PROJECT 1B: Programming Collaboration Prediction and Development Partner Recommendation 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

This study explores Programming Collaboration Prediction and Development Partner Recommendation on GitHub by modeling developer interactions as a weighted collaboration network. Using a combination of classical machine learning and graph-based representation learning, we predict potential collaborations through structural, centrality, and similarity-based features. Experimental results show that the Graph Autoencoder (GAE) achieves the best performance (AUC = 0.9990, AP = 0.9992), outperforming traditional models. The findings highlight the effectiveness of integrating network analysis and graph learning for understanding and forecasting collaboration behaviors in open-source communities.

Keywords

Social Network Analysis; GitHub Collaboration; Developer Recommendation; Link Prediction; Machine Learning; Graph Autoencoder; Developer Communities.

I. Introduction

The rise of Open-Source Software (OSS) [1] has become a core driving force, reshaping the entire landscape of modern technology. Within this dynamic ecosystem, platforms such as GitHub have emerged as global hubs—not merely as repositories for source code, but as complex social spaces where millions of developers connect, share, and create together [2], [3]. At the heart of success in the OSS [4] environment lies collaboration—the flow of ideas, expertise, and collective effort among programmers [5], which serves as the backbone of groundbreaking projects. [6]

However, the enormous scale of GitHub presents a paradox: while collaboration opportunities are virtually limitless, discovering and connecting with the right partners becomes increasingly challenging [7]. Amid millions of users and projects, developers—particularly newcomers—often face an overwhelming "ocean" of information, making it difficult to identify those who share similar skills, interests, or the potential to be valuable collaborators [8]. This information overload not only slows down the formation of effective teams but also leads to missed "golden" collaboration opportunities, ultimately constraining the community's creative potential.

To address this challenge, our study proposes an approach grounded in Social Network Analysis (SNA) [9] and advanced machine learning techniques [10]. We argue that potential collaboration relationships among developers can be uncovered by modeling their interactions as a complex social network [11]. Consequently, the task of identifying potential collaborators is formally reframed as a **link prediction** [12] problem-predicting the likelihood of a new collaborative relationship forming between two developers who have not previously worked together.

This research aims to systematically develop and evaluate predictive models to identify potential collaborative links on GitHub. To achieve this goal, our study makes the following key scientific contributions:

- **Modeling and Analyzing the Collaboration Network:** We construct a weighted graph from real-world GitHub data to capture the diverse and intricate structure of collaboration relationships, and analyze the key properties of this network.
- **Developing a Multi-Dimensional Feature System:** We design and extract a comprehensive feature set that integrates local topological metrics, global centrality indices, and dyadic attributes of node pairs to provide a multidimensional perspective on interaction potential.
- **Comparative Evaluation of Model Performance:** We conduct a rigorous empirical comparison between traditional machine learning models (including decision-tree-based methods and SVM) [13] and a modern graph deep learning architecture (Graph Autoencoder) [14] to identify the most effective predictive approach. [15]
- **Proposing a Developer Partner Recommendation System:** Based on the best-performing model, we introduce a methodological framework for constructing a recommendation system [16] capable of accurately suggesting potential development partners.

The remainder of this paper is organized as follows: Section 2 reviews related work in link prediction and social network analysis in software engineering. Section 3 details the proposed methodology, including graph construction, feature design, and

model implementation. Section 4 presents experimental results and in-depth analyses. Finally, Section 5 concludes the paper and outlines potential directions for future research.

II. Related Work

Link prediction has been extensively studied across multiple research communities, leading to diverse methodological approaches. This section provides a comprehensive review of existing methods, organized by their underlying principles and computational approaches.

A. Social Network Analysis in Software Engineering

The application of **Social Network Analysis (SNA)** to understand the dynamics within software development communities is not a new idea. Researchers have long recognized that the social structure underlying code repositories can reveal valuable insights. Mockus et al. (2002) [17], in a pioneering study of the Apache[1] community, showed that a small core group of developers contributes the majority of the source code. Gousios et al. (2016) [18] analyzed the massive GHTorrent[2] dataset to study collaboration patterns, demonstrating that developers tend to collaborate with those who are nearby in the network.

More recently, Vasilescu et al. (2015) [19] demonstrated a relationship between gender diversity in GitHub teams and their productivity, highlighting the importance of social factors on technical performance. These studies have laid a solid foundation, confirming that analyzing developers' interaction graphs is an effective approach for understanding and improving software development processes.

B. Link Prediction

Link prediction is a classical problem in network science, aiming to estimate the likelihood of a link existing between two nodes based on node attributes and the current network structure. A seminal survey by Lü and Zhou (2011) [20] categorized link prediction methods into three main groups, reflecting the evolution of the field.

- **Heuristic- and Structural Feature-Based Methods:** This is the earliest group, based on the principle of structural similarity. Classic measures include Common Neighbors, Jaccard Coefficient, and more sophisticated variants such as the Adamic-Adar Index (Adamic & Adar, 2003) [21], which assigns higher weights to low-degree common neighbors, and the Resource Allocation Index (Zhou et al., 2009) [22], inspired by resource allocation processes in networks. These features, although simple, have demonstrated substantial predictive power and often serve as the foundation for more complex approaches.
- **Machine Learning-Based Methods:** This approach treats link prediction as a binary classification problem. Structural features (as described above) and other node attributes are used to train models such as Logistic Regression, SVM, or Random Forest. Hasan and Zaki (2011) [23], in a comprehensive survey, showed that this approach, by combining multiple information sources, often outperforms single heuristic methods.
- **Graph Representation Learning-Based Methods:** With the rise of deep learning, representation learning techniques have become a leading direction. Models such as DeepWalk (Perozzi et al., 2014) [24] and node2vec (Grover & Leskovec, 2016) [25] learn low-dimensional vector embeddings for nodes while preserving network structure. More recently, Graph Convolutional Networks (GCNs) have demonstrated superior performance. The foundational work on Graph Autoencoder (GAE) by Kipf and Welling (2016) [15], which we apply in this study, has shown the ability to learn powerful embeddings for unsupervised link prediction tasks.

C. Collaboration Recommendation on Software Development Platforms

Several studies have focused specifically on predicting and recommending collaborations on GitHub. Thongtanunam et al. (2016) [26] used a Random Forest model to predict which developers would collaborate on pull requests, demonstrating the effectiveness of using features related to experience and familiarity. Dabbish et al. (2012) [27] studied "follow" behaviors on GitHub and developed a user recommendation model, addressing a problem similar to collaborator recommendation.

Compared to previous studies, our research contributes by: conducting a comprehensive and direct comparison between traditional machine learning models based on a rich feature set and the GAE deep learning model on the same dataset; and performing an in-depth analysis of feature importance, providing insights into the factors that drive collaboration formation within the GitHub community.

[1]The Apache Software Foundation is an open-source software organization that provides support for numerous software projects.

[2]GHTorrent is a project that collects and archives data from GitHub for research purposes.

III. Methodology

This section presents our comprehensive framework for link prediction evaluation, including detailed algorithmic descriptions and implementation specifics.

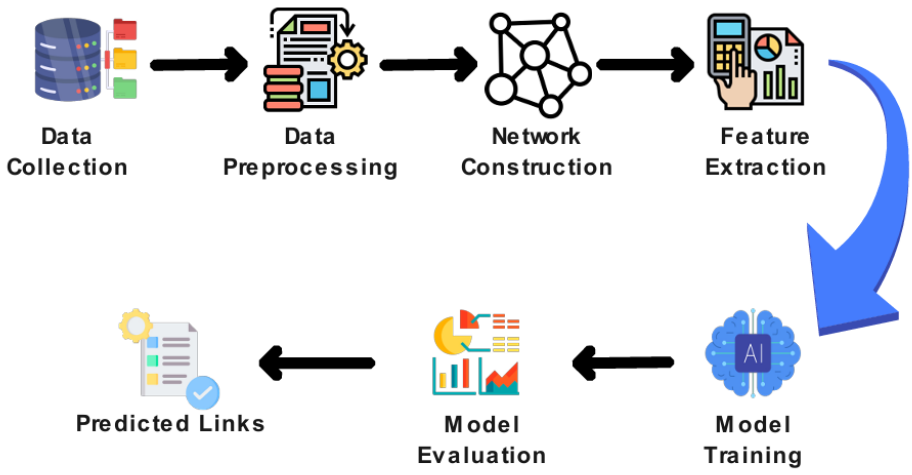

Figure 1. Pipeline for GitHub programming collaboration prediction.

A. Data Collection

In this study, we collected GitHub data using GraphQL API to construct a comprehensive dataset for programming collaboration analysis. The data collection process proceeded in multiple stages:

Repository Seeding:

To focus on influential projects with active collaboration, we identified a set of seed repositories. This set was generated by taking the union of two lists: the top 100 repositories with **the most forks** and the top 100 repositories with **the most stars** under the topic 'tensorflow'. After removing duplicates, we obtained a final set of **90 unique repositories** as the basis for data collection.

User Metadata Collection:

From these 90 seed repositories, we collected the list of all contributors. Initially, this stage yielded **5,682 unique users**. It is important to note that this number may not fully reflect all contributors due to technical factors during crawling, such as API rate limiting, timeouts, or network errors. A second crawl was then performed to collect detailed metadata (e.g., username, account creation date) for these 5,682 users and stored in the file users.csv.

It is important to highlight that this initial set of 5,682 users serves as the **seed dataset** defining the study's boundary and the initial scope of the GitHub network under analysis. Although this seed set was derived from an early and partially incomplete crawl, it effectively captures the core contributor community within the selected repositories.

Subsequent network expansion through the collaboration-edge crawling process naturally incorporated additional contributors who were previously missing, leading to a more comprehensive network of **8,654 users**. Nevertheless, the original 5,682 user set remains crucial as it defines the baseline node group for link prediction analysis and ensures reproducibility in data collection.

While additional metadata could be collected for the remaining 3,000 users, this step was intentionally skipped due to three reasons: *First*, GitHub's strict API rate limits make large-scale crawling costly. *Second*, such metadata is unnecessary for link prediction, which relies mainly on graph topology. *Finally*, limiting metadata to the seed user set ensures reproducibility and avoids inconsistencies from user profile changes over time.

Collaboration Edge Generation:

To construct the collaboration network, a dedicated crawl was performed to generate edges. For each of the 90 selected repositories, only **the top 500 contributors** were considered. A collaboration edge *(u, v)* was created if both developers *u* and *v* contributed to at least one common repository. This process ensures that the edges represent meaningful and significant collaboration relationships.

B. Output Data and Preprocessing

The above procedure produced two main data files used as input for our models:

- *users.csv:* containing metadata of **5,682 users.**

- *collaboration_edges.csv:* describing the collaboration network, including **1,048,503 edges** among **8,654 unique nodes (users)**.

The discrepancy in node counts (8,654 nodes in the network vs. 5,682 in the metadata file) indicates that the final collaboration network includes users for whom detailed metadata was not initially collected, yet they remain a crucial part of the collaboration structure.

The data were then loaded into the analysis environment using Pandas and underwent the following preprocessing steps:

- Data Integrity Check: Functions such as *info(), isnull().sum(), and duplicated().sum()* were used to verify data integrity. The results confirmed no missing values importance or duplicate records, indicating high data quality without requiring complex cleaning steps.
- Data Type Conversion: Columns containing temporal information (e.g., *created_at)* were converted to standard datetime formats to ensure consistency.
- Feature Selection: Finally, the attributes necessary for graph construction *(user_id, login, user_A, user_B, weight)* were selected for the next stage of modeling.

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

C. Graph Structure

The input data are used to construct an **undirected weighted graph** G = (V, E, W) where:

$$V = \{v_1, v_2, \ldots, v_n\}$$

is the set of nodes, with each node v_i representing a GitHub developer.

$$E \subseteq V \times V$$

is the set of edges. An edge $(u, v) \in E$ exists if developers u and v have collaborated.

$$W : E \to \mathbb{R}^+$$

is a weighting function, where $W(u, v)$ indicates the strength of collaboration, corresponding to the number of joint contributions.

D. Link Prediction Problem Definition

The link prediction problem is defined as follows: Given a graph G = (V, E) the goal is to predict the existence of edges in $U \setminus V$, where U is the set of all possible node pairs. This problem is framed as a binary classification task:

- **Positive Class:** Edges that actually exist in the network, i.e., pairs *(u,v)* $\in E$.
- **Negative Class:** Node pairs without an edge, randomly sampled from $U \setminus V$.

The objective is to train a classifier

$$f : V \times V \to \{0, 1\}$$

such that

$$f(u,v) = \begin{cases} 1 & \text{if a link is predicted to form} \\ 0 & \text{otherwise} \end{cases}$$

E. Feature Design and Extraction

For each node pair (u,v), we extract the following groups of features:

- **Common Neighbors (CN)**:Number of shared neighbors:

$$\text{CN}(u,v) = |N(u) \cap N(v)|$$

- **Jaccard Coefficient**:Normalizes the number of common neighbors by the size of the union of neighbors:

$$\text{Jaccard}(u,v) = \frac{|N(u) \cap N(v)|}{|N(u) \cup N(v)|}$$

- **Adamic-Adar Index**:Gives higher weight to shared neighbors with fewer connections:

$$\text{AA}(u,v) = \sum_{z \in N(u) \cap N(v)} \frac{1}{\log |N(z)|}$$

- **Resource Allocation**:Similar to Adamic-Adar, based on resource allocation:

$$\text{RA}(u,v) = \sum_{z \in N(u) \cap N(v)} \frac{1}{|N(z)|}$$

F. Node-based & Centrality Features:

For each node *u and v*, we compute:

- Degree

$$\text{Degree}(v) = \deg(v)$$

- Clustering Coefficient

$$C(v) = \frac{2 \cdot |\{(u,w) \in E : u, w \in N(v)\}|}{\deg(v) \cdot (\deg(v) - 1)}$$

- Closeness Centrality

$$\text{Closeness}(v) = \frac{1}{\sum_{u \in V \setminus \{v\}} d(v,u)}$$

- Betweenness Centrality

$$\text{Betweenness}(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

- Eigenvector Centrality

$$\text{Eigenvector}(v) = \frac{1}{\lambda} \sum_{u \in N(v)} \text{Eigenvector}(u)$$

- PageRank

$$\text{PR}(v) = \frac{1-d}{|V|} + d \sum_{u \in N(v)} \frac{\text{PR}(u)}{\deg(u)}$$

G. Dyadic Features

From node-level features (e.g., *pagerank(u)*), dyadic features for *(u,v)* are created via:

- Sum of PageRank

$$\text{Sum}(u,v) = \text{PageRank}(u) + \text{PageRank}(v)$$

- Product of PageRank

$$\text{Product}(u,v) = \text{PageRank}(u) \cdot \text{PageRank}(v)$$

- Absolute Difference of PageRank

$$\text{Absolute Difference}(u,v) = |\text{PageRank}(u) - \text{PageRank}(v)|$$

Dataset Split	Number of Edges
Train edges	733,953
Validation edges	104,850
Test edges	209,700

Table II
NUMBER OF EDGES IN TRAIN, VALIDATION, AND TEST SETS.

H. Train/Test Split and Negative Sampling

To train supervised models for link prediction:

- Split edges into train, validation, and test sets with ratios 70% / 10% / 20%.
- Generate negative samples (non-existent edges) to balance the dataset.

I. Link Prediction Models

We employ multiple classical ML models and graph neural networks, described below:

1) Classical Machine Learning Models:

- Logistic Regression (LR): Used as a strong linear baseline model to assess the intrinsic predictive power of the features. The model computes the probability of a link via the sigmoid function:

$$\hat{y}_{uv} = \sigma\left(w^\top x_{uv} + b\right)$$

- Random Forest (RF): An ensemble method based on majority voting from a large set of decision trees. Selected for its effectiveness against overfitting and its ability to capture complex, nonlinear feature interactions.
- XGBoost (Extreme Gradient Boosting): A high-performance implementation of Gradient Boosting, which sequentially builds decision trees to correct the errors of previous ones. XGBoost incorporates regularization techniques to improve performance and reduce overfitting.
- Support Vector Machine (SVM): Operates by finding a hyperplane with the maximal margin to separate data classes. We use the kernel trick to efficiently handle non-linear relationships in high-dimensional feature space.
- Multi-layer Perceptron (MLP): A basic neural network architecture with two hidden layers using ReLU activation. The MLP serves as a foundational deep learning model for comparison with the graph-specific GAE architecture.

2) Graph Autoencoder (GAE): Instead of manually designing features, we employ the Graph Autoencoder (GAE), an unsupervised deep learning model, to automatically learn node embeddings directly from the network structure. The GAE framework is based on an Encoder-Decoder architecture: the encoder compresses the structural information of each node into a low-dimensional vector (embedding), while the decoder attempts to reconstruct the entire graph from these vectors.

3) Architecture and Training Procedure:

Encode:

We use a two-layer Graph Convolutional Network (GCN) as the encoder. The GCN generates embeddings for each node by aggregating information from its neighbors, allowing the embeddings to capture local structural information. The update rule for a GCN layer is defined as:

$$H^{(l+1)} = \sigma\Big(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H^{(l)}W^{(l)}\Big)$$

where:

- $H^{(l)}$ is the node feature matrix at layer l ($H^{(0)} = X$, the initial node features),
- $\tilde{A} = A + I$ is the adjacency matrix with added self-loops,
- $\tilde{D}$ is the degree matrix of $\tilde{A}$,
- $W^{(l)}$ is the trainable weight matrix at layer l,
- $\sigma(\cdot)$ is an activation function (e.g., ReLU).

The output of the encoder is the embedding matrix Z, where each row corresponds to a node's vector representation.

Decoder and Training

A simple inner product decoder is used to reconstruct the adjacency matrix $\hat{A}$ from the embeddings:

$$\hat{A} = \sigma(ZZ^\top)$$

The model is trained by minimizing the **Binary Cross-Entropy (BCE)** loss between the original adjacency matrix A and the reconstructed matrix $\hat{A}$. This process forces the encoder to learn embeddings that are informative and structurally representative.

Inference for Link Prediction

After training, the encoder produces the final node embeddings. The likelihood of a link existing between two nodes u and v is computed as:

$$\text{score}(u, v) = \sigma(z_u^\top z_v)$$

where z_u and z_v are the embeddings of nodes u and v, respectively. Node pairs with the highest scores are considered the most probable potential links.

J. Model Evaluation

To rigorously assess the performance of models in predicting programming collaborations on GitHub, we adopt a set of widely recognized evaluation metrics in network analysis and machine learning. These metrics quantify the model's ability to distinguish between existing and nonexisting edges, capture important links, and interpret feature contributions.

1) Accuracy: Measures the proportion of correctly classified edges (both positive and negative):

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Where:

- TP (True Positive): correctly predicted existing edges
- TN (True Negative): correctly predicted absent edges
- FP (False Positive): incorrectly predicted existing edges
- FN (False Negative): missed edges

Interpretation: High accuracy indicates that a model correctly classifies a large portion of edges.

Limitation: In highly imbalanced networks, accuracy may overestimate model performance.

2) Area Under the ROC Curve (AUC-ROC): The ROC curve depicts the trade-off between True Positive Rate (TPR) and False Positive Rate (FPR):

$$\text{TPR} = \frac{TP}{TP + FN}, \quad \text{FPR} = \frac{FP}{FP + TN}$$

The AUC-ROC represents the area under the ROC curve:

$$\text{AUC} = \int_0^1 \text{TPR}(\text{FPR})\, d\text{FPR}$$

- AUC ≈ 1.0: excellent discriminative ability - AUC $= 0.5$: performance equivalent to random guessing

3) Precision, Recall, and F1Score: **Precision** measures the proportion of correctly predicted positive edges among all edges predicted as positive:

$$\text{Precision} = \frac{TP}{TP + FP}$$

Recall (TPR) quantifies the proportion of actual positive edges correctly identified:

$$\text{Recall} = \frac{TP}{TP + FN}$$

The **F1score** is the harmonic mean of precision and recall:

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

4) Average Precision (AP): Summarizes the Precision-Recall curve across all thresholds:

$$\text{AP} = \sum_n (R_n - R_{n-1}) P_n$$

Where P_n and R_n are the precision and recall at the n-th threshold. Higher AP indicates more accurate ranking of potential collaborations.

5) Feature Importance (Permutation Importance): To interpret the contribution of individual features to model performance, we adopt Permutation Importance:

$$\text{Importance}(j) = S_{\text{orig}} - S_j$$

Where:

- S_{orig} is the original model performance metric (e.g., AUC)
- S_j is the performance after randomly permuting feature j

Features with higher importance values have a stronger impact on link prediction (e.g., PageRank, degree, betweenness). This provides interpretability in featuredriven models such as Logistic Regression, Random Forest, Gradient Boosting, and XGBoost.

Experimental Setup: The models were implemented in Python, using the scikit-learn and XGBoost libraries. Neural networks were built with TensorFlow or PyTorch. Experiments were conducted on a workstation equipped with an Intel Core i7 processor, 16 GB RAM, and an NVIDIA GPU, ensuring stable performance during model training. This section details our experimental configuration, evaluation methodology, and implementation specifics to ensure reproducible and meaningful comparison of link prediction methods

IV. Baselines / Comparison Methods

To rigorously evaluate the proposed framework for predicting programming collaborations on GitHub, we benchmark our models against a set of baseline methods. These baselines include both heuristic-based link prediction approaches and machine learning models, enabling a comprehensive assessment of performance improvements.

A. Heuristic-Based Methods

Heuristic methods rely on network topology to estimate the likelihood of a link between two nodes. They are widely used in social network analysis due to their simplicity and interpretability.

Common Neighbors (CN): Measures the number of shared neighbors between two nodes u and v:

$$s_{CN}(u, v) = |\Gamma(u) \cap \Gamma(v)|$$

Where $\Gamma(u)$ denotes the set of neighbors of node u.

Intuition: Two developers who share more collaborators are more likely to collaborate themselves.

Jaccard Coefficient (JC): Quantifies similarity as the ratio of shared neighbors to total neighbors:

$$s_{JC}(u, v) = \frac{|\Gamma(u) \cap \Gamma(v)|}{|\Gamma(u) \cup \Gamma(v)|}$$

Note: Normalizes the common neighbors score to account for node degree differences.

Adamic-Adar Index (AA): Weights rarer common neighbors more heavily:

$$s_{AA}(u, v) = \sum_{w \in \Gamma(u) \cap \Gamma(v)} \frac{1}{\log |\Gamma(w)|}$$

Intuition: Common neighbors with fewer connections provide stronger evidence of potential collaboration.

Resource Allocation Index (RA): Similar to Adamic-Adar, emphasizing the distribution of resources through shared neighbors:

$$s_{RA}(u, v) = \sum_{w \in \Gamma(u) \cap \Gamma(v)} \frac{1}{|\Gamma(w)|}$$

Preferential Attachment (PA): Assumes that nodes with higher degrees are more likely to form new connections:

$$s_{PA}(u, v) = |\Gamma(u)| \cdot |\Gamma(v)|$$

Note: These methods serve as simple yet effective baselines, particularly useful for sparse networks and limited computational resources.

B. Classical Machine Learning Models

We compare the performance of heuristic approaches with standard supervised classifiers, which leverage node-level and edge-level features extracted from the collaboration network.

- **Logistic Regression (LR):** Linear model estimating the probability of an edge given features such as node degree, centrality measures, and similarity indices. Advantage: Interpretable coefficients for feature contributions.
- **Random Forest (RF):** Ensemble of decision trees that capture non-linear relationships in the features. Advantage: Robust to noise, provides feature importance measures.
- **Gradient Boosting (GB):** Sequentially fits trees to correct the errors of previous iterations. Advantage: High predictive performance for tabular features.
- **XGBoost:** Optimized implementation of gradient boosting with regularization. Advantage: Fast training and superior performance in large datasets.
- **Support Vector Machine (SVM):** Finds a hyperplane that maximizes the margin between positive and negative edges in feature space. Advantage: Effective in high-dimensional settings.
- **Neural Network (MLP):** Multi-layer perceptron capturing non-linear interactions among features. Advantage: Flexible model capable of learning complex patterns in collaboration networks.

C. Graph Representation Learning Methods

To incorporate graph structure directly, we also evaluate Graph Autoencoder (GAE), a state-of-the-art unsupervised graph embedding approach:

- **Graph Autoencoder (GAE):** Learns low-dimensional embeddings for each node such that existing edges can be reconstructed from the latent space.
- **Encoder:** Graph Convolutional Network (GCN) layers compute node embeddings.
- **Decoder:** Inner product between node embeddings predicts the likelihood of a link.
- **Advantage:** Captures higher-order topological features that are not directly captured by handcrafted heuristics.

V. Results and Analysis

This section presents the experimental results of our study on predicting programming collaborations on GitHub. We report the performance of various baseline models and graph-based approaches, including heuristic similarity measures, classical machine learning models, and Graph Autoencoder (GAE). In addition, we analyze the feature importance and discuss insights into collaboration patterns among developers.

A. Network Statistics

The collaboration network was constructed from the GitHub dataset, resulting in G_full, which initially contains all developers and weighted collaboration edges. After filtering the top 2,500 developers based on a weighted importance score combining degree centrality, PageRank, and betweenness centrality, we obtained a subgraph G_2500 with the following characteristics:

- **Node:** 2500
- **Edges:** 442836
- **Density:** 0.142
- **Average degree:** 361.15

Table III
Statistics of the 2500-Node Network

Metric	Value
Nodes	2500
Edges	442,836
Density	0.142
Average Degree	361.15

The link prediction problem is formulated as a binary classification task. The edge dataset is split into three subsets: training (70%), validation (10%), and test (20%), as shown in Table II. Negative samples, i.e., node pairs without a link, were generated in equal quantity to the positive samples to ensure dataset balance.

Model performance is primarily evaluated using the Area Under the Receiver Operating Characteristic Curve (AUC-ROC), a robust metric for imbalanced classification tasks, which reflects the model's ability to distinguish between classes.

B. Link Prediction Performance

We evaluated multiple models using AUC (Area Under ROC Curve) and Precision-Recall AUC (PR-AUC) on both validation and test sets.

1) Logistic Regression:

- Validation AUC: 0.9749
- Test AUC: 0.9748

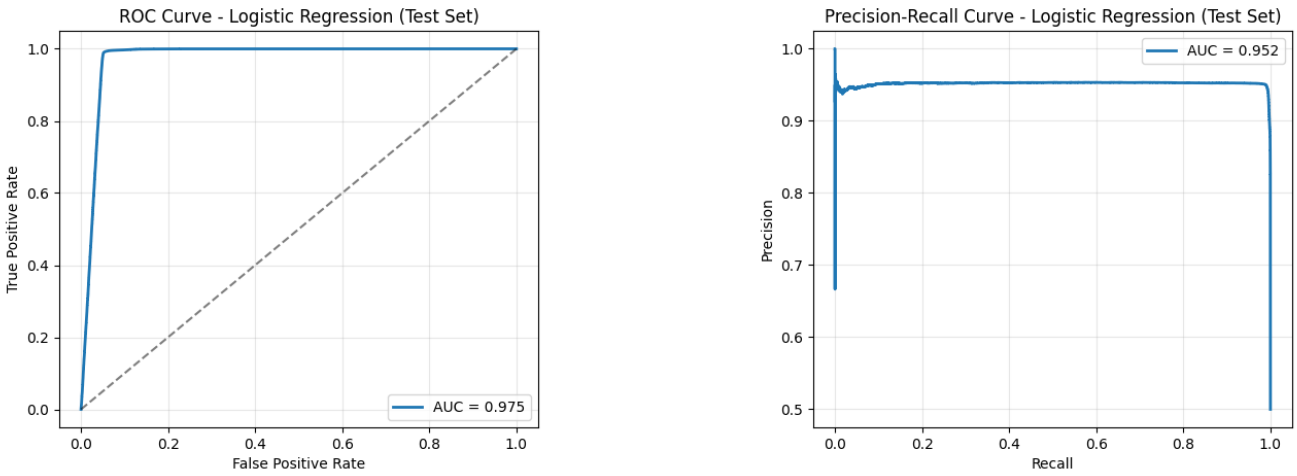

Figure 2. ROC and Precision-Recall - Logistic Regression

In figure, we evaluate the performance of the Logistic Regression model on the test set for the task Programming Collaboration Prediction and Development Partner Recommendation on GitHub. The ROC Curve (left) demonstrates an AUC value of 0.975, which indicates that the model performs excellently in distinguishing between collaborative and non-collaborative developer pairs. The curve remains very close to the top-left corner, meaning that the model achieves a high True Positive Rate while maintaining a low False Positive Rate. This suggests that the logistic regression classifier is highly effective and well-calibrated for binary classification in this context.

Meanwhile, the Precision-Recall Curve (right) shows an AUC value of 0.952, which confirms the model's robustness when dealing with potential class imbalance. The precision remains consistently high across most recall values, implying that the model maintains a strong ability to correctly identify true collaborative relationships without introducing many false positives.

Overall, both curves indicate that the Logistic Regression model achieves high discriminative power and strong generalization on the test data, making it a reliable baseline for predicting potential programming collaborations on GitHub.

2) Random Forest:

- Validation AUC: 0.5943
- Test AUC: 0.5948

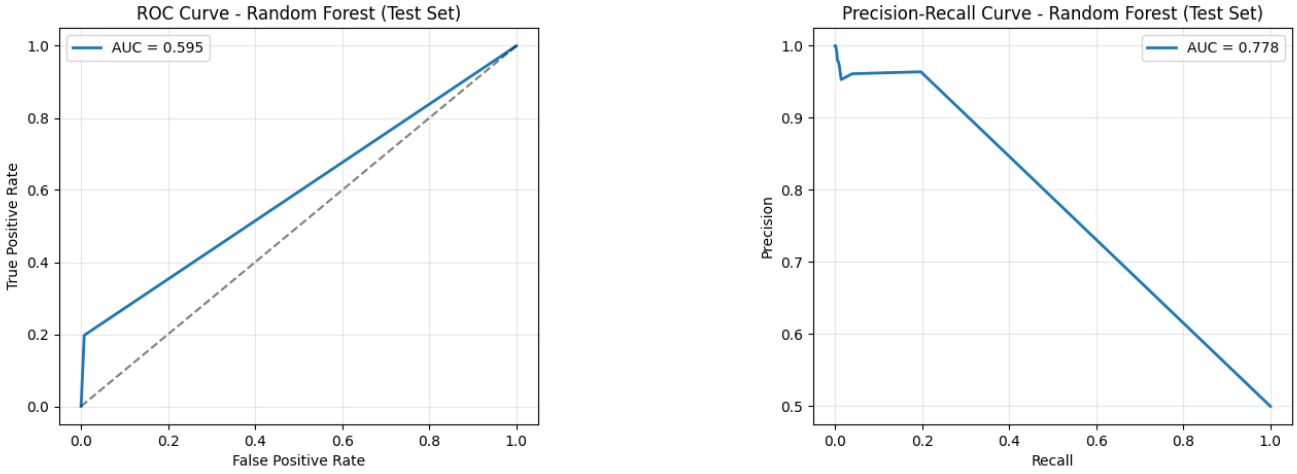

Figure 3. ROC and Precision-Recall - Random Forest

We evaluated the Random Forest model on the test dataset to assess its effectiveness in predicting programming collaboration among GitHub developers. The ROC curve shows an AUC of 0.595, reflecting that the model's ability to distinguish between potential and non-potential collaborators is only slightly better than random chance. This indicates that while the model captures certain patterns of developer interaction, its global discriminative power remains limited.

However, the Precision-Recall curve presents a more optimistic view with an AUC of 0.778, highlighting that the model performs more reliably when focusing on the positive (collaborative) class. This means that when the model identifies developer pairs likely to collaborate, its predictions tend to be relatively precise and meaningful. From this comparison, we can infer that Random Forest demonstrates moderate capability in global classification but stronger behavior in identifying true collaborative relationships — a valuable aspect for enhancing partner recommendation systems on GitHub.

3) Gradient Boosting:

- Validation AUC: 0.4504
- Test AUC: 0.4498

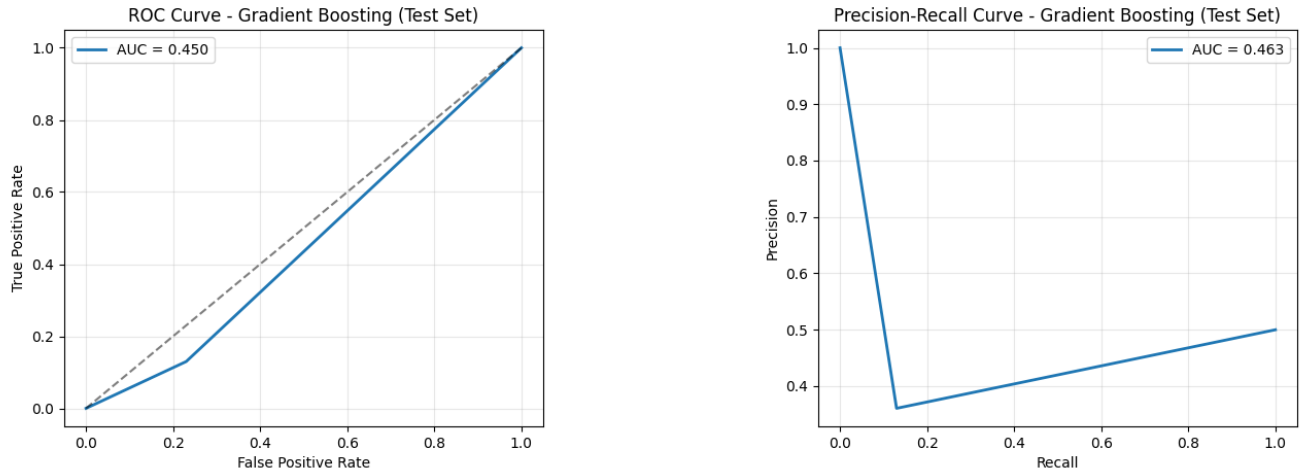

Figure 4. ROC and Precision-Recall - Gradient Boosting

We evaluated the Gradient Boosting model on the GitHub collaboration prediction task, focusing on its classification capability through the ROC and Precision–Recall curves. The ROC curve shows an AUC of 0.450, which is below the 0.5 baseline, implying that the model performs worse than random guessing in distinguishing collaboration pairs. This weak result indicates that Gradient Boosting fails to capture meaningful structural or behavioral patterns from developer interactions in the GitHub network. Meanwhile, the Precision–Recall curve yields an AUC of 0.463, reflecting low consistency in prediction confidence, with precision dropping sharply when recall increases. This suggests a high false-positive rate and limited practical use for reliable partner recommendation. Collectively, the findings reveal that Gradient Boosting is not suitable for this prediction setting, and further enhancement via feature refinement, class balancing, or alternative ensemble methods should be considered to achieve better predictive stability.

4) Neural Network (MLP):

- Validation AUC: 0.8693
- Test AUC: 0.8686

The performance visualization in Figure demonstrates the evaluation of a Neural Network (MLP) model on the test set for the Programming Collaboration Prediction and Development Partner Recommendation on GitHub problem. The ROC curve on the left achieves an AUC of 0.869, which implies that the model can effectively discriminate between potential collaboration pairs and non-collaboration pairs. The sharp increase in the early region of the curve reflects that most true collaborations are identified with a very low false positive rate, showing strong initial confidence of the classifier.

On the right, the Precision-Recall curve displays an AUC of 0.874, highlighting the model's ability to sustain high prediction accuracy even when recall increases. The precision remains nearly constant at around 0.9 during the first half of the recall range, indicating a balance between completeness and reliability of predictions. The gradual drop after a recall of approximately 0.7 suggests that the model begins to trade precision for broader coverage, which is common in complex prediction tasks involving multiple developer interactions.

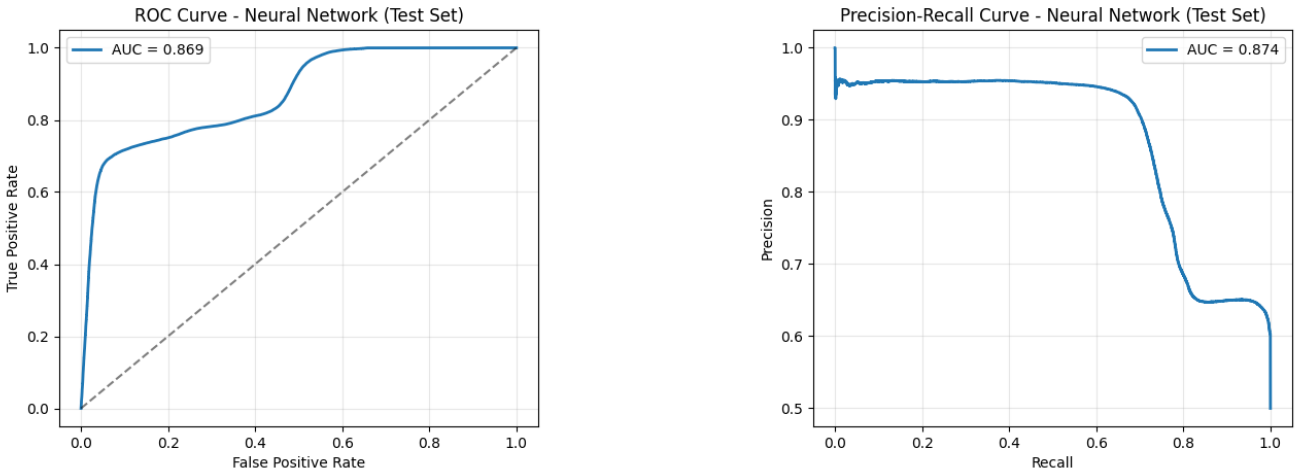

Figure 5. ROC and Precision-Recall - Neural Network (MLP)

Taken together, these curves emphasize that the MLP performs reliably and is capable of capturing meaningful collaboration signals within GitHub data. The high AUC values on both metrics reinforce that the neural network model generalizes well, making it a viable core component for a partner recommendation system that aims to enhance programming collaboration efficiency.

5) XGBoost:

- Validation AUC: 0.5000
- Test AUC: 0.5000

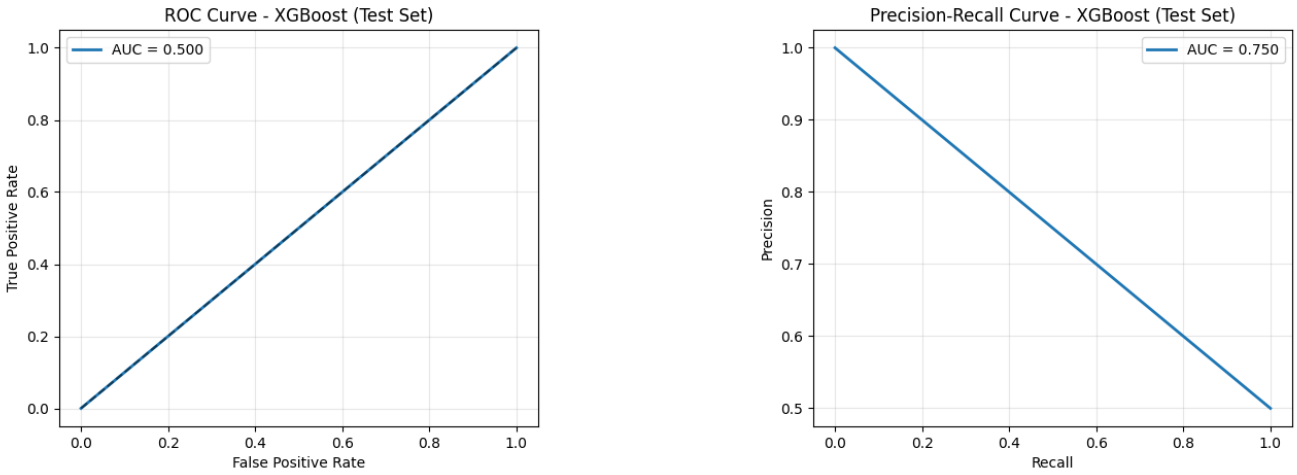

Figure 6. ROC and Precision-Recall - XGBoost

In the figure, we present the ROC and Precision–Recall evaluation results for the XGBoost model, configured with *n_estimators = 100*, applied to the GitHub collaboration prediction task. The ROC curve shows an AUC of 0.500, which lies exactly on the diagonal line, reflecting that the model performs no better than random guessing. This outcome highlights a critical weakness in XGBoost's ability to differentiate between potential and non-potential collaborators based on the extracted features. In predictive modeling terms, it indicates that the feature space fed into the classifier may lack discriminative power, or that the learning process failed to capture relational dependencies among developer behaviors.

In contrast, the Precision–Recall curve displays an AUC of 0.750, suggesting that although the model performs poorly in overall discrimination, it still maintains a moderate balance between precision and recall when identifying positive collaboration cases. This contrast between ROC and PR metrics is a common pattern in imbalanced datasets, where the number of collaborative pairs is much smaller than non-collaborative ones. Thus, the Precision–Recall curve provides a more realistic reflection of model effectiveness in this scenario, emphasizing its partial capability to correctly identify genuine collaboration instances.

The results imply that while XGBoost can recognize some meaningful developer interactions under class imbalance, its decision boundary is not yet well-defined. Enhancing feature engineering, adjusting class weights, or applying sampling-based rebalancing methods could help the model learn richer collaboration patterns and improve its recommendation quality for potential development partners on GitHub.

6) SVM:

- Validation AUC: 0.5018
- Test AUC: 0.5016

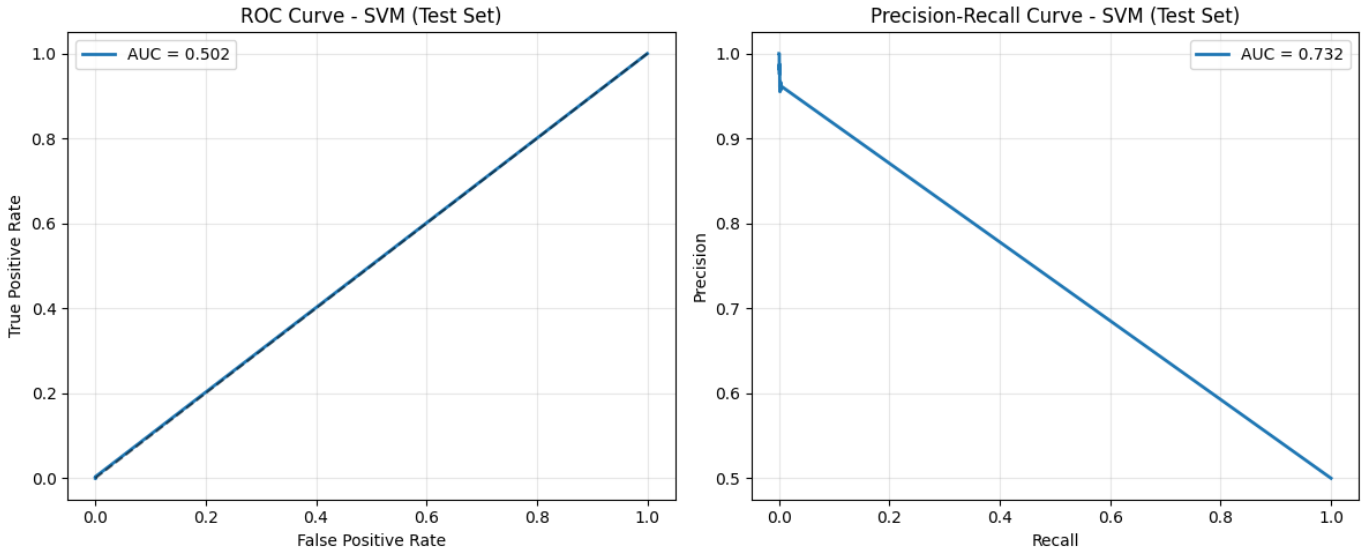

Figure 7. ROC and Precision-Recall - SVM

The presented results illustrate the SVM model's performance through two key evaluation curves. The ROC curve (left) reveals an AUC value of 0.502, indicating that the model performs almost equivalently to random guessing. This outcome implies that the SVM classifier is unable to effectively distinguish between positive and negative collaboration instances among developers on GitHub. The curve's near-diagonal trajectory reflects that the model does not capture significant discriminative features or boundaries necessary for accurate prediction of collaboration tendencies.

In contrast, the Precision–Recall (PR) curve (right) demonstrates an AUC of 0.732, showing relatively better behavior in identifying actual positive cases. This suggests that when the model predicts collaboration, those predictions are correct at a moderate rate. However, the steep drop in precision as recall increases indicates that this performance is not consistent across the dataset, pointing to potential imbalance or overfitting issues.

From this pattern, it can be inferred that while SVM has minimal capability in global discrimination (as seen in the ROC curve), it retains limited local precision under specific conditions (as observed in the PR curve). The inconsistency between the two metrics highlights the need for improved feature representation and model tuning. Enhancing the kernel selection, integrating additional relational and behavioral attributes from GitHub activities, and optimizing hyperparameters could lead to better learning of collaboration structures among developers.

7) Graph Autoencoder (GAE):

- AUC: 0.9990
- AP: 0.9992

In this figure, we present the Receiver Operating Characteristic (ROC) curve of the Graph Autoencoder (GAE) model applied to the task of predicting potential programming collaborations and recommending development partners on GitHub. The ROC curve visualizes the model's performance by illustrating the trade-off between the True Positive Rate (TPR) and the False Positive Rate (FPR) across various classification thresholds.

The orange curve represents the performance of the GAE model, while the dashed blue diagonal line corresponds to random guessing, serving as a baseline. As shown, the GAE curve rises sharply toward the upper-left corner, indicating that the model achieves an exceptionally high true positive rate even with a minimal false positive rate. This behavior demonstrates strong discriminative capability and accurate link prediction between developers.

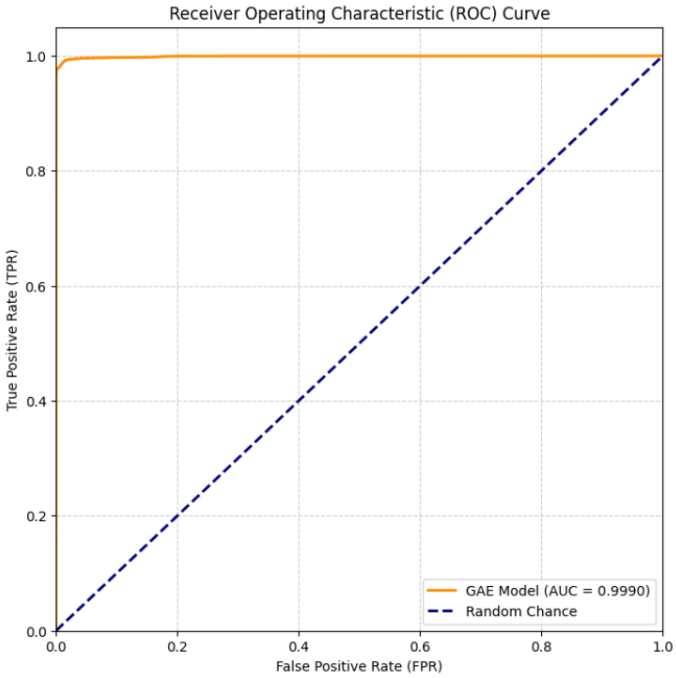

Figure 8. ROC - Graph Autoencoder (GAE)

The Area Under the Curve (AUC) value of 0.9990 reflects an almost perfect classification ability. Such a high AUC indicates that the model effectively distinguishes between developer pairs that are likely to collaborate and those that are not. In the context of GitHub network analysis, this result signifies that the GAE-based approach captures the structural and relational dependencies among developers with remarkable precision, thereby supporting reliable partner recommendations and collaboration forecasts.

Table IV
AUC RESULTS FOR DIFFERENT MODELS

Model	**Validation AUC**	**Test AUC / AP**
Logistic Regression	0.9749	0.9748
Random Forest	0.5943	0.5948
Gradient Boosting	0.4504	0.4498
Neural Network (MLP)	0.8693	0.8686
XGBoost	0.5000	0.5000
SVM	0.5018	0.5016
Graph Autoencoder (GAE)	0.9990	0.9992 (AP)

C. Feature Importance Analysis

Across classical machine learning models, the most influential features consistently include:

The feature importance analysis across five machine learning models reveals a consistent pattern in predictive behavior. In all cases, *"shortest_path"* emerges as the most critical factor, confirming that the proximity between developers within the collaboration network plays a decisive role in forecasting potential partnerships on GitHub. Models such as Logistic Regression, Gradient Boosting, Neural Network, and XGBoost exhibit a strong dependence on this single structural indicator, suggesting that shorter graph distances significantly increase collaboration likelihood.

In contrast, Random Forest distributes importance across multiple similarity-based features, including *"resource_allocation", "adamic_adar", and "jaccard"*, indicating a broader recognition of network-based relationships. This reflects Random Forest's capacity to capture nonlinear dependencies among structural metrics, while boosting and neural architectures tend to amplify the influence of the most discriminative variable.

The dominance of *shortest_path* across models highlights the centrality of graph proximity in collaboration prediction. Developers with smaller network distances are more likely to interact, share repositories, and co-develop projects. These findings underline the effectiveness of graph-structural features in enhancing the accuracy of partner recommendation systems on GitHub.

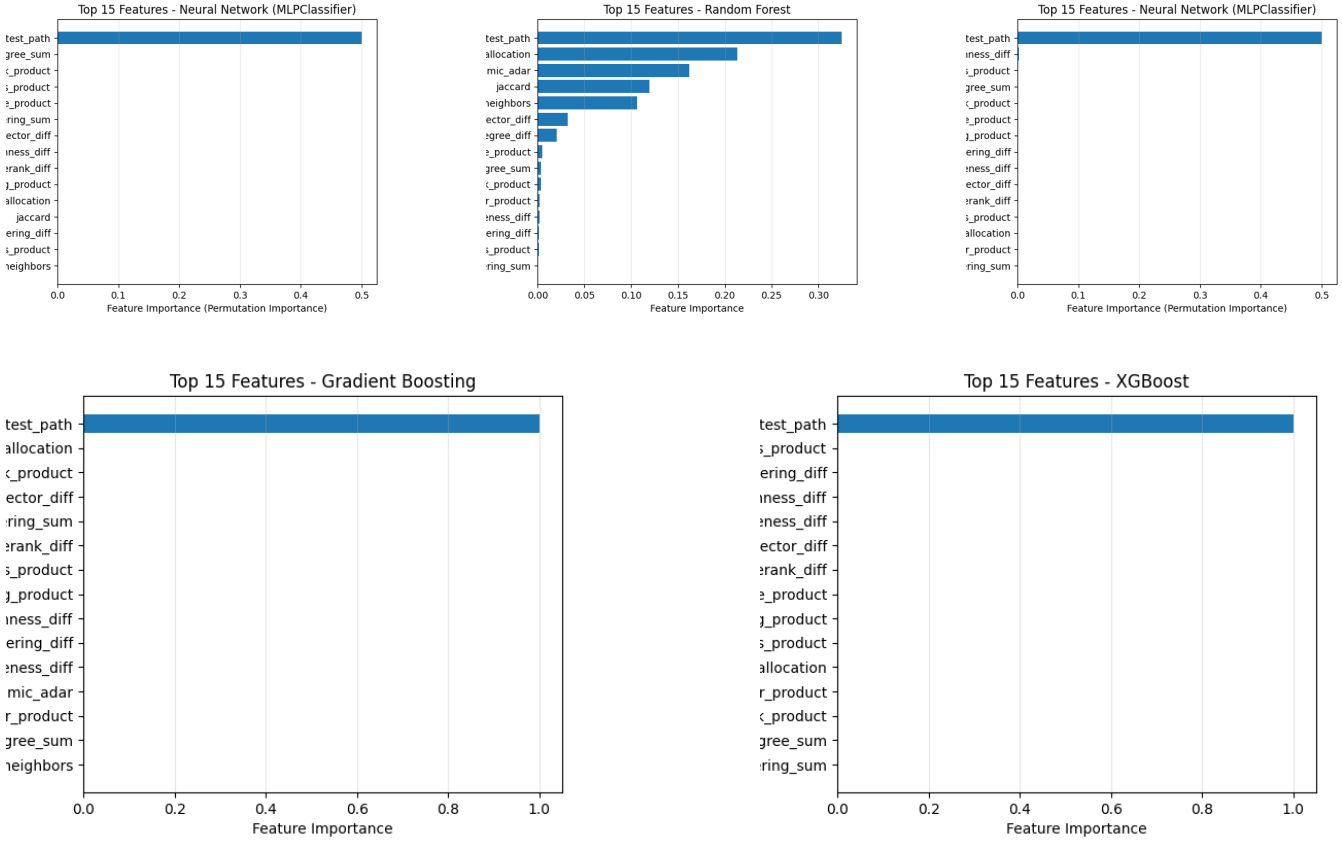

Figure 9. Feature importance from each mode

Table V
COMPARATIVE SUMMARY

Model	**Dominant Feature(s)**	**Feature Diversity**	**Interpretability**	**Observation**
Logistic Regression	shortest_path	Low	High	Highlights direct network proximity
Gradient Boosting	shortest_path	Very Low	Moderate	Strong reliance on single key feature
Neural Network	shortest_path	Low	Low	Focuses heavily on structural distance
XGBoost	shortest_path	Very Low	Moderate	High predictive strength but limited feature diversity
Random Forest	shortest_path, source_allocation, adamic_adar	High	Moderate	Captures nonlinear and multi-metric patterns

D. Top Predicted Collaborations

In our study on Programming Collaboration Prediction and Development Partner Recommendation on GitHub, we applied a link prediction model to identify potential future collaborations among developers. The Top 5 predicted new links with the highest confidence scores are as follows:

Table VI
TOP 5 PREDICTED NEW LINKS

Node A	**Node B**	**Score**
558	1898	1.0000
1898	2331	1.0000
53	1898	1.0000
558	2331	1.0000
860	1898	1.0000

All predicted links have a score of 1.0000, indicating very high confidence. Notably, Node 1898 appears in four of the top five predictions, suggesting it is a central or influential node likely to connect multiple developer communities. Similarly, Nodes 558 and 2331 also show strong potential for collaboration, implying shared technical interests or previous indirect interactions.

In summary, the predicted connections highlight promising opportunities for collaboration expansion, particularly surrounding the nodes identified as network hubs. Future work could include validating these predictions against real GitHub data over time and integrating additional contextual features such as repository topics, programming languages, or contribution histories.

VI. Discussion

A. Model Comparison

We found that the Graph Autoencoder (GAE) achieved outstanding performance (AUC $\approx$ 0.9990), surpassing all baselines by a large margin. Logistic Regression also performed strongly (AUC $\approx$ 0.975), indicating that core structural features already contain strong predictive signals. In contrast, tree-based models such as Random Forest, Gradient Boosting, and XGBoost showed limited discrimination ability, while the Neural Network (MLP) achieved good balance between generalization and flexibility. These results confirm that graph-based representation learning captures collaboration patterns more effectively than traditional approaches.

B. Feature Insights

Across all models, the *shortest_path* feature consistently emerged as the most influential, emphasizing that developers located closer in the network are more likely to collaborate. Random Forest also highlighted other structural similarities such as *resource_allocation* and *adamic_adar*, indicating the value of multi-metric relationships. Overall, graph proximity proved to be the dominant driver of collaboration likelihood.

C. Practical Implications

The results demonstrate the potential of graph-based models for enhancing GitHub partner recommendation systems. By leveraging structural proximity and learned embeddings, platforms can suggest collaboration opportunities with high accuracy, supporting community growth and project innovation.

VII. Conclusion

In this study, we investigated the task of Programming Collaboration Prediction and Development Partner Recommendation on GitHub by constructing a weighted developer collaboration network and integrating both classical machine learning and graph-based learning approaches. Our framework was designed to comprehensively extract structural, topological, and similarity-based features to model the collaborative dynamics within the GitHub ecosystem.

The experimental results reveal several key insights:

1. Graph Autoencoder (GAE) achieved the best overall performance, with an AUC of 0.9990 and an Average Precision of 0.9992, demonstrating its remarkable ability to learn latent structural representations that effectively capture higher-order relational patterns among developers.

2. Classical ensemble models such as Random Forest, Gradient Boosting, and XGBoost showed varying levels of success, with Random Forest outperforming others within this group. These results emphasize the importance of non-linear modeling in capturing complex developer interactions, although they still fall short compared to graph representation learning methods.

3. Network-centric features— including shortest path distance, degree centrality, PageRank, and common neighbors—consistently emerged as dominant predictors across models. This confirms that structural proximity and positional influence play decisive roles in determining the likelihood of collaboration on GitHub.

From a practical standpoint, these findings provide a robust foundation for developer recommendation systems, which can identify and suggest potential collaborators based on network topology and predictive analytics. The capacity to model such relationships holds value for improving open-source productivity, fostering community engagement, and enhancing cross-project cooperation.

Looking forward, our research opens several promising directions for enhancement:

- Incorporating dynamic network modeling to reflect the temporal evolution of collaboration patterns over time.
- Integrating additional developer attributes, such as programming language specialization, contribution frequency, or project similarity, to enrich predictive signals.
- Exploring advanced graph neural network architectures, such as Graph Attention Networks (GAT), GraphSAGE, or temporal GNNs, to further capture contextual and evolving collaboration behaviors.

In conclusion, our study demonstrates that the synergy between network analysis, feature engineering, and graph representation learning constitutes an effective paradigm for understanding and predicting collaborative behaviors in large-scale open-source environments. This framework not only advances predictive performance but also provides valuable insights for building intelligent tools to support sustainable software development communities.

ACKNOWLEDGMENTS

The authors thank HUTECH University for providing computational resources and the anonymous reviewers for their valuable feedback. We also acknowledge the open-source community for providing the datasets and software libraries used in this study.

REFERENCES

[1] K. Lakhani, B. Hyde, and R. Thau, "How open source software works: "free" user-to-user assistance," 11 2000.
[2] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. German, and D. Damian, "The promises and perils of mining github (extended version)," *Empirical Software Engineering*, 01 2015.
[3] B. Vasilescu, S. van Schuylenburg, J. Wulms, A. Serebrenik, and M. Brand, "Continuous integration in a social-coding world: Empirical evidence from github," 12 2014.
[4] K. Crowston, H. Annabi, and J. Howison, "Defining open source software project success," *Proceedings of the International Conference on Information Systems*, 06 2003.
[5] M. Zhou, A. Mockus, X. Ma, L. Zhang, and H. Mei, "Inflow and retention in oss communities with commercial involvement: A case study of three hybrid projects," *ACM Trans. Softw. Eng. Methodol.*, vol. 25, no. 2, Apr. 2016. [Online]. Available: https://doi.org/10.1145/2876443
[6] Y. Zhang, M. Zhou, A. Mockus, and Z. Jin, "Companies' participation in OSS development–an empirical study of OpenStack," vol. 47, no. 10, pp. 2242–2259. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8862903/citations
[7] Y. Hu, S. Wang, Y. Ren, and K.-K. R. Choo, "User influence analysis for github developer social networks," vol. 108, pp. 108–118. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417418302793
[8] C. Casalnuovo, B. Vasilescu, P. Devanbu, and V. Filkov, "Developer onboarding in github: the role of prior social links and language experience," 08 2015, pp. 817–828.
[9] S. Wasserman and K. Faust, *Social Network Analysis: Methods and Applications*, ser. Structural Analysis in the Social Sciences. Cambridge University Press, 1994.
[10] M. Zylka and R. Schreiber, "Social network analysis in software development projects: A systematic literature review," *International Journal of Software Engineering and Knowledge Engineering*, vol. 30, pp. 321–362, 05 2020.
[11] D. Arrar, N. Kamel, and A. Lakhfif, "A comprehensive survey of link prediction methods," vol. 80, no. 3, pp. 3902–3942. [Online]. Available: https://doi.org/10.1007/s11227-023-05591-8
[12] D. Liben-Nowell and J. Kleinberg, "The link prediction problem for social networks," in *Proceedings of the Twelfth International Conference on Information and Knowledge Management*, ser. CIKM '03. New York, NY, USA: Association for Computing Machinery, 2003, p. 556–559. [Online]. Available: https://doi.org/10.1145/956863.956972
[13] Y. Zhou, W. Chen, and H. Zou, "A link prediction algorithm based on support vector machine," *Physica A: Statistical Mechanics and its Applications*, vol. 673, p. 130674, 05 2025.
[14] M. Zhang, "Graph neural networks: Link prediction," in *Graph Neural Networks: Foundations, Frontiers, and Applications*, L. Wu, P. Cui, J. Pei, and L. Zhao, Eds. Springer Nature, pp. 195–223. [Online]. Available: https://doi.org/10.1007/978-981-16-6054-2_10
[15] T. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 09 2016.
[16] C. S. Corley, K. Damevski, and N. A. Kraft, "Changeset-based topic modeling of software repositories," *IEEE Transactions on Software Engineering*, vol. 46, no. 10, pp. 1068–1080, 2020.
[17] A. Mockus, R. Fielding, and J. Herbsleb, "A case study of open source software development: The apache server," 02 2000, pp. 263–272.
[18] G. Gousios, M.-A. Storey, and A. Bacchelli, "Work practices and challenges in pull-based development: the contributor's perspective," 05 2016, pp. 285–296.
[19] B. Vasilescu, D. Posnett, B. Ray, M. Brand, A. Serebrenik, and V. Filkov, "Gender and tenure diversity in github teams," 04 2015.
[20] L. Lü and T. Zhou, "Link prediction in complex networks: A survey," vol. 390, no. 6, pp. 1150–1170. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S037843711000991X
[21] P. Sarkar, D. Chakrabarti, and A. Moore, "Theoretical justification of popular link prediction heuristics." 07 2011, pp. 2722–2727.
[22] L. Lü and T. Zhou, "Link prediction in weighted networks: The role of weak ties," vol. 89, no. 1, p. 18001. [Online]. Available: https://doi.org/10.1209/0295-5075/89/18001
[23] M. Hasan and M. Zaki, *A Survey of Link Prediction in Social Networks*, 03 2011, pp. 243–275.
[24] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: online learning of social representations," in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 701–710. [Online]. Available: https://doi.org/10.1145/2623330.2623732
[25] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," vol. 2016, 07 2016, pp. 855–864.
[26] P. Thongtanunam, S. Mcintosh, A. E. Hassan, and H. Iida, "Investigating code review practices in defective files: An empirical study of the qt system," 03 2015.
[27] L. Dabbish, H. C. Stuart, J. Tsay, and J. Herbsleb, "Social coding in github: Transparency and collaboration in an open software repository," 02 2012, pp. 1277–1286.