = D - A_s$; nodes are sorted by the second-smallest eigenvector of L (the Fiedler vector). This ordering groups structurally similar nodes together, producing block-diagonal patterns when community structure exists, and is the most theoretically motivated choice [von Luxburg 2007]. Degree ordering sorts nodes in descending order of total degree. Hub packages appear in the top rows of the matrix, concentrating edge density in the upper-left corner of the image, and Random ordering is a fixed-seed permutation ($\text{seed} = 42$) that provides a reference for ordering sensitivity. If random-ordered images produce discrimination comparable to structured orderings, the visual signal may

reflect an artefact of the ordering choice rather than graph structure. A systematic comparison across orderings is left for future work.

3.4. Visual Feature Extraction

From each 64×64 image we extract 16 features across four complementary families. LBP [Ojala et al. 2002] (4 features: mean, std, entropy, uniformity) encodes local microtexture by comparing each pixel to its eight neighbours, reflecting local connectivity structures around package pairs. GLCM [Haralick et al. 1973] (4 features: contrast, correlation, energy, homogeneity) captures second-order spatial co-occurrence, measuring the regularity and clustering of dependency connections. Fourier (3 features: low-frequency energy ratio, log total energy, peak concentration) decomposes the image into spatial frequencies; community-structured graphs concentrate energy at low frequencies while irregular graphs spread it to high ones. Basic stats (5 features: visual density, Shannon entropy [Shannon 1948], mean, std, skewness) relate directly to edge density and the distributional shape of the adjacency matrix.

4. Experimental Evaluation

Dependency subgraphs were collected for all 30 packages directly from the Libraries.io API using breadth-first search at radius $r = 2$. The API returns the declared dependencies for a given package version; two-hop traversal therefore captures the immediate transitive neighbourhood. Group differences are assessed with the two-sided Mann-Whitney U test [Mann and Whitney 1947] ($\alpha = 0.05$), chosen over a t-test given $n = 15$ per group; effect sizes are reported as Cohen's d and rank-biserial correlation $r_{rb} = 1 - 2U/(n_1 n_2)$. Because multiple visual features, graph metrics, and node orderings are evaluated, all reported p-values are uncorrected; results are explicitly exploratory.

4.1. Visual Feature Discriminability

Table 1 shows the six visual features that achieve statistical significance ($p < 0.05$) under spectral node ordering. Four of them come from the LBP family, making texture features the dominant discriminative signal. Effect sizes range from small to medium; three features reach $|d| \geq 0.5$ (medium effect).

Table 1. Significant visual features under spectral ordering (real npm data, $n = 15$ per group). Means shown for vulnerability-disclosed (C) and control (K) groups. Effect sizes: Medium $= 0.5 \leq |d| < 0.8$.

Feature	Family	p-value	$\bar{x}_C$	$\bar{x}_K$	$\lvert d \rvert$
`lbp_mean`	LBP	0.028	7.80	7.70	0.494
`lbp_std`	LBP	0.028	0.870	1.220	0.740
`shannon_entropy`	Stats	0.034	0.567	0.803	0.527
`visual_density`	Stats	0.038	0.058	0.079	0.440
`lbp_uniformity`	LBP	0.038	0.892	0.851	0.497
`lbp_entropy`	LBP	0.042	0.430	0.600	0.558

A finding worth pausing on is the direction of the differences. Vulnerability-disclosed packages show *lower* visual density (0.058 vs. 0.079), lower Shannon entropy

(0.567 vs. 0.803), and lower LBP standard deviation. This is counterintuitive if one expects vulnerability-disclosed packages to be complex and densely connected. The explanation likely lies in our dataset composition: the vulnerability-disclosed packages are foundational utilities such as *lodash*, *underscore*, and *y18n*, which are designed to have zero or very few own dependencies. Their outward BFS subgraph is therefore small and structurally regular. The control group includes higher-level packages such as *express* and *moment*, which draw on richer dependency ecosystems. Visual features capture this structural difference; however, the pattern likely reflects functional category, foundational utilities versus higher-level frameworks, rather than a generalizable risk signal. Figure 2 plots all p-values for both the visual features (left panel, spectral ordering) and the classical graph metrics (right panel). The dashed red line marks $\alpha = 0.05$; features to its left achieve nominal significance. No classical metric crosses that threshold.

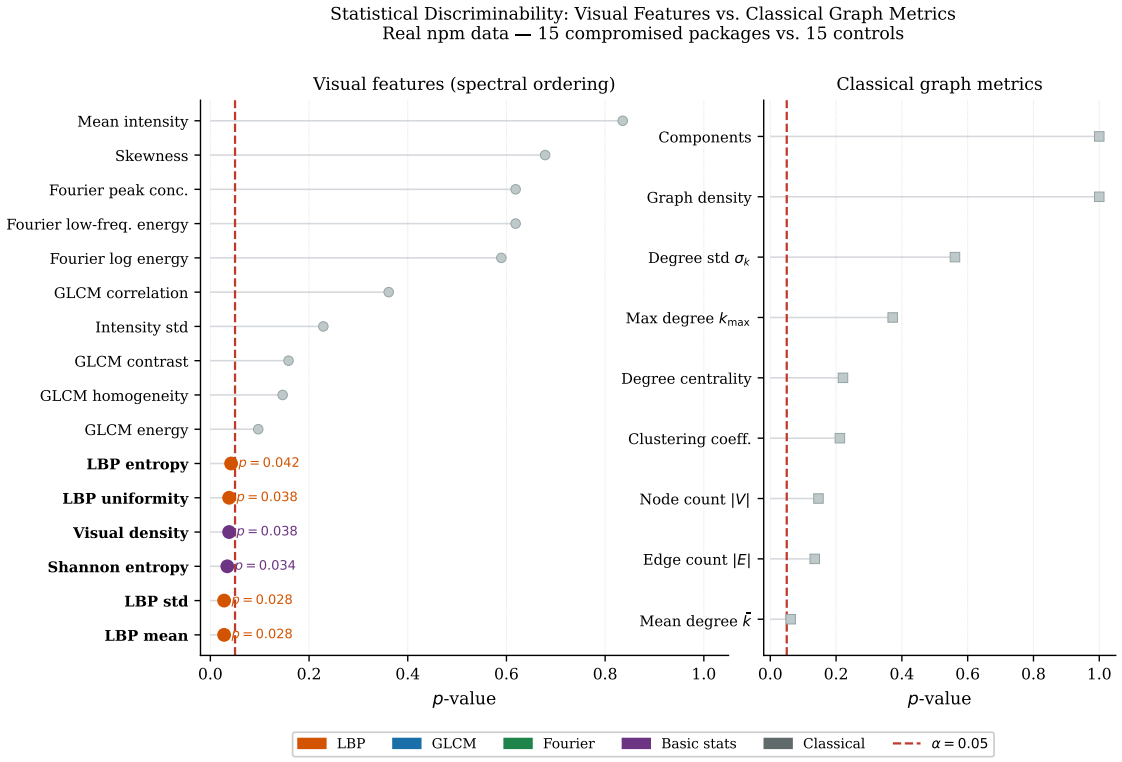

Figure 2. p-values for all visual features (spectral ordering, left) and classical graph metrics (right).

4.2. Visual Features vs. Classical Graph Metrics

None of the nine classical graph metrics achieves $p < 0.05$. The closest is mean degree ($\bar{k}$) at $p = 0.062$, with $|d| = 0.708$, just missing significance. Density reaches $p = 1.0$, and number of connected components also $p = 1.0$, meaning the two groups are indistinguishable on those dimensions.

This is the central empirical finding of this exploratory study. On the same 30 packages, visual features show nominal group separation in six cases while classical metrics show none. The non-significance of classical metrics may also reflect the small sample and specific group pairing rather than a fundamental limitation of scalar approaches.

This comparison does not establish predictive superiority; it shows that the image representation encodes structural information not captured by the nine tested scalar metrics at this sample size.

4.3. Limitations

The small sample ($n = 15$) gives limited power; all p-values are uncorrected across 25 features and three orderings, so results are preliminary. The groups differ in functional category (foundational utilities vs. higher-level frameworks), which may confound results independently of vulnerability status; future work should pair groups by dependency count and functional category. Results are specific to spectral ordering; other orderings may yield different patterns. Bilinear resizing of a binary matrix may introduce artificial intensities affecting LBP and GLCM. Finally, all packages are from npm and reflect a post-disclosure snapshot; transfer to other registries, pre-disclosure temporal analysis, and predictive cross-validation remain open.

5. Conclusions and Future Work

This work presents an exploratory study of visual dependency-graph representations for supply chain security. In 30 npm packages, six of sixteen visual features showed nominal group separation between CVE-disclosed and no-CVE packages ($p < 0.05$), while none of nine classical graph metrics did. These results suggest adjacency-matrix images retain structural information not captured by scalar metrics, but should not be interpreted as detection performance without predictive evaluation. LBP features were the most discriminative; the observed direction (CVE packages show sparser, more regular neighbourhoods) likely reflects functional category, foundational utilities vs. higher-level frameworks, rather than a generalizable risk signal.

Future work includes larger and better-paired datasets, with controls matched by dependency count, popularity, and functional category; pre-disclosure temporal snapshots to assess whether visual patterns emerge before public vulnerability reports; evaluation on other registries, such as PyPI, Maven, and Cargo; and classifier-based risk scoring with proper cross-validation. Additional experiments should also report a systematic sensitivity analysis across node orderings and image-resizing strategies, separating structural signal from artefacts introduced by the graph-to-image conversion.

References

Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A., and Vandergheynst, P. (2017). Geometric deep learning: Going beyond Euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42.

Decan, A., Mens, T., and Constantinou, E. (2018). On the impact of security vulnerabilities in the npm package dependency network. In *Proceedings of the 15th International Conference on Mining Software Repositories (MSR)*, pages 181–191. ACM.

Google (2021). OSV: Open source vulnerabilities database. `https://osv.dev`. Accessed: 2025.

Haralick, R. M., Shanmugam, K., and Dinstein, I. (1973). Textural features for image classification. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-3(6):610–621.

Ladisa, P., Plate, H., Martinez, M., and Barais, O. (2023). SoK: Taxonomy of attacks on open-source software supply chains. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, pages 1509–1526. IEEE.

Libraries.io (2021). Libraries.io open source repository and dependency metadata. `https://libraries.io/api`. Accessed: 2025.

Mann, H. B. and Whitney, D. R. (1947). On a test of whether one of two random variables is stochastically larger than the other. *Annals of Mathematical Statistics*, 18(1):50–60.

Nataraj, L., Karthikeyan, S., Jacob, G., and Manjunath, B. (2011). Malware images: Visualization and automatic classification. In *Proceedings of the 8th International Symposium on Visualization for Cyber Security (VizSec)*, pages 1–7. ACM.

Ojala, T., Pietikäinen, M., and Mäenpää, T. (2002). Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):971–987.

Shannon, C. E. (1948). A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423.

von Luxburg, U. (2007). A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416.

Williams, L., Benedetti, G., Hamer, S., Paramitha, R., Rahman, I., Tamanna, M., Tystahl, G., Zahan, N., Morrison, P., Acar, Y., Cukier, M., Kästner, C., Kapravelos, A., Wermke, D., and Enck, W. (2025). Research directions in software supply chain security. *ACM Trans. Softw. Eng. Methodol.*, 34(5).

Zimmermann, M., Staicu, C.-A., Tenny, C., and Pradel, M. (2019). Small world with high risks: A study of security threats in the npm ecosystem. In *Proceedings of the USENIX Security Symposium*, pages 995–1010. USENIX Association.