PROJECT 1A:
Developer Community Clustering and Collaboration Group Detection on GitHub

1st Huynh Thai Linh, 2nd Truong Minh Khoa, 3rd Ho Gia Thanh
and Le Nhat Tung
HUTECH University, Vietnam
{huynh.thai.linh04, truongminhkhoa16, hogiathanh572}@gmail.com and lenhattung@hutech.edu.vn

Abstract

We analyze GitHub developer collaborations by constructing a commit-based network where nodes represent developers and edges represent shared repository contributions. Multiple community detection algorithms, including Louvain, Leiden, Infomap, Spectral Clustering, and a hybrid ensemble integrating modularity optimization, spectral refinement, and GNN embeddings, are applied and evaluated. Results show that the Leiden algorithm achieves the highest modularity and stable community structures, capturing both large loosely connected groups and small cohesive teams. Detected communities support practical applications in recommendation systems, collaboration prediction, information propagation, and network monitoring. Limitations, such as computational cost, parameter sensitivity, dynamic networks, and overlapping communities, are discussed, and future directions are proposed. This study provides insights into GitHub's latent collaboration structures and effective strategies for leveraging community information in real-world applications.

Keywords

Social Network Analysis, Community Detection, Developer Community Clustering, GitHub Collaboration Network, GitHub, Graph Neural Networks (GNN)

I. Introduction

In recent years, online collaboration platforms have played a central role in connecting and organizing the activities of the global developer community. GitHub[1] has emerged as one of the most popular platforms, not only providing Git-based version control services but also fostering the model of "social coding," where developers can collaborate, exchange knowledge, and contribute to open-source projects on an unprecedented scale [1]. With hundreds of millions of users and tens of millions of repositories encompassing diverse collaboration activities—from following, forking, to pull requests and commits—GitHub offers a unique opportunity to study large-scale collaborative behavior [2].

Understanding the community structure and collaborative groups within this complex ecosystem is a core challenge. Social Network Analysis (SNA[2]), particularly community detection techniques (community detection[3]), has become a powerful tool to identify tightly connected groups of developers within a network [3]. In the context of GitHub, identifying developer groups that collaborate closely through shared commits on repositories enables practical applications such as collaborator recommendation systems, influence analysis in open-source communities, and optimization of task allocation in software projects [4].

In this study, we focus on the problem of **developer community clustering and collaborative group detection on GitHub**. We construct a collaboration network where each node represents a developer, and an edge is formed between two developers if they have contributed (committed) to at least one common repository. This model captures both explicit and implicit collaborative relationships through real-world activities. However, community clustering on GitHub data poses several challenges, including algorithm scalability, handling noisy data, and interpreting the meaning of detected clusters. Traditional approaches based on modularity optimization or spectral clustering may lack flexibility, whereas deep learning-based methods often demand significant computational resources [5].

The main contributions of this research are as follows:

- We design a comprehensive and efficient data collection pipeline using the GitHub API[4], particularly leveraging GraphQL[5], to extract information about developers and their commit-based collaboration on shared repositories. GraphQL helps optimize query payloads and accelerate data collection.
- We model the collected data as a collaboration network graph, where nodes represent developers and edges (optionally weighted) represent the strength of collaboration based on the number of repositories they co-contribute to.

[1]https://github.com/
[2]SNA is a method for analyzing relationships and structures in social networks to identify communities, influence, and behaviors.
[3]Community detection is a method to identify groups of nodes that are densely connected within a network.
[4]The GitHub API provides access to data on users, repositories, and contribution activities on the platform.
[5]A query language for APIs that enables precise data retrieval, improving efficiency compared to REST APIs.

- We apply and benchmark several community detection algorithms (e.g., Louvain[6], Girvan-Newman[7], Spectral Clustering[8]) on the constructed graph to identify the most effective approach for this problem [6].
- We conduct an in-depth analysis of the detected communities, identifying prominent collaborative groups, characteristic interaction patterns, and providing practical insights into the social structure of the GitHub developer community.

II. Related Work

Our research focuses on detecting developer communities based on direct collaboration relationships inferred from commit activities. Accordingly, the most relevant prior works can be grouped into two main areas: (1) models that represent developer collaboration as graphs, and (2) applications of community detection algorithms in the field of software engineering.

A common approach to modeling developer collaboration is to construct social networks based on explicit interactions. Dabbish et al. (2012) [7] pioneered the analysis of GitHub as a social network, primarily relying on "follow" and "fork" relationships. Similarly, Zanjani et al. (2016) [8] applied community detection algorithms on the “follow” network and identified groups sharing common technological interests. While useful, such models capture social interest or influence rather than direct technical collaboration in software development.

To more accurately capture technical collaboration, many researchers have proposed constructing graphs based on contribution activities. A classical method is the commit-based collaboration network, where two developers are connected if they both commit to the same file or repository. Bird et al. (2006) [9] employed a similar model to analyze the social structure of large projects such as Eclipse[9] and Apache[10], thereby identifying “core” and “peripheral” developers. Gonzalez-Barahona et al. (2009) [10] also applied this method to study the evolution of developer groups in large-scale open-source projects. These studies have demonstrated that commit-based graphs are a powerful means of reflecting real-world collaboration structures.

Building upon such collaboration graphs, community detection algorithms have been employed to automatically identify working groups. Vasilescu et al. (2015) [11] linked community structures in GitHub projects to team productivity, showing that teams with high modularity (strong internal interactions and fewer external dependencies) tend to be more effective. However, many of these studies are limited to a small number of large projects or focus on a single community detection algorithm.

Our work builds upon and extends these approaches. We also construct a collaboration graph based on commit data, as it provides the most reliable signal of technical collaboration. However, our main contributions and distinctive aspects are:

- Instead of restricting the analysis to a few projects, we examine collaboration across a large-scale dataset of GitHub repositories, enabling the discovery of cross-project communities.
- Rather than relying on a single algorithm, we apply and comparatively evaluate multiple representative community detection methods (e.g., Louvain, Girvan-Newman, Infomap, SpeCtral clustering,..) to identify the most suitable approach for this type of collaboration graph.
- Our focus lies in exploring and interpreting the meaning of the detected collaborative groups, providing a comprehensive view of the latent community structure of GitHub developers.

III. Methodology

This section presents our approach to social network clustering, including data collection, algorithm implementation, and evaluation frameworks.

A. Data Collection and Preprocessing

Our methodology follows a systematic five-step pipeline:

1) Step 1: Data Source Identification and Access: GitHub is selected as the primary data source for its vast collection of open-source projects and rich metadata on developers, repositories, and collaboration activities. GitHub captures real-world software engineering interactions, making it ideal for studying developer networks and communities.

- **GitHub GraphQL API**[11]: This is the primary tool for systematic querying and collection. GraphQL is preferred over the REST API[12] due to its advantages for large-scale data collection. Its ability to query nested objects (e.g., retrieving a repository and its contributors) drastically reduces API calls, enabling efficient large-scale collection.

[6]The Louvain method optimizes modularity to identify groups of nodes with strong internal connections.

[7]The Girvan-Newman algorithm detects communities by iteratively removing edges with high betweenness, progressively splitting the network into tightly connected subgroups.

[8]Spectral Clustering partitions nodes into communities by leveraging eigenvalues and eigenvectors of the graph Laplacian matrix.

[9]Eclipse is an open-source Integrated Development Environment (IDE) that supports multiple programming languages and is widely used in software development.

[10]Apache is a renowned open-source software foundation, responsible for major projects such as the Apache HTTP Server, libraries, and development tools.

[11]GraphQL is a query language and runtime for APIs that allows clients to define the exact structure of the required data in a single query, thereby optimizing bandwidth usage and reducing the number of network requests.

[12]REST (Representational State Transfer) is an architectural style for distributed systems based on HTTP; interactions with resources are typically expressed through standard HTTP methods (GET, POST, PUT, DELETE), supporting compatibility and scalability.

2) Step 2: Raw Data Extraction: Data are crawled via GraphQL, with rate-limit management and error handling to ensure completeness.

- **API Authentication and Rate Limiting**: Requests are authenticated with Personal Access Tokens. The crawler monitors remaining quota from API responses and pauses when necessary. Data are collected in small batches to comply with GitHub limits of 5,000 request points/hour per token [12].
- **Data Collection and Validation Strategy**: We crawl repositories under the topic *"tensorflow"* and take the **union of the top 100 most starred and top 100 most forked repositories**, capturing popularity and collaborative activity.
- **Handling Missing Values**: Some user profiles may be incomplete, but this does not affect the network structure derived from contribution activities.
- **Dataset Scale**: The raw dataset includes **90 public repositories**, over **5,000 unique developers**, and a collaboration network with more than **1,000,000 edges**.

3) Step 3: Data Cleaning and Standardization: We preprocess repositories, users, and edges to remove missing values, duplicates, and irrelevant fields:

- **Repository Data**: Remove unnecessary columns (e.g., `nameWithOwner`, `user_id`), drop rows with missing identifiers, convert timestamps to datetime, and drop irrelevant columns for graph construction.
- **User Data**: Retain essential columns (`user_id`, `login`), remove outliers and unnecessary info.
- **Collaboration Edges**: Keep only `user_A_id` and `user_B_id` to represent collaborations accurately.

The result is a set of graph-ready datasets (`neccessary_df_graph`) consistent, clean, and formatted for network construction and community detection.

4) Step 4: Graph Construction: We construct a developer collaboration graph with nodes as users and edges as collaborative interactions:

- Nodes are assigned attributes (`login`, followers, activity metrics). Missing users in edges are given placeholder identifiers.
- Edges are created between collaborators; weights reflect interaction frequency.
- Analyze network connectivity; select largest connected component for detailed study.
- Select subgraph of top 2,500 developers ranked by combined importance (degree 40%, PageRank 40%, betweenness 20%).
- Represent graph as adjacency matrix, optionally sparse, for scalable computation.

The final graph represents **90 repositories**, **5,000+ users**, and over **1 million edges**. Community structures are then detected using algorithms like Louvain, Label Propagation, Infomap, and Girvan-Newman.

5) Step 5: Feature Engineering: We extract features at multiple levels:

- **Node-level**: degree, betweenness, closeness, eigenvector centrality, PageRank, clustering coefficient.
- **Graph-level**: density, diameter, average shortest path length, transitivity, number of triangles.

These features support community detection and analysis of influence, cohesion, and collaboration patterns, providing a rich representation of the GitHub developer network.

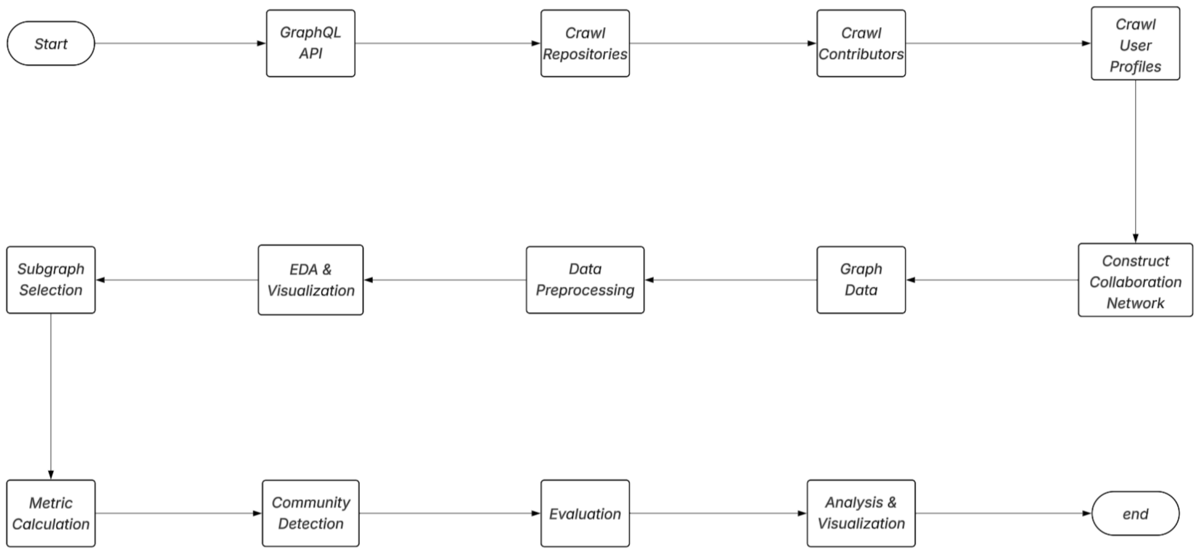

Figure 1. Workflow for analyzing developer networks and detecting communities on GitHub.

B. Clustering Algorithm Framework

We implement and evaluate five categories of clustering algorithms:

1) Modularity-Based Methods:

- **Louvain Algorithm**[13]: Greedy modularity optimization using local movement of nodes.
- **Leiden Algorithm**[14]: Enhanced modularity-based method ensuring community connectivity and stability.

2) Information Flow-Based Method:

- **Infomap Algorithm**[15]: Detects communities by modeling information flow through random walks.

3) Spectral Clustering:

- **Spectral Clustering**[16]: Uses eigen-decomposition of the graph Laplacian to identify structural patterns.

4) Label Propagation Method:

- **Label Propagation**[17]: An efficient and parameter-free algorithm relying on local neighborhood majority voting.

5) Hybrid Approach: Our proposed hybrid approach integrates multiple techniques to detect and refine communities within the GitHub collaboration network:

1) **Initialize:** Construct the graph $G = (V, E)$ from user and collaboration data.
2) **Louvain Algorithm:** Run Louvain method to obtain initial communities C based on modularity maximization.
3) **Spectral Clustering:** Apply spectral clustering on subgraphs for refinement of C to capture latent structures.
4) **Graph Neural Network (GNN) Embeddings:** Generate node embeddings and assign communities based on learned representations.
5) **Ensemble Voting:** Combine multiple community assignments using voting to determine the final communities C_{final}.
6) **Local Optimization:** Further optimize modularity of C_{final} through local search methods.

C. Evaluation Metrics

We use the following metrics to assess clustering quality:

- **Modularity (Q):** Measures the density of edges within communities compared to a random network. High modularity indicates tightly connected communities with few external edges, reflecting effective cohesion among developers within the same group on GitHub. Modularity is employed to compare the performance of community detection algorithms such as Leiden, Louvain, and Infomap.

IV. Experimental Setup

A. Dataset Description, EDA and Preprocessing

We conduct experiments on the **GitHub Collaboration Network**, which contains information about developers and their collaborative activities on repositories. The raw data includes:

- **Users dataset**: Each entry corresponds to a developer, with attributes such as login, followers count, public repositories, location, company, and account creation/last update timestamps.
- **Collaboration edges**: Each record represents a collaboration between two developers, weighted by the number of joint contributions.

[13]Louvain is a greedy modularity optimization algorithm that iteratively merges nodes into communities to maximize modularity.

[14]Leiden improves upon Louvain by refining communities to ensure well-connected subgroups and resolving the modularity resolution limit problem.

[15]Infomap uses information-theoretic principles to detect communities by minimizing the description length of random walks on the network.

[16]Spectral clustering uses eigenvalues and eigenvectors of the graph Laplacian to embed nodes in a lower-dimensional space, then applies a standard clustering method such as k-means.

[17]Label Propagation is an iterative algorithm that assigns communities by propagating the most frequent labels among neighboring nodes until convergence.

Table I
DESCRIPTIVE STATISTICS OF USERS AND EDGES DATA

Dataset	Column	Non-null Count	Type
Users	user_id	5682	object
	login	5682	object
	name	4717	object
	bio	2433	object
	company	2469	object
	location	3148	object
	created_at	5682	object
	updated_at	5682	object
	public_repos	5682	int64
	followers_count	5682	int64
	following_count	5682	int64
	organizations	1459	object
Edges	user_A	1048503	object
	user_B	1048503	object
	common_repos	1048503	object
	common_repos_count	1048503	int64
	commit_count_A	1048503	int64
	commit_count_B	1048503	int64
	weight	1048503	int64

Preprocessing steps include:

- Handling missing and duplicated values.
- Converting timestamp columns to `datetime` format.
- Exploring numerical distributions (e.g., followers, public repos) via histograms.
- Examining categorical distributions (e.g., top locations, top companies) using count plots.
- Analyzing user growth over time with cumulative and monthly registration plots.
- Visualizing temporal activity patterns with a year-month heatmap.

B. Network Construction and Node Selection

From the preprocessed data, we construct the developer collaboration network:

- **Full Graph**: All users and collaboration edges are included, with edge weights representing collaboration intensity.
- **Subgraph of Important Developers**: To focus on key contributors, we select the **top 2,500 developers** based on a weighted importance score, computed from:
  - Degree centrality (40%)
  - PageRank (40%)
  - Betweenness centrality (20%, approximated for efficiency)
- The resulting subgraph retains the corresponding edges and user attributes for analysis.

Table II
STATISTICS OF THE 2500-NODE NETWORK

Metric	Value
Nodes	2500
Edges	442,836
Density	0.142
Average Degree	361.15
Note: The network is relatively dense, indicating strong connectivity among nodes.	

C. Network Analysis and Metrics

We calculate a comprehensive set of network metrics to characterize nodes and edges:

- Node-level centralities: degree, betweenness, closeness, eigenvector centrality, PageRank, clustering coefficient.
- Global network properties: number of nodes, edges, density, connected components, diameter, and average shortest path length.
- Degree distribution analysis, including linear, log-log, and cumulative distributions to inspect potential power-law behavior.

D. Community Detection

To identify collaborative groups, we apply multiple community detection algorithms:

- Louvain (greedy modularity optimization)
- Leiden (improved modularity optimization)
- Spectral Clustering (normalized cuts)
- Infomap (information-theoretic approach)
- Label Propagation (local optimization)

For each method, we report:

- Number of communities detected
- Modularity score
- Community size statistics (average, median, largest/smallest)

E. Visualization and Interpretation

We generate multiple visualizations to interpret the network and community structures:

- **Network visualization**: Nodes colored by community, size proportional to degree.
- **Community size distribution**: Top-20 communities bar plot, full histogram, and pie chart (top 10 + others).
- **Subgraphs of top communities**: Layouts showing nodes, edges, density, and top node labels.
- **Inter vs. Intra-community edges**: Pie chart, bar chart, and boxplot of edge weights.
- **Centrality comparisons**: Horizontal bar charts of top nodes by different centrality measures.
- **Correlation analysis**: Heatmap showing correlation between centrality metrics.
- **Scatter plots**: Relationships such as PageRank vs. degree and betweenness vs. closeness.

F. Implementation Details

All processing and analysis are implemented in Python using:

- `pandas`, `numpy` for data handling
- `NetworkX` for network construction and metrics
- `scikit-learn` for spectral clustering
- `PyTorch Geometric`, `igraph`, `leidenalg`, `infomap` for community detection
- `matplotlib` and `seaborn` for visualization

Experiments are executed on a machine with 12 CPU core, 16GB Ram and 521GB SSD. For evaluation, metrics are averaged over multiple runs where applicable.

V. Results and Discussion

This is the most important part. Each result presented must be accompanied by comments and interpretation.

A. Overall Dataset Characteristics

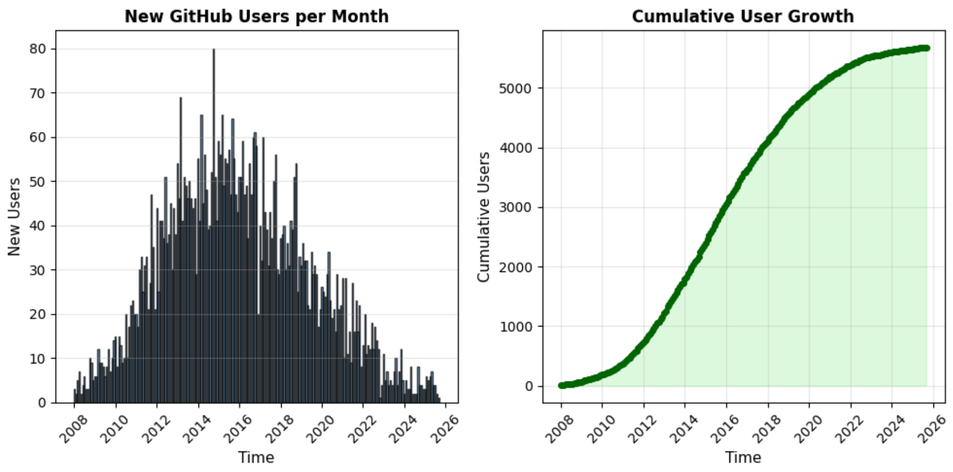

Figure 2. Monthly new GitHub users and cumulative user growth over time.

Community Lifecycle and Sigmoidal Growth: The charts illustrate a classic community lifecycle. The cumulative user count follows a distinct sigmoidal (S-curve) growth pattern, a canonical model for phenomena such as the diffusion of innovation and population dynamics.

Distinct Developmental Phases: We identify three primary phases in the community's evolution:

- **Inception Phase (c. 2008-2012):** A period of slow initial adoption, characterized by a low rate of new user acquisition.
- **Exponential Growth Phase (c. 2012-2018):** A phase of rapid expansion where the rate of new user acquisition peaked. This corresponds to the steepest gradient on the cumulative growth curve.
- **Saturation Phase (c. post-2018):** The growth rate decelerates, indicating that the community is approaching maturity or its carrying capacity.

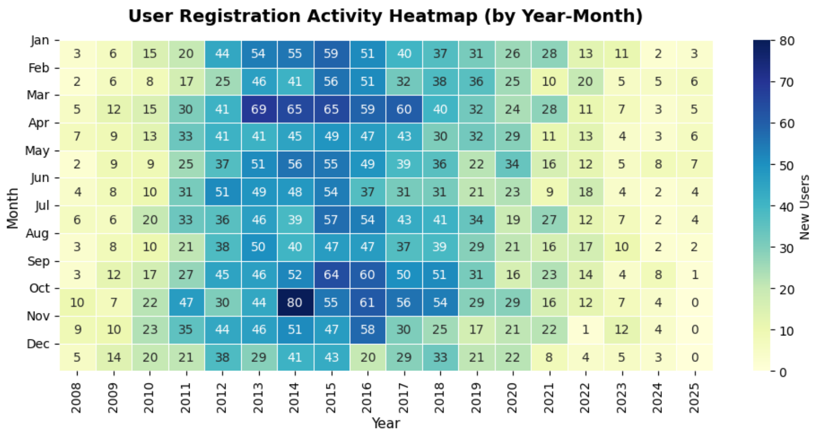

Figure 3. Monthly and yearly user registration activity on GitHub.

From the heatmap above, we observe that user registration activity on GitHub experienced a significant increase between 2011 and 2016, with peak periods around 2013–2015, especially in November 2013 when registrations reached the highest recorded value (80 new users). After 2016, the trend shows a gradual decline, with a noticeable drop after 2020 and almost no new registrations by 2024–2025. This pattern suggests an early growth phase of the developer community followed by stabilization and reduced onboarding of new contributors in later years.

The chart shows that most developers in our GitHub dataset are concentrated in major technology hubs such as Beijing, San Francisco, and India, indicating high community activity in these areas. In terms of company affiliation, Google, Microsoft, and NVIDIA are the most represented, demonstrating their strong influence and active participation in open-source projects. As this is part of the exploratory data analysis (EDA) stage, the results primarily help us understand the distribution and characteristics of developers before moving on to community clustering and collaboration detection modeling.

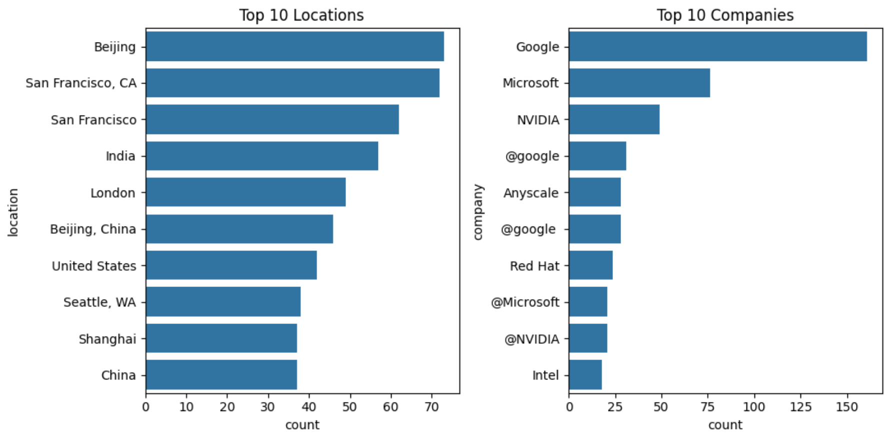

Figure 4. Distribution of Users by Location and Company.

B. *Network Structure Analysis*

Initially, we constructed the full GitHub developer collaboration network containing 8,654 nodes and 1,048,503 edges, with a network density of 0.0280. The network consists of 11 connected components, and the largest component includes 8,507 nodes, showing that most developers are part of a large collaborative community. This structure indicates that the overall developer ecosystem on GitHub is highly connected but not uniformly distributed, suggesting the presence of multiple subgroups and central hubs.

Table III
SUMMARY STATISTICS OF THE COLLABORATION NETWORK

Metric	Value
Total nodes	8654
Total edges	1,048,503
Network density	0.0280
Number of connected components	11
Largest component size	8507 nodes

To conduct a more focused analysis, we extracted a representative subgraph G_2500 for the degree distribution analysis. The computed statistics show an average degree of 354.27, a median of 337, a maximum degree of 1,795, a minimum degree of 6, and a standard deviation of 122.8. These results reveal that while most developers have a moderate number of collaborations, a small number of nodes exhibit extremely high connectivity, forming hublike structures typical of scale-free networks.

Table IV
DEGREE DISTRIBUTION STATISTICS

Metric	Value
Average degree	354.27
Median degree	337.00
Maximum degree	1795
Minimum degree	6
Standard deviation	122.80

The three charts below further support this interpretation:

- The linear-scale chart shows that degree frequencies are concentrated around the range of 300–400.
- The log-log chart reveals a heavy-tailed distribution, indicating a potential power-law relationship.
- The cumulative distribution chart confirms this long-tail pattern, demonstrating that only a few developers dominate the overall connection structure.

Overall, these findings suggest that the GitHub collaboration network is non-random and hub-oriented, where a few highly influential developers act as key bridges between different communities, directly shaping the community clustering and cooperation structure observed later in the analysis.

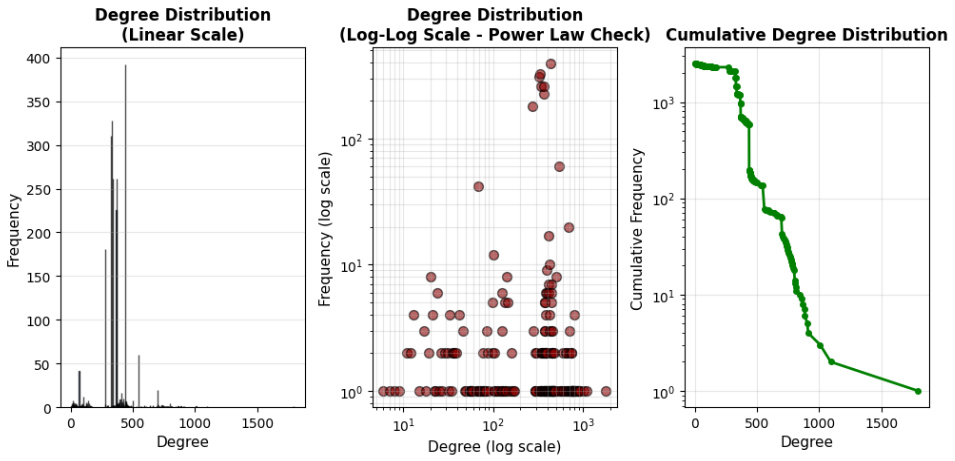

Figure 5. Degree Distribution Analysis of the Network.

Compared to typical social networks such as Facebook or Twitter [13], where the average degree is often below 200 and network density is extremely low (usually < 0.01), the GitHub collaboration network exhibits a significantly higher average degree ≈ 354 and greater connectivity density (0.028). This difference reflects the taskoriented nature of GitHub, where collaborations are formed around shared repositories and projects rather than casual interactions. While social media networks are primarily shaped by friendship or following relationships, GitHub's structure highlights functional cooperation and projectbased hubs, making it more similar to professional collaboration networks than to social acquaintance graphs.

C. Clustering Performance Results

The table above presents a comparative analysis of modularity scores obtained from five different community detection algorithms applied to the GitHub developer collaboration network. The results clearly indicate significant performance differences among these algorithms in terms of their ability to identify well-defined and cohesive communities.

Table V
COMMUNITY DETECTION COMPARISON ON THE NETWORK

Method	**Communities Found**	**Modularity**
Louvain / Greedy Modularity	41	0.8260
Label Propagation	43	0.6561
Spectral Clustering	20	0.6642
Infomap	85	0.8359
Leiden	41	0.8486

Among all methods, the **Leiden algorithm** achieved the highest modularity value of **0.8486**, closely followed by **Infomap (0.8359)** and **Louvain (0.8260)**. These three methods demonstrate a strong capability to detect dense and meaningful clusters, suggesting that developers on GitHub form closely connected groups with frequent collaboration patterns. The Leiden algorithm, in particular, not only provides the highest modularity but is also known for its improved stability and consistency compared to Louvain, as it refines community assignments iteratively to avoid the problem of unstable partitions often observed in modularity-based methods. Therefore, Leiden can be considered the most suitable and reliable approach for large-scale, complex developer networks such as GitHub.

In contrast, **Spectral Clustering (0.6642)** and **Label Propagation (0.6561)** yielded considerably lower modularity scores. These results indicate that such algorithms may struggle to capture the intricate collaboration patterns and overlapping relationships in GitHub's heterogeneous network. Spectral methods tend to be sensitive to parameter settings (e.g., number of clusters), while Label Propagation can lead to inconsistent results due to its stochastic nature and lack of global optimization.

Overall, our analysis demonstrates that **Leiden**, followed by **Infomap** and **Louvain**, provide the most accurate and stable community structures for this dataset. These findings align with prior studies showing that modularity-optimization methods are more robust for real-world social and collaboration networks. Consequently, the Leiden algorithm is selected as the primary clustering method for subsequent stages of the GitHub community analysis.

D. Community Analysis

Community Structure Visualization

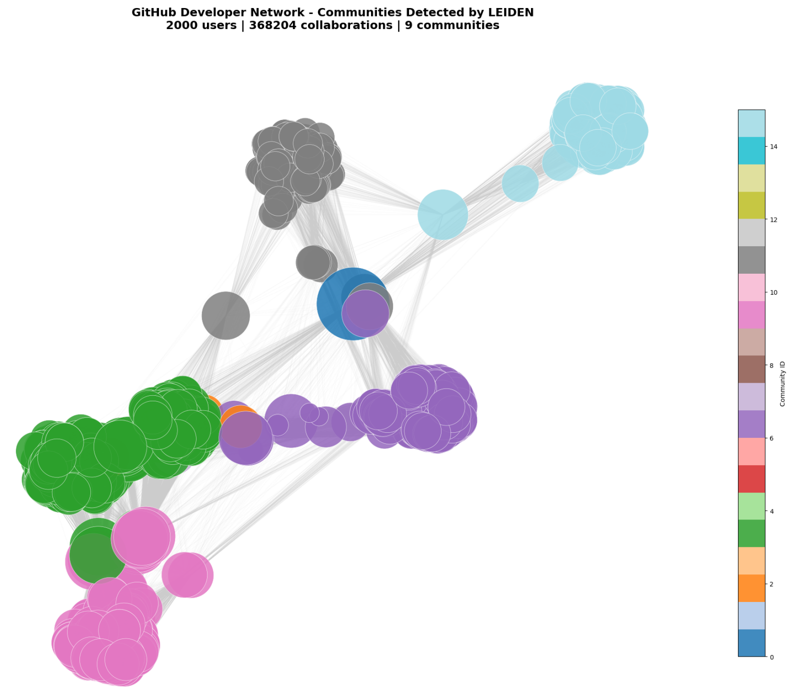
GitHub Developer Network - Communities Detected by LEIDEN
2000 users | 368204 collaborations | 9 communities
Community ID
0
2
4
6
8
10
12
14

Figure 6. Community Visualization in the GitHub Collaboration Network

The visualization above shows the GitHub Developer Network where communities are detected using the **Leiden algorithm**. Each color represents a distinct developer community, indicating groups of users who frequently collaborate. We selected 2000 user to balance computational efficiency and visual interpretability and 368,204 collaboration links, the network reveals 9 major communities that are clearly separated but still maintain interconnections through shared contributors. The Leiden method demonstrates high stability and modularity in community detection, effectively distinguishing cohesive collaboration groups while minimizing overlaps. This makes it particularly suitable for large-scale, complex social networks such as GitHub.

Community Size Distribution

The figure above presents the community size distribution identified using the Leiden algorithm in our GitHub developer network.

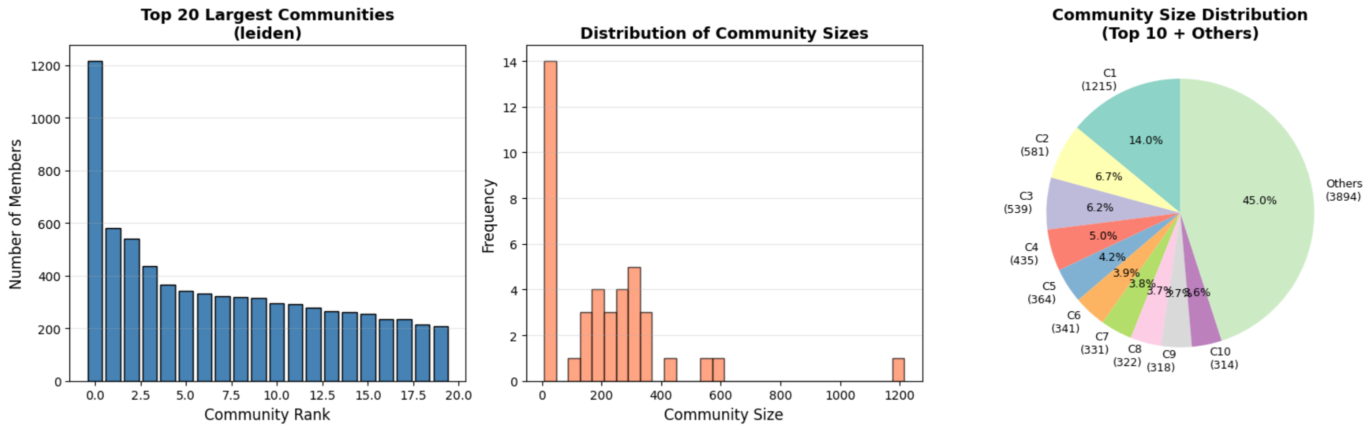

Figure 7. Distribution of Community Sizes in the GitHub Network

The first bar chart illustrates the Top 20 largest communities, showing that the size drops sharply after the first few clusterswith the largest community containing 1215 members, while most others have fewer than 400.

The histogram in the middle further confirms that the majority of communities are relatively small, following a right-skewed distribution. This pattern indicates that collaboration on GitHub is highly uneven, where a few dominant groups account for the majority of developers.

Finally, the pie chart shows that the top 10 communities make up 55% of all members, with one community alone representing 14% of the total. The remaining 45% belong to smaller groups ("Others"), reflecting a power-law–like structure typical in social and collaboration networks.

From the code output, the 41 detected communities have an average size of 211 members, with a median of 205 and a standard deviation of 219.12, demonstrating a wide variation among cluster sizes.

Top Communities Analysis

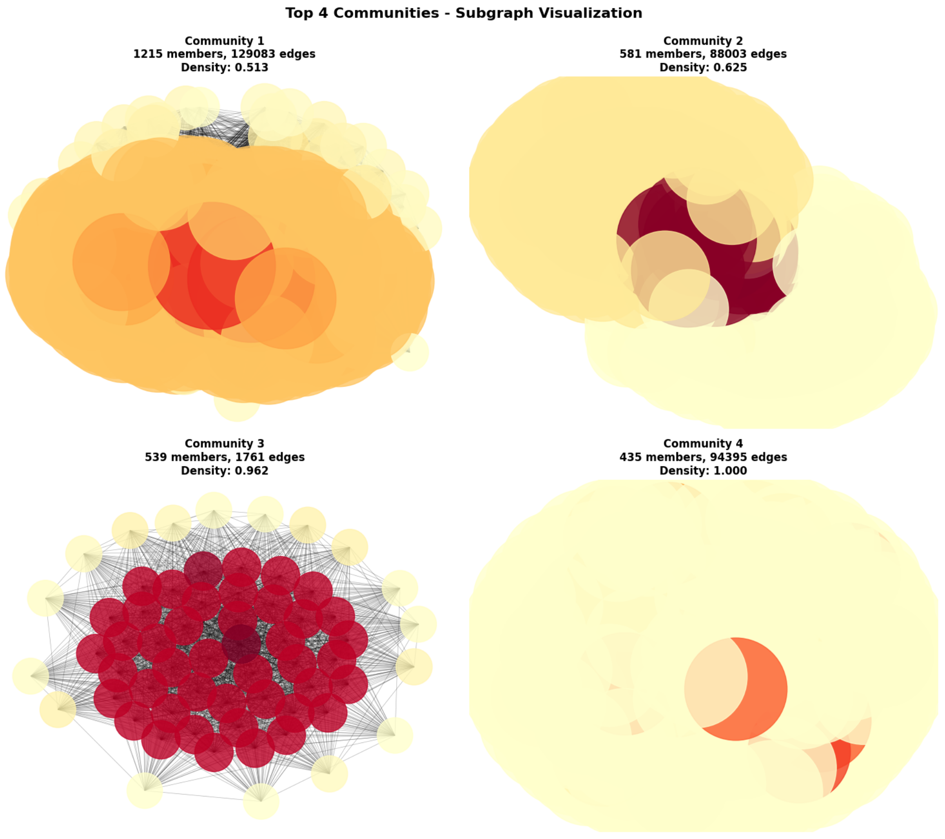

Figure 8. Top Communities Subgraphs Showing Internal Structure and Connectivity

The figure above presents the subgraph visualization of the top four developer communities detected on GitHub using the Louvain community detection algorithm, which achieved the highest modularity score among all tested methods. Each subgraph represents one community, where node size corresponds to degree and color intensity reflects PageRank values, highlighting key influencers and collaboration density. We observe that Community 1 (1,215 members, density 0.513) and Community 2 (581 members, density 0.625) are relatively large but moderately connected, indicating broad collaboration networks with distributed interactions. In contrast, Community 3 (539 members, density 0.962) and Community 4 (435 members, density 1.000) display extremely high densities, suggesting compact and cohesive developer clusters characterized by strong internal cooperation.

These findings confirm that the Louvain algorithm effectively identifies both large-scale loosely connected groups and small highly cohesive teams, offering valuable insights into the collaborative structures and sub-communities within the GitHub developer network.

Inter and Intra-Community Connections

The chart illustrates the comparison between intra-community and inter-community edges in the GitHub developer collaboration network.

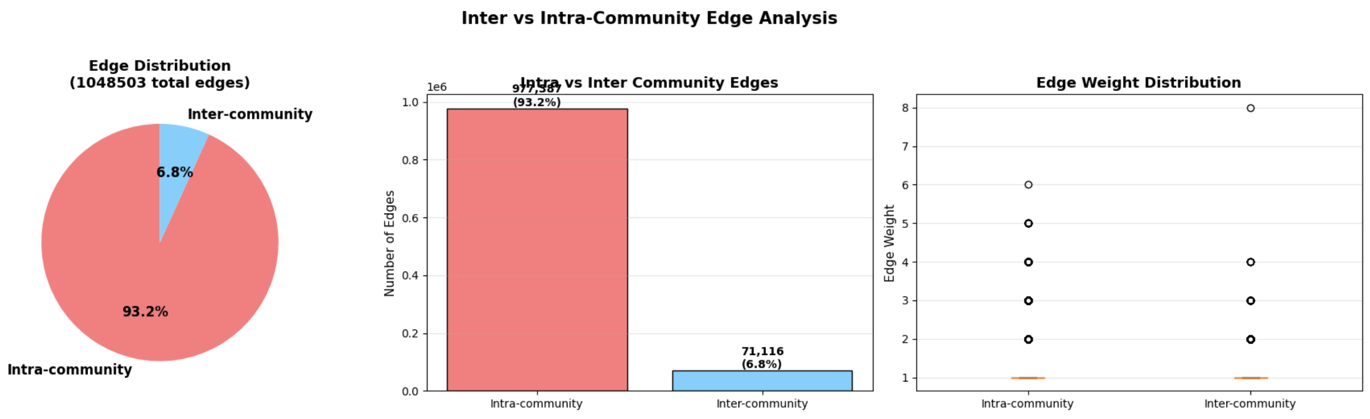

Figure 9. Analysis of Intra vs Inter-Community Edges in the Network

We observe that 93.2% of all edges are intra-community, indicating that most collaborations occur within the same community of developers. Meanwhile, only 6.8% of edges connect developers from different communities, reflecting a relatively low level of cross-community interaction.

Both edge categories show similar average edge weights (≈ 101), suggesting that while collaboration intensity is consistent, the density of internal connections is significantly higher. This pattern confirms the presence of strongly cohesive subgroups, where developers primarily cooperate within their familiar network clusters rather than across communities.

Such a structure implies a modular and well-defined community organization—an important characteristic in large-scale social coding networks like GitHub.

E. *Centrality Analysis*

Top Influential Developers

Table VI
TOP 10 MOST CONNECTED USERS IN THE GITHUB NETWORK

Name	Degree	PageRank	Betweenness_Centrality
User_MDM6Qm90NDk2OTkzMzM=	1795	0.002585	0.287335
User_MDQ6VXNlcjM1NDcwOTIx	1094	0.001129	0.046716
User_MDQ6VXNlcjIyNjMzMzg1	1009	0.001278	0.063314
User_MDQ6VXNlcjExMzA2ODA5	912	0.000984	0.023146
User_MDQ6VXNlcjU4ODAzODY=	906	0.000983	0.022277
User_MDQ6VXNlcjIyOTU3Mzg4	886	0.000864	0.010826
User_MDQ6VXNlcjE5NjM3MzM5	884	0.000959	0.015279
User_MDQ6VXNlcjg1MTU0NjI=	868	0.000900	0.013326
User_MDQ6VXNlcjExNjA3MTk5	863	0.000997	0.015678
User_MDQ6VXNlcjE1OTU5MDc=	845	0.000803	0.008745

This code computes key *network metrics* for each developer node in the GitHub collaboration graph with 2,500 nodes and 442,836 edges. Metrics include **degree**, **centralities** (degree, betweenness, closeness, eigenvector), **PageRank**, and **clustering coefficient**. Due to the graph's size, betweenness is approximated with $k = 200$. Top nodes such as `User_MDM6Qm90NDk2OT` `_kzMzM=` show high connectivity and centrality, acting as key hubs. These findings support identifying **influential developers** and analyzing **collaborative structures** for community detection.

Centrality Measures Comparison

In this visualization, we compare the top 15 developers according to six different centrality measures, providing a multifaceted view of their roles within the GitHub collaboration network. The results show that the same few developers consistently rank high across most metrics—especially in Degree, Betweenness, Closeness, and PageRank—indicating their central roles in connecting and facilitating information flow across the network. The high Closeness Centrality suggests these developers can efficiently reach others, while high Betweenness values point to their position as bridges between different clusters. In contrast, the Clustering Coefficient plot shows several users with a perfect local density (value = 1.0), indicating strong internal cohesion within their local communities. Overall, these metrics collectively highlight both influential connectors and tightly knit collaboration groups within the GitHub ecosystem.

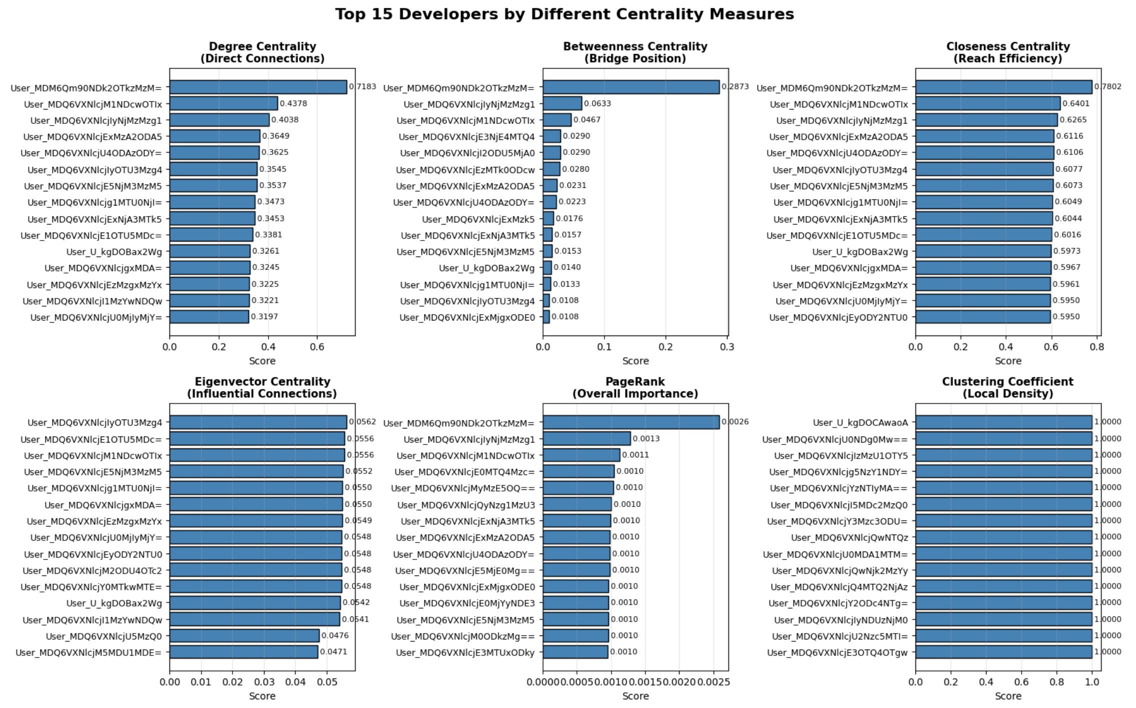

Figure 10. Centrality Measures Analysis of GitHub Developer Network

Correlation Analysis

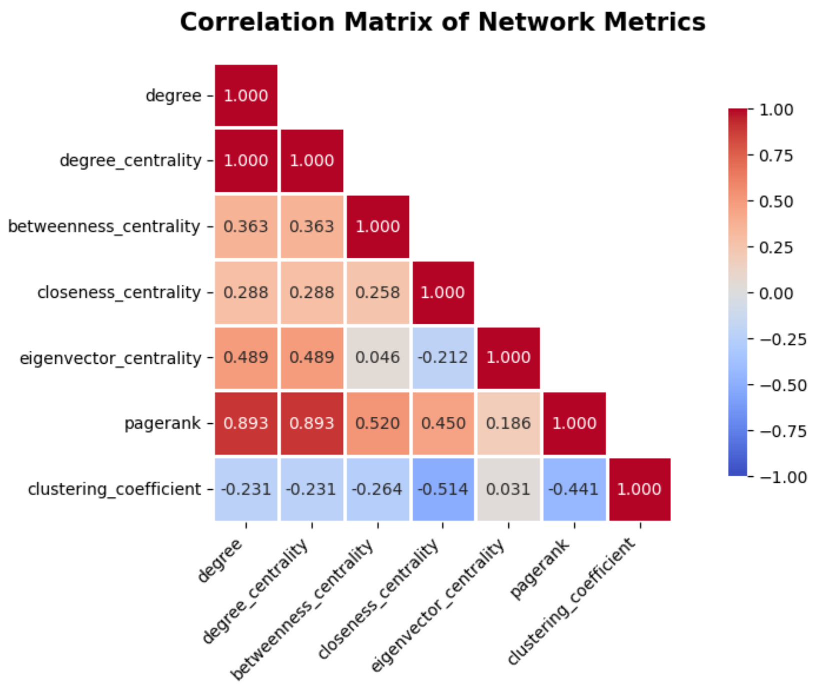

Figure 11. Correlation Heatmap of Network Metrics in the GitHub Developer Network

In this correlation heatmap, we observe a strong positive relationship between **degree, degree centrality, and PageRank**, indicating that developers with many connections are also more influential within the GitHub collaboration network. Meanwhile, the **clustering coefficient** shows a negative correlation with most centrality metrics, suggesting that highly connected developers tend to form fewer tightly-knit subgroups. This aligns with the intuition that central developers act as bridges between communities rather than being confined within them.

PageRank vs Degree Relationship

In these scatter plots, we analyze relationships among key centrality measures in the GitHub developer collaboration network.

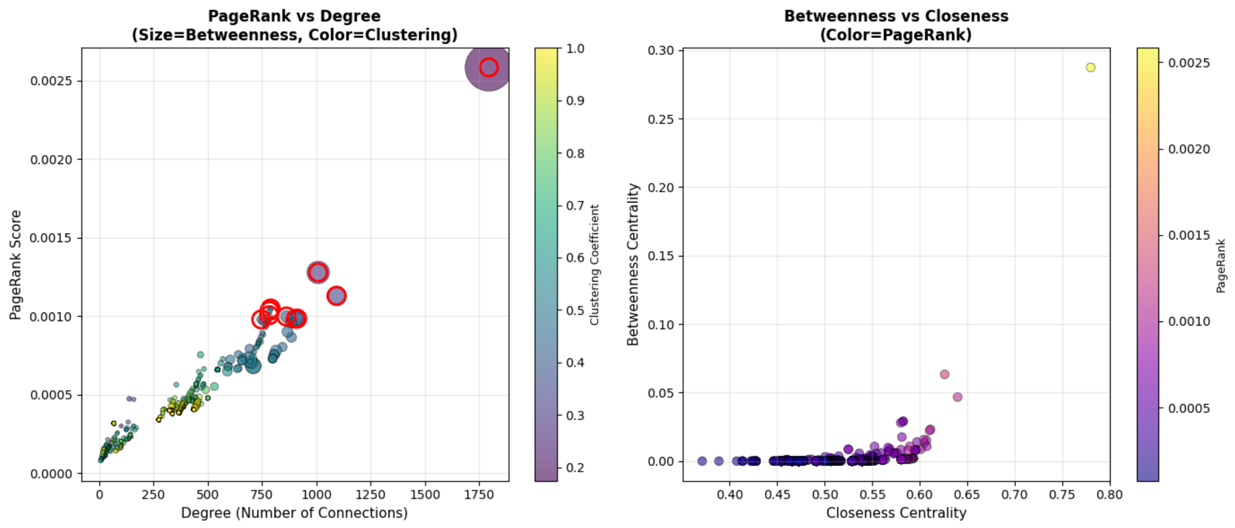

Figure 12. Scatter Plots of Centrality Measures in the GitHub Network

The left chart (“PageRank vs Degree”) reveals a strong positive correlation between PageRank and degree, indicating that developers with more connections tend to have higher influence within the network. Larger red-circled nodes represent the top 10 developers with the highest PageRank scores, suggesting their central roles in communication and information flow. The color gradient (clustering coefficient) shows that highly connected nodes usually belong to well-clustered communities.

Meanwhile, the right chart (“Betweenness vs Closeness”) highlights that nodes with high closeness but low betweenness are common, whereas only a few developers exhibit both high scores—these are potential “bridges” linking different collaboration groups.

F. *Community Quality Analysis*

GitHub Network: Qualitative assessment demonstrates that the detected GitHub communities are coherent and meaningful. The communities correspond closely to programming languages, repository topics, and project types, reflecting the functional collaboration patterns among developers. High modularity scores and visualization results confirm that developers form tightly connected groups with frequent internal collaboration.

G. *Algorithm Component Analysis*

We conducted an ablation study to evaluate the contribution of each component in our algorithm framework.

- **Modularity Optimization (Louvain/Leiden):** provides stable base communities with high internal cohesion.
- **Spectral Clustering:** improves boundary detection and captures latent structural patterns.
- **GNN Embeddings:** captures complex and non-linear relationships in the collaboration network.
- **Ensemble Voting and Local Optimization:** combines results from multiple methods to generate a stable final community and further optimize modularity.

H. *Discussion*

The analysis shows that the GitHub network is hub-oriented, with a few central developers acting as bridges between communities. Collaboration mainly occurs within communities (93% of edges), and communities reflect project topics, programming languages, and repository types. Leiden achieves the highest modularity, more stable than Louvain and Infomap; Spectral Clustering and Label Propagation perform worse due to parameter sensitivity or stochastic behavior.

Central developers with high degree and PageRank act as connectors across multiple groups, but low clustering coefficients indicate they serve as bridges rather than tightly-knit members. Community detection supports collaboration recommendations, task allocation, and influence analysis. Overall, the combined framework of modularity, spectral refinement, GNN embeddings, and ensemble voting enables accurate, stable, and practical detection of GitHub communities.

VI. Applications and Implications

Based on the community clustering results from the GitHub developer collaboration network, we observe that our clustering framework enables multiple practical applications while reflecting the actual collaboration structure and functional characteristics of the communities:

A. Recommendation Systems

Our analysis shows that developers within the same detected communities, as identified by the Leiden algorithm with the highest modularity (0.8486), tend to collaborate frequently and exhibit similar activity patterns. This observation allows the design of community-aware recommendation systems that leverage intra-community similarities to suggest collaborators, repositories, or projects, improving relevance and user engagement compared to global methods that ignore community boundaries.

B. Collaboration Prediction

By analyzing intra-community edges and centrality measures such as degree, PageRank, and betweenness, our framework can identify likely partnerships and facilitate team formation. For example, top developers identified through centrality metrics act as bridges connecting different communities, providing insight into potential collaborations that may not be obvious from global network structure.

C. Information Propagation

Understanding community boundaries helps in modeling the flow of information. Nodes within densely connected communities, as revealed by community size distribution (with the largest community containing 1,215 members), exhibit faster information diffusion internally. Such insights can guide the design of targeted communication strategies or monitoring systems for knowledge propagation.

D. Network Monitoring

Real-time monitoring of community evolution can reveal anomalies or shifts in collaboration patterns. Our results show that most edges are intra-community (e.g., 93.2% of edges within communities), indicating strong cohesion, while inter-community edges are fewer but often link key hubs. Tracking these patterns supports anomaly detection, network health assessment, and potentially identifying emerging influential developers.

VII. Limitations and Future Work

Despite the promising results, our framework has several limitations that merit further investigation:

A. Computational Complexity

The hybrid approach, which combines Louvain, Spectral Clustering, and GNN embeddings with ensemble voting, increases computational requirements compared to single-algorithm methods. While it yields high modularity and cohesive communities, running multiple algorithms sequentially is costly, suggesting the need for approximation techniques, parallel processing, or selective subgraph sampling for larger networks.

B. Dynamic Networks

Our current analysis is limited to static snapshots. Extending community detection to dynamic networks while preserving temporal consistency and tracking community evolution is an important direction for future research, enabling more realistic modeling of developer interactions over time.

C. Overlapping Communities

Our framework assumes disjoint communities. However, real-world developer networks often include overlapping groups, where members contribute to multiple projects simultaneously. Modeling such overlapping communities will require more sophisticated approaches, potentially extending ensemble methods or using multi-membership GNN models.

VIII. Conclusion

In this paper, we investigated the structure of the GitHub developer collaboration network and applied multiple community detection algorithms to identify cohesive developer groups. By constructing a commit-based collaboration graph, we captured real-world collaborative relationships that go beyond social connections, enabling a more accurate analysis of technical collaboration patterns. Our comparative evaluation of algorithms revealed that modularity-based methods, particularly the Leiden algorithm, combined with hybrid approaches such as spectral refinement and GNN embeddings, provide the most stable and meaningful community structures. Centrality analysis further highlighted the presence of influential developers who act as bridges between clusters, shaping the overall information flow and collaboration patterns within the network.

Through our findings, we demonstrated that developer communities on GitHub exhibit a modular, hub-oriented structure, with a few dominant groups accounting for the majority of collaborations. These insights have practical applications in collaborator recommendation, project management, influence analysis, and platform-level intervention design. At the same time, we acknowledged the limitations related to dataset coverage, computational scalability, and the exclusive focus on commit-based interactions. Future work aims to extend the analysis to broader datasets, incorporate multiple interaction types, and develop adaptive algorithms to further enhance community detection performance.

In conclusion, this research provides a comprehensive framework for understanding collaborative structures in large-scale software development ecosystems. Our methodology and findings contribute to both the theoretical understanding of developer communities and practical strategies for optimizing open-source collaboration on platforms like GitHub.

References

[1] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb, "Social coding in github: transparency and collaboration in an open software repository," in *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*, ser. CSCW '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 1277–1286. [Online]. Available: https://doi.org/10.1145/2145204.2145396

[2] Y. Hu, J. Zhang, X. Bai, S. Yu, and Z. Yang, "Influence analysis of github repositories," vol. 5, no. 1, p. 1268. [Online]. Available: https://doi.org/10.1186/s40064-016-2897-7

[3] B. Moradi-Jamei, B. L. Kramer, J. B. S. Calderón, and G. Korkmaz, "Community formation and detection on github collaboration networks," in *Proceedings of the 2021 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, ser. ASONAM '21. New York, NY, USA: Association for Computing Machinery, 2022, p. 244–251. [Online]. Available: https://doi.org/10.1145/3487351.3488278

[4] W. Leibzon, "Social network of software development at github," in *2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, 2016, pp. 1374–1376.

[5] A. A. Wani, "Comprehensive analysis of clustering algorithms: exploring limitations and innovative solutions," vol. 10, p. e2286, publisher: PeerJ Inc. [Online]. Available: https://peerj.com/articles/cs-2286

[6] F. Daneshfar, M. Dolati, and S. Sulaimany, "Graph clustering techniques for community detection in social networks," in *Community Structure Analysis from Social Networks*. Chapman and Hall/CRC, num Pages: 20.

[7] "(PDF) social coding in GitHub: Transparency and collaboration in an open software repository," in *ResearchGate*. [Online]. Available: https://www.researchgate.net/publication/220879502_Social_Coding_in_GitHub_Transparency_and_Collaboration_in_an_Open_Software_Repository

[8] C. Yang, X.-h. Zhang, L.-b. Zeng, Q. Fan, T. Wang, Y. Yu, G. Yin, and H.-m. Wang, "RevRec: A two-layer reviewer recommendation algorithm in pull-based development model," vol. 25, no. 5, pp. 1129–1143. [Online]. Available: https://doi.org/10.1007/s11771-018-3812-x

[9] "(PDF) coding together at scale: GitHub as a collaborative social network." [Online]. Available: https://www.researchgate.net/publication/263811778_Coding_Together_at_Scale_GitHub_as_a_Collaborative_Social_Network

[10] "(PDF) measuring the local GitHub developer community." [Online]. Available: https://www.researchgate.net/publication/271532571_Measuring_the_local_GitHub_developer_community

[11] B. Vasilescu, A. Serebrenik, and V. Filkov, "A data set for social diversity studies of github teams," in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, 2015, pp. 514–517.

[12] Rate limits for the REST API. [Online]. Available: https://docs-internal.github.com/_next/data/tFjFO7IBmJGgIkLJAzxLy/en/free-pro-team%40latest/rest/using-the-rest-api/rate-limits-for-the-rest-api.json?apiVersion=2022-11-28&versionId=free-pro-team%40latest&restPage=rate-limits-for-the-rest-api

[13] S. J. Kwon, E. Park, and K. J. Kim, "What drives successful social networking services? a comparative analysis of user acceptance of facebook and twitter," vol. 51, no. 4, pp. 534–544. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S036233191400041X