Structural Evolution of the Python Dependency Network: Evidence from a Decade of Ecosystem Growth

Juan Ignacio Catania[1], Mateo Guerrero Schmidt[1], Sofia Gutierrez[1], Martina Rosario Pérez[1], and Carlos Sarraute[2,3]

[1] Universidad de Buenos Aires
[2] Disruptive Research Institute
[3] Instituto Tecnológico de Buenos Aires (ITBA)
csarraute@itba.edu.ar

Abstract. We present a longitudinal network analysis of the Python package dependency ecosystem, revealing systematic structural transformations over nearly a decade of evolution. Software package ecosystems can be modeled as complex networks whose topology reflects development practices and technological trends. To characterize this evolution, we construct directed dependency graphs for two large-scale snapshots (2016 and 2025) and compute structural metrics, centrality measures, and community organization.
Our results show a massive expansion of the ecosystem accompanied by clear topological shifts. The number of packages increases by more than an order of magnitude, while dependencies grow even faster, indicating strong densification. We also observe an increase in average degree, a decrease in modularity, and a marked reconfiguration of central nodes. Whereas earlier stages were dominated by web-development and compatibility libraries, recent snapshots show data science, scientific computing, and testing packages emerging as structural hubs.
These findings indicate that the Python ecosystem evolves toward a more interconnected architecture and follows growth patterns consistent with established laws of complex networks, including densification and hub reorganization. More broadly, this study provides empirical evidence that software dependency ecosystems exhibit systematic structural evolution and demonstrates the value of network science methods for understanding the dynamics, organization, and robustness of large technological systems.

Keywords: complex networks · dependency networks · software ecosystems · network evolution · centrality analysis

1 Introduction

The Python library ecosystem has experienced extraordinary growth over the past decade, driven by the expansion of disciplines such as data science, machine learning, scientific computing, and software engineering. As the number

of available packages increases, so does the complexity of dependency relationships among them, giving rise to structures that can be analyzed as large-scale directed networks.

From the perspective of network science, software ecosystems constitute complex technological systems in which the global architecture emerges from the local interaction between components. The study of these networks allows identifying structural patterns, functional roles of nodes, and evolutionary regularities that are not evident at the individual level. In particular, analyzing the temporal evolution of dependency networks is fundamental to understanding how the centrality, modularity, and connectivity of these systems change as they grow.

Although there are studies comparing the evolution of multiple package ecosystems [5, 7], empirical knowledge about how the architecture of a single large-scale ecosystem evolves longitudinally from a network science perspective is still limited. This lack of temporal analysis makes it difficult to assess whether such systems follow known structural laws in complex networks, such as progressive densification or reduction of effective diameter [8], reorganization of hubs, or changes in modularity.

In this work we analyze the dependency network of Python libraries in two time snapshots separated by almost a decade (2016 and 2025). For each year we construct a directed graph in which nodes represent packages and edges represent dependency relationships, and we calculate structural and centrality metrics, analyze relevant subgraphs, and detect communities using partitioning algorithms.

This work makes the following main contributions: (i) Analyzes two large-scale graphs of the PyPI ecosystem separated by 9 years. (ii) Quantitatively characterizes their structural evolution. (iii) Identifies changes in central nodes and community organization. (iv) Provides empirical evidence on how real software ecosystems evolve.

The results show that the Python ecosystem not only grows in size, but undergoes significant structural reorganization, with increased density, reduced modularity, and a shift of central libraries towards areas such as data science, testing, and static typing.

2 Related work

The analysis of software ecosystems using network science tools falls within the general study of complex systems representable as graphs. Package dependency networks constitute natural examples of directed technological networks, comparable to complex infrastructures such as the Web, citation networks, or biological networks. Previous studies have shown that large-scale technological systems tend to exhibit universal structural regularities —heavy-tailed degree distributions, presence of hubs, hierarchical organization, and non-random modularity [13, 12]— which respond to underlying generative mechanisms such as incremental growth and preferential attachment [1]. At the macroscopic level, laws have been identified that characterize the growth of real graphs, such as the laws of

densification and reduction of effective diameter, which show that many empirical networks become progressively denser as they grow [8], constituting a relevant theoretical framework for the analysis of software ecosystems.

Within the specific field of package dependency networks, [7] analyzed multiple repositories finding evidence of increasing centralization and unequal distribution of dependencies, while [5] performed a longitudinal comparison of seven package ecosystems and observed that growth is usually accompanied by greater structural coupling and dependence on a reduced subset of central libraries. A similar analysis has also been applied to specific ecosystems such as R packages, characterizing their dependency structure and collaboration patterns [11, 9]. From a robustness perspective, studies in the npm ecosystem have shown that nodes in critical structural positions can act as single points of failure [14]. Complementarily, from a socio-technical perspective, [3] showed that community norms and compatibility decisions govern the evolution of these ecosystems, suggesting that changes in development practices have observable structural correlates.

In this context, the present work differs from previous studies by examining in depth the internal structural evolution of a single ecosystem —Python— using two snapshots separated by almost a decade. This approach allows directly contrasting whether regularities such as densification, hub reconfiguration, and loss of modularity, documented in other domains [8], also manifest in the PyPI ecosystem.

3 Graph construction and network metrics

3.1 Data sources

To carry out the analysis of the Python library dependency network, we constructed two datasets corresponding to the years 2016 and 2025, both originating from the Python Package Index (PyPI), the main public and official source for Python package distribution.

The 2025 dataset was obtained directly from PyPI through a scraping process on its public metadata, described in detail in Section 3.2. The 2016 dataset comes from previous work by Kevin Gullikson, a student at Texas University [6], who also extracted the information from PyPI, although through a different access method —available at that time— which is no longer operational today. Both datasets are therefore comparable in origin, differing only in the extraction method used in each period.

All the code, data, and resulting graphs from this work are publicly available in the repository:
`https://github.com/empromptu/python-ecosystem-evolution`

3.2 Preprocessing and cleaning

The information download was carried out from the metadata published by each package on PyPI, which includes, among other fields, the package name, its

version, the list of declared dependencies, and a brief description of the package. Once the raw data was collected, we applied a cleaning and standardization process that comprised the following steps:

1. Removal of duplicate packages or redundant versions, keeping only the relevant version for each year.
2. Normalization of package names, due to style variations or inconsistencies in how some projects declare their dependencies.
3. Filtering of invalid or orphan dependencies, that is, those that refer to packages that do not exist in the index.
4. Final construction of a consistent list of pairs (`package` $\rightarrow$ `dependency`), suitable for modeling as a directed graph.

This process ensured that both datasets were comparable with each other and that the resulting network reflected the real structure of the Python library ecosystem as faithfully as possible for each year analyzed. The code used for data cleaning is part of the project repository mentioned above, while the final datasets are deposited in Zenodo [4].

3.3 Dependency graph construction and limitations

From the preprocessed data, we constructed a directed graph for each year using the NetworkX library, where each node represents a Python package and each directed edge ($u \rightarrow v$) indicates that package u declares v as a dependency. Edges are unweighted, since the analysis focuses on the topological structure of the network and not on the frequency or intensity of dependencies. The resulting graphs for 2016 and 2025 are deposited in Zenodo [4].

This work has some limitations that should be considered when interpreting the results. First, the 2025 dataset was constructed from PyPI metadata, so it may include biases inherent to the index (outdated packages, incomplete metadata, or undeclared optional dependencies). Second, the temporal comparison is made between two specific snapshots (2016 and 2025), which prevents capturing annual dynamics or short-term effects.

3.4 Network metrics considered

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a directed graph representing the dependency network, where $\mathcal{V}$ is the set of nodes (packages) and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ the set of directed edges (dependency relationships). To characterize its structure, the following metrics were calculated.

Betweenness centrality. Measures the fraction of shortest paths between pairs of nodes that pass through a node $v \in \mathcal{V}$: $C_B(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$, where σ_{st} is the total number of shortest paths between s and t, and $\sigma_{st}(v)$ those that pass through v.

PageRank. We used the PageRank algorithm [10], which assigns to each node a score defined recursively as

$$PR(v) = \frac{1-d}{|\mathcal{V}|} + d \cdot \sum_{u \in \mathcal{V}_{in}(v)} \frac{PR(u)}{k_{out}(u)},$$

where d is the damping factor (usually $d = 0.85$) and $\mathcal{V}_{in}(v)$ the set of nodes pointing to v.

Modularity and community detection. The community structure was detected using the Louvain algorithm [2], which maximizes modularity Q, defined as

$$Q = \frac{1}{2m} \sum_{ij} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta(c_i, c_j),$$

where A_{ij} is the adjacency matrix, $m = |\mathcal{E}|$, k_i is the degree of node i, and $\delta(c_i, c_j)$ is 1 if nodes belong to the same community and 0 otherwise. All metrics were calculated using the NetworkX library in Python.

4 Structural properties of the ecosystem in 2016

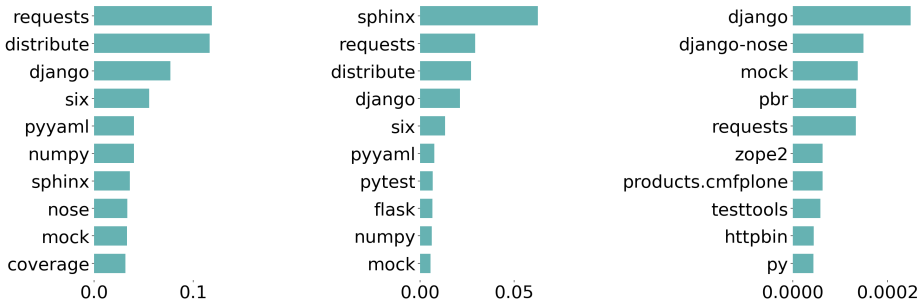
In this section we analyze the structure of the graph from global properties to community organization. To begin the analysis of the graph, we first explored some of its fundamental structural characteristics, such as the number of nodes, the number of edges, and the average degree. The graph, which will be referred to as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, has $|\mathcal{V}| = 25,819$ nodes and $|\mathcal{E}| = 72,189$ directed edges. The average in-degree and out-degree are identical, both with an approximate value of 2.79, leaving the total average degree close to $\langle k \rangle \simeq 5.59$.

4.1 Centrality distribution

After this preliminary analysis, we proceeded to examine $\mathcal{G}$ in more detail using the three centrality metrics mentioned in the methodology section. The results of the ten most prominent nodes in each metric are shown in Figure 1.

As a first observation, we identify that the requests library appears recurrently in the top 5 of all measures, positioning it as a highly relevant node from multiple perspectives. The fact that requests occupies such high positions in *degree centrality* and *PageRank* indicates that it is pointed to by numerous nodes, that it is structurally close to much of the graph and that it receives links from other important nodes. Additionally, its presence in *PageRank* in second place reinforces the idea that its role is not only due to the number of dependencies – but it is also central within the system hierarchy.

The results show that both in *Degree centrality* and *PageRank*, several packages repeat in the top. However, when observing *Betweenness centrality*, the picture changes: many of the libraries highlighted in this metric do not appear



(a) Degree centrality (b) PageRank (c) Betweenness centrality

Fig. 1. Distribution of the main centrality metrics in the 2016 dependency graph. The ten packages with the highest value according to each metric are shown. The recurrence of certain packages across metrics indicates the existence of structurally dominant nodes that play central roles in the global organization of the network.

in the previous rankings. This is expected, since the metric captures nodes that, without having large degrees or privileged positions in terms of closeness, are essential for connecting different regions of the graph.

At the centrality level, we distinguish nodes that are important for their global visibility (such as requests and distribute) and others whose relevance is related to their function as "articulation" between groups (such as mock, and pbr). This diversity of structural roles is consistent with the typical dynamics of broad and decentralized technological networks such as the Python ecosystem.

4.2 Community structure

We analyzed the entire network to identify communities using the Louvain algorithm seeking to maximize modularity [2]. This step allows knowing how many communities exist, the size of each one, and the relative importance of each node through the calculation of degree and *PageRank*. Here we found 639 communities, with the largest community having 5840 nodes. The modularity of that division was 0.615.

To be able to graph and carry out a more detailed analysis, we created subgraphs with the most important nodes according to *PageRank*, allowing us to analyze the structure of the network focusing on the central nodes. We can see one of these graphs in Figure 2.

4.3 Semantic characterization of communities

To better understand the composition of each community, we decided to study the textual descriptions of the packages that compose them. The text was processed by removing common or irrelevant words in order to identify the most descriptive ones for each community. In parallel, we evaluated the relevance of

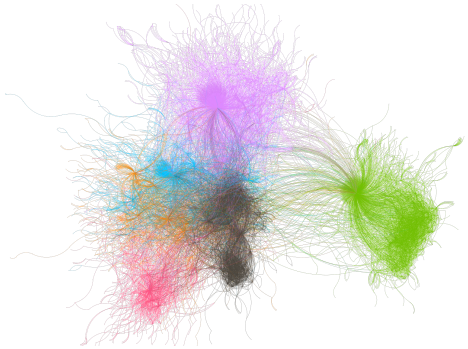


Fig. 2. Subgraph of the 2016 ecosystem from the nodes with highest *PageRank*, filtering by the largest connected component and the 6 largest communities of that subgraph. The result has 6,548 nodes and 16,244 edges. It can be seen that most communities are well separated with little overlap between them.

each node through its *PageRank* value. This combination allows understanding the predominant themes of each community.

(a) First largest community (b) Second largest community

Fig. 3. Wordclouds of the brief descriptions of packages in the two largest communities in number of packages of the 2016 graph. Barplots with nodes belonging to each community ordered according to their PageRank value in the complete network. A strong presence of web development tools can be seen.

Analyzing the 10 largest communities, we can see that most have a focus on web tools. In Figure 3 we see that the first community is led by `distribute` and contains many packages with low *PageRank*. The second community is led by the `requests` package and consists of parsing and APIs tools. It is interesting to note that scientific libraries only appear as the seventh largest community in 2016.

5 Structural properties of the ecosystem in 2025

For the dataset corresponding to the year 2025, the initial exploration strategy was repeated, although with certain adjustments necessary due to the significantly larger size of the dataset. Unlike the graph corresponding to 2016, the

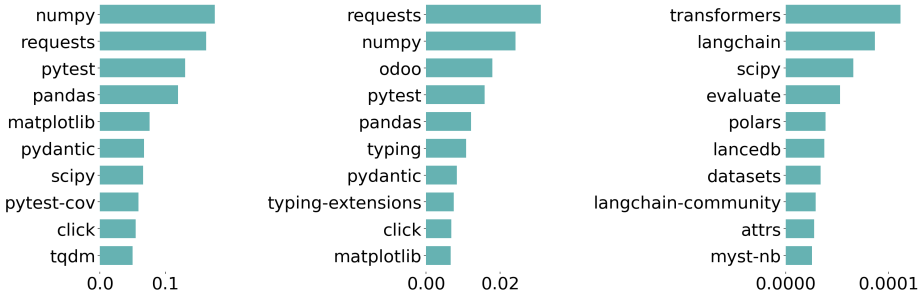
2025 network not only shows a significant increase in size, but also relevant structural changes in terms of density, centrality, and community organization, which are analyzed below.

The constructed graph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ is a directed graph composed of $|\mathcal{V}'| = 405,872$ nodes and $|\mathcal{E}'| = 2,076,516$ edges. The analysis of its degrees reveals that both the in-degree and the out-degree have an average close to 5.11, which places the total average degree around $\langle k \rangle \simeq 10.23$.

5.1 Centrality distribution

When calculating centrality metrics, it was not possible to directly compute *betweenness centrality* due to the high computational complexity of the algorithm and the significantly larger dataset. Therefore, we chose to estimate it using an approximation based on random sampling.

Figure 4 illustrates the most relevant nodes according to the metrics considered. We see that requests is again positioned among the most influential packages in the graph, reflecting its sustained use within the ecosystem and its central role in projects that require HTTP communication.



(a) Degree centrality (b) PageRank (c) Betweenness centrality

Fig. 4. Distribution of the main centrality metrics in the 2025 dependency graph. The ten nodes with the highest value for each metric are shown.

A library called pytest emerges among the most influential, which is a widely used tool for test automation. Its position in the top spots reflects the massive adoption of testing practices in software development in recent times.

Another name that gains relevance is pandas, a fundamental library for data analysis. Its presence among nodes with the highest centrality is consistent with the exponential growth of data processing tasks in the Python community. Finally, the NumPy library also appears among the most influential. It is a central library of the Python scientific ecosystem that provides optimized data structures for numerical computation, particularly the multidimensional array. NumPy constitutes the foundation upon which numerous libraries oriented to data analysis, mathematical modeling, and machine learning are developed.

As in 2016, the results show similarities between packages in *degree centrality* and *PageRank*, while in *degree centrality* other packages appear. This distin-

guishes between nodes that are important for their global visibility and others whose relevance is related to their function as "articulation" between groups. Together, these results show that, while certain packages maintain their historical predominance, the 2025 graph reveals a scenario where new areas of interest —such as automated testing and data analysis— acquire considerable structural weight within the network.

5.2 Community structure

A similar analysis to that done with the previous network was carried out but with very different results. We found 3,829 communities, the largest with 116,312 nodes and modularity dropped notably to 0.426. In Figure 5 we can see how one of the graphs made on this dataset turned out.

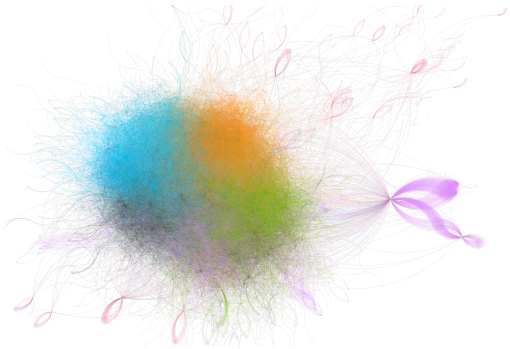


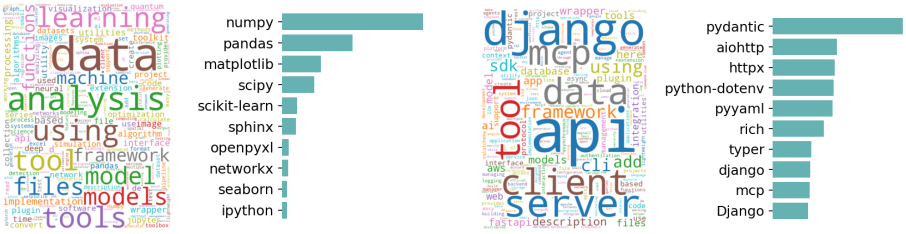
Fig. 5. Subgraph of the 2025 ecosystem from the nodes with highest *PageRank*, filtering by the largest connected component and the 6 largest communities of that subgraph. The result has 7,899 nodes and 41,318 edges. The increase in density and the blurring of boundaries between communities evidence a reduction in modularity and greater global interconnectedness compared to the 2016 network.

From these images we can better understand the drop in modularity since the dividing lines of communities are much more diffuse. The new dataset has many more edges, which complicates separation. For example, in the subgraph containing the 10,000 nodes with highest *PageRank*, the 2016 subgraph (unfiltered) has 24,915 links, while the 2025 subgraph contains 50,823 links.

5.3 Semantic characterization of communities

When comparing the wordclouds of the 2016 network with the 2025 network, significant changes are seen. In Figure 6 we see that the community of scientific packages (numpy, pandas, matplotlib, scipy, scikit-learn) went from being the seventh largest community to being the first. But additionally, the sixth largest community is more specific to machine learning and artificial intelligence (torch, openai, tensorflow, transformers). It is striking that tqdm appears in this community, a library for showing progress bars, suggesting that many of these packages have

long execution times. On the other hand, packages related to web development have lost centrality: only in fifth place does the community with parser packages appear (requests, beautifulsoup4, lxml). Finally, looking at the general level, we can see that whereas before the most common words were API, django, client; now they are data, API, and tool.



(a) First largest community (b) Second largest community

Fig. 6. Wordclouds of the brief descriptions of packages in each of the two largest communities in number of packages of the 2025 graph. Barplots with nodes belonging to each community ordered according to their PageRank value in the complete network.

6 Evolutionary dynamics of the ecosystem

With the aim of characterizing the structural evolution of the Python dependency ecosystem, we compared global metrics of the graphs corresponding to 2016 and 2025 (see Table 1). This comparison allows analyzing not only the quantitative growth of the system, but also qualitative changes in its topological organization.

Table 1. Global metrics of the dependency graph by year.

Metric	2016	2025
Nodes	25,819	405,872
Edges	72,189	2,076,516
Average degree	5.59	10.23
Modularity	0.615	0.426

6.1 Densification law

In the evolution of the ecosystem, massive growth is observed: the number of nodes increases by more than an order of magnitude and the number of edges grows even faster. This pattern suggests a densification process, a phenomenon characteristic of growing real networks, in which the number of links increases superlinearly with respect to the number of nodes.

To evaluate whether the ecosystem growth follows known patterns of real networks, we analyzed the relationship between the number of edges and the

number of nodes in both snapshots. In many complex networks, a densification law of the form $|\mathcal{E}(t)| \sim |\mathcal{V}(t)|^{\alpha}$, is observed, where $\alpha > 1$ indicates superlinear growth. From the observed values, the exponent can be estimated as

$$\alpha = \frac{\log|\mathcal{E}_{2025}| - \log|\mathcal{E}_{2016}|}{\log|\mathcal{V}_{2025}| - \log|\mathcal{V}_{2016}|}.$$

The estimated densification exponent is $\alpha \approx 1.22$, which indicates superlinear growth of the number of edges with respect to the number of nodes. This result is consistent with patterns observed in previously studied technological and information networks, where values $\alpha > 1$ reflect processes of progressive structural integration.

6.2 Interdependence and functional reorganization

Second, the increase in average degree indicates that modern packages tend to depend on more libraries than in earlier stages of the ecosystem. This result suggests greater component reuse and deeper integration between modules, which implies an increase in structural coupling.

From the mesoscopic point of view, the decrease in modularity observed in 2025 indicates that boundaries between communities become less defined. This behavior is consistent with a process of progressive integration between previously more separated functional domains, such as web development, scientific computing, and testing tools.

Finally, the centrality analysis shows a reconfiguration of structurally dominant nodes. While in 2016 libraries associated with compatibility and web development predominated, in 2025 packages linked to data science, static typing, and automated testing emerge as central hubs. This shift reflects deep changes in development practices and the predominant application domains of the language.

Together, these results suggest that the Python ecosystem not only grows in size, but undergoes a systematic structural transformation characterized by densification, greater interdependence, and functional reorganization of its central nodes. It is important to note that highly popular libraries such as PyTorch do not appear as central nodes in the dependency network, which reflects the difference between usage popularity and structural centrality.

7 Conclusions

We presented a longitudinal network analysis of the Python package dependency ecosystem using large-scale snapshots from 2016 and 2025. Our results reveal strong ecosystem expansion accompanied by clear topological transformations, including densification, decreasing modularity, and substantial hub reorganization.

The ecosystem evolved from a structure dominated by web-development and compatibility libraries toward one centered on data science, scientific computing, and testing frameworks. These changes suggest increasing interdependence among packages and the emergence of new technological cores.

More broadly, our findings show that software dependency ecosystems exhibit systematic structural evolution consistent with established complex-network growth patterns. This work also illustrates the value of network science methods for studying the organization, robustness, and long-term dynamics of large technological systems.

References

1. Barabási, A.L., Albert, R.: Emergence of scaling in random networks. Science **286**(5439), 509–512 (1999). https://doi.org/10.1126/science.286.5439.509
2. Blondel, V.D., Guillaume, J.L., Lambiotte, R., Lefebvre, E.: Fast unfolding of communities in large networks. Journal of statistical mechanics: theory and experiment **2008**(10), P10008 (2008)
3. Bogart, C., Kästner, C., Herbsleb, J., Thung, F.: How to break an api: Cost negotiation and community values in three software ecosystems. Proceedings of the 24th ACM SIGSOFT Symposium on Foundations of Software Engineering pp. 109–120 (2016)
4. Catania, J.I., Pérez, M.R., Guerrero Schmidt, M., Gutierrez, S., Sarraute, C.: Python package dependency network dataset (2016–2025) (2025). https://doi.org/10.5281/zenodo.18912507
5. Decan, A., Mens, T., Claes, M.: An empirical comparison of dependency network evolution in seven software packaging ecosystems. Empirical Software Engineering **24**, 381–416 (2019). https://doi.org/10.1007/s10664-018-9630-5
6. Gullikson, K.: Python dependency analysis. `https://kgullikson88.github.io/blog/pypi-analysis.html` (2016), accessed: 2026-02-28
7. Kikas, R., Gousios, G., Dumas, M., Pfahl, D.: Structure and evolution of package dependency networks. In: Proceedings of the 14th International Conference on Mining Software Repositories. pp. 102–112 (2017). https://doi.org/10.1109/MSR.2017.55
8. Leskovec, J., Kleinberg, J., Faloutsos, C.: Graph evolution: Densification and shrinking diameters. ACM Transactions on Knowledge Discovery from Data **1**(1), 2 (2007). https://doi.org/10.1145/1217299.1217301
9. Mora-Cantallops, M., Sánchez-Alonso, S., García-Barriocanal, E.: A complex network analysis of the comprehensive R archive network (CRAN) package ecosystem. Journal of Systems and Software **170**, 110744 (2020). https://doi.org/10.1016/j.jss.2020.110744
10. Page, L., Brin, S., Motwani, R., Winograd, T.: The PageRank citation ranking: Bringing order to the web. Tech. rep., Stanford InfoLab (1999), `http://ilpubs.stanford.edu:8090/422/`
11. Salgado, A., Caridi, I.: Network of R packages: A characterization of an empirical collaborative network. Chaos, Solitons & Fractals **155**, 111756 (2022). https://doi.org/10.1016/j.chaos.2021.111756
12. Valverde, S., Ferrer-Cancho, R., Solé, R.V.: Scale-free networks from optimal design. Europhysics Letters **60**(4), 512–517 (2005)
13. Valverde, S., Solé, R.V.: Hierarchical small worlds in software architecture. arXiv preprint cond-mat/0307278 (2003)
14. Zimmermann, M., Staicu, C.A., Tenny, C., Pradel, M.: Small world with high risks: A study of security threats in the npm ecosystem. In: Proceedings of the 28th USENIX Security Symposium. pp. 995–1010 (2019)