Network Science Project Proposal

The PyPI Dependency Network for the Data Science & ML Ecosystem

Ömer Yildirim, Yuxuan Wang, Qingcheng Wang, Yi Wang

November 15, 2025

1. Motivation and Problem Statement

The Python ecosystem, particularly for Data Science and Machine Learning (DS/ML), is a vast, complex, and highly interconnected web of software packages hosted on the Python Package Index (PyPI)[1]. The stability, security, and reproducibility of virtually all modern DS/ML projects depend on this intricate graph of dependencies.

However, this interconnectedness creates structural risks. A bug, security vulnerability, or maintenance failure in a single ”core” package can cascade, potentially breaking thousands of downstream projects. This project aims to move beyond simple metrics to model, analyze, and understand the structural properties of this ecosystem.

Our central research questions are:

- What does the macro-structure of the DS/ML ecosystem look like?
- Which packages are most critical to the ecosystem's function and stability?
- How resilient is this network to the failure of its most critical components?
- Can we identify distinct ”technology stacks” (e.g., visualization, data-wrangling) from the network's community structure?

2. Dataset and Data Collection Methodology

The dataset for this project will be a directed graph, $G = (V, E)$, which will be constructed as a core task of our work. The nodes (V) will represent Python packages, and a directed edge (A, B) will exist if package A lists package B as a dependency.

Data Source

We will collect the data using the public PyPI JSON API.[2] This API provides metadata for each package (e.g., `pypi.org/pypi/pandas/json`) and includes a `requires_dist` field listing its dependencies.

[1] `https://pypi.org/`
[2] `https://docs.pypi.org/api/json/`

Defining the DS/ML Ecosystem (Data Sampling)

To define the boundary of the "DS/ML ecosystem," we will employ a **seed-based crawl**. We will begin with a curated seed set of approximately 15-20 foundational packages (e.g., `pandas`, `numpy`, `scikit-learn`, `tensorflow`, `pytorch`). We will then recursively fetch their dependencies to build a comprehensive graph of the packages that support this ecosystem.

Level of Abstraction of the Graph Dataset

The `requires_dist` field contains complex strings specifying versions, environments, and optional extras. For our analysis, we will use a clear level of abstraction:

1. We will **ignore optional dependencies** (e.g., those marked with `extra == 'test'`).
2. We will **parse only the core package name**, ignoring all version specifiers (e.g., `"numpy>=1.23"` will be parsed as a dependency on `numpy`).

This abstraction allows us to model the fundamental topological structure of the core ecosystem, which is the focus of our project.

3. Project Objectives and Analysis Methodology

We will use methods from the course, implemented primarily with libraries such as NetworkX,[3] to achieve a set of specific objectives. Each method is appropriate for its intended goal.

Objective 1: Identify Core Packages To find the most central and authoritative packages.

Methodology: We will use **directed centrality measures**. **PageRank** is ideal for this, as it measures transitive importance in a directed graph, capturing how "important" a package's dependents are. We will supplement this with **in-degree** (hub) centrality to find packages with the most direct dependents.

Objective 2: Analyze Macro-Structure To understand the ecosystem's high-level organization and dependency flow.

Methodology: We will apply **Strongly Connected Component (SCC) decomposition**. This method is designed to find the "bow-tie" structure (IN, OUT, and CORE components) of a directed graph, which will allow us to visualize the flow of dependencies and identify the core "engine" of the ecosystem.

Objective 3: Discover "Technology Stacks" To find clusters of packages that are frequently used together.

Methodology: We will apply the **Louvain community detection algorithm** to an undirected projection of the graph. The Louvain method is highly suitable as it is a scalable algorithm that optimizes modularity, allowing us to discover densely connected "stacks" (e.g., a visualization stack of `matplotlib`, `seaborn`, etc.).

Objective 4: Assess Ecosystem Resilience To understand how robust the ecosystem is to failures.

Methodology: We will perform **targeted node removal experiments**. We will simulate failures by removing the top-k nodes identified by our centrality analysis (Objective 1) and measuring the impact on network integrity (e.g., the size of the largest weakly connected component). This directly tests the network's resilience to failures in its most critical components.

[3] `https://networkx.org/documentation/stable/reference/index.html`

Objective 5: Baseline Comparison To determine if the findings from Objectives 1, 3, and 4 (centrality, community structure, and resilience) are fundamental properties of the ecosystem network, or if they are artifacts of the network's general topology and depth hierarchy.

Methodology: We will implement a **depth-based randomization** (or **level-based randomization**) method. This approach, often used for Directed Acyclic Graphs (DAGs), creates a randomized null model that preserves the depth or level of each node in the dependency chain. We will then re-run the analyses for Objective 1 (Centrality), Objective 3 (Community Detection), and Objective 4 (Resilience) on this randomized graph. By comparing the results (e.g., centrality rankings, modularity scores, resilience curves) from the real network to this null model, we can validate if our findings are non-trivial or if they can be explained simply by the network's depth-based structure.