ISSN (Online): 2456-0448
International Journal Of Innovative Research In Management, Engineering And Technology
Vol. 11, Issue 3, March 2026

AI-Driven Recommendation Engine for Open-Source Contributions

[1] Srinivas, [2] Vishal Kumar Prasad, [3] M. Keerthika
[3] Assistant Professor/CSE PMC TECH Hosur, India
[2] [3] Dept. of CSE PMC TECH Hosur, India

Abstract: ***Open-source software (OSS) development has become a cornerstone of modern software engineering, providing developers with invaluable opportunities for skill development, portfolio building, and collaborative learning. However, a significant barrier persists for beginners: the difficulty of identifying appropriate repositories, understanding where to begin, and navigating the complexity of large-scale open-source projects — commonly referred to as the "cold start" problem. This paper presents an AI-Driven Recommendation Engine for Open-Source Contributions, a web-based intelligent platform that leverages the GitHub REST API, Natural Language Processing (NLP) via TF-IDF vectorization, and cosine similarity to match users with the most relevant open issues based on their individual skill sets, programming language proficiencies, and domain interests. Issues are ranked using a composite scoring mechanism integrating semantic similarity, difficulty classification, and repository health metrics. Experimental evaluation with 50 users and 15,000 open issues across 500 repositories demonstrates Precision@5 of 0.78 and MRR of 0.84, with a 34% improvement over collaborative filtering baselines in cold-start scenarios. The system further incorporates a structured onboarding module and dynamic contribution tracking for continuous personalization.***

INDEX TERMS : Developer onboarding, open-source contribution, recommendation engine, NLP, TF-IDF, cosine similarity, GitHub API, cold start problem, issue recommendation, collaborative filtering.

I. INTRODUCTION

Open Source Software (OSS) ecosystems like GitHub and BitBucket provide developers shared and convenient technical platforms for social interactions, technical collaborations, and reputation buildings . Large numbers of professional and amateur developers contribute to OSS ecosystems to pursue their interests and goals. However, due to various technical and social barriers, beginners often face enormous difficulties in finding a suitable project or issue to contribute — commonly called the onboarding problem.

Despite the clear benefits of open-source contribution, participation remains disproportionately concentrated among experienced developers. Studies of GitHub contribution patterns consistently reveal that a small fraction of developers account for the vast majority of contributions, while the large pool of students, bootcamp graduates, and early-career professionals remains largely on the periphery. The primary reason for this disparity is not a lack of motivation, but a lack of guidance and appropriate tooling to help newcomers identify where to start.

The open-source contribution process involves several steps that can be overwhelming for beginners: identifying a project that aligns with their skills, understanding the project codebase and contribution workflow, finding an issue of appropriate difficulty, and successfully submitting a pull request. Each of these steps presents a potential point of failure, and the absence of a structured, personalized guidance system makes the experience unnecessarily intimidating.

A. Problem Statement

Current tools for discovering open-source contribution opportunities are largely generic and do not account for individual developer profiles. GitHub's built-in issue browsing and repository discovery features rely on manual searches, trending algorithms, and social signals (stars, forks), none of which are tailored to a specific developer's skill level or learning objectives. Formally, the problem is: given a user profile comprising explicitly stated and implicitly inferred programming skills, domain interests, experience level, and learning preferences, identify and rank a set of open issues from GitHub repositories such that recommended issues maximize both the probability of successful contribution and the developer's learning growth.

B. Proposed Approach

To address these challenges, we propose an AI-Driven Recommendation Engine for Open-Source Contributions. The system leverages the GitHub REST API for real-time issue data collection, applies NLP-based TF-IDF vectorization for content representation, and employs cosine similarity for semantic matching between user skill profiles and issue descriptions. A multi-factor composite ranking mechanism integrates semantic relevance, difficulty classification, and repository health metrics to produce personalized issue recommendations for developers at all experience levels.

In summary, we make the following contributions:

- We build a web-based intelligent recommendation platform for GitHub issue-level contribution discovery.
- We design a hybrid explicit-implicit user profiling approach that resolves the cold start problem from day one.
- We implement a multi-factor NLP-based ranking mechanism combining TF-IDF similarity (50%), difficulty classification (30%), and repository health scoring (20%).
- We develop a structured onboarding module and dynamic contribution tracking system for continuous personalization.

C. Paper Organization

The remainder of this paper is organized as follows. Section II presents the literature review. Section III describes existing methodology and limitations. Section IV details the proposed system architecture and methodology. Section V discusses novelty and contributions. Section VI presents results and discussion. Section VII concludes the paper.

II. LITERATURE SURVEY

The field of open-source project recommendation has attracted considerable research attention, motivated by the growing importance of OSS and the challenges faced by contributors in navigating large repository ecosystems. This section reviews the most relevant prior work, organized by primary methodological paradigm.

Behavioral and Collaborative Filtering Approaches

Sun et al. (2021) introduced a comprehensive system for personalized project recommendation on GitHub, framing the problem as a ranking task based on developer activity signals including repository starring, forking, watching, and contribution history. Their approach constructs a developer-project interaction matrix and applies matrix factorization techniques to learn latent representations of both users and projects. The system achieves high precision for active users but relies heavily on rich interaction histories, making it poorly suited for cold-start scenarios involving new or infrequent users.

Sharma and Mahajan (2020) conducted an extensive survey of recommender system paradigms applicable to the GitHub context, reviewing content-based filtering, collaborative filtering, hybrid approaches, knowledge-based systems, and utility-based methods. Their work provides a valuable theoretical foundation but does not implement or evaluate any algorithm on real-world GitHub data, leaving a gap between theoretical coverage and practical applicability.

B. Sequential and Deep Learning Models

Kim et al. (2022) proposed a sequential recommendation model for GitHub repositories that leverages deep learning architectures — specifically transformer-based sequence models — to capture the temporal dynamics of user-repository interactions. By modeling the order in which users interact with repositories, their approach learns evolving user preferences and produces recommendations that reflect current interests rather than static historical patterns. Experimental results demonstrate that sequential models outperform conventional collaborative filtering baselines on both precision and recall. However, the model requires large volumes of interaction data and remains susceptible to the cold start problem.

C. Pre-trained Language Model Approaches

Dang et al. (2023) introduced LEGION, a recommendation system that employs pre-trained language models (PLMs) to understand the semantic content of GitHub repository topics, README files, and issue descriptions. By encoding both user interests and repository content in a shared semantic space, LEGION achieves superior recommendation quality compared to keyword-matching baselines, particularly for repositories whose relevance is not captured by exact keyword overlap. The primary limitation is its significant computational overhead, which may be prohibitive for resource-constrained deployments.

D. Graph-Based Recommendation

A 2023 study published in ScienceDirect explored the application of Graph Convolutional Networks (GCNs) to open-source project recommendation by constructing a heterogeneous graph connecting developers, repositories, programming languages, and topics. The GCN learns rich structural representations by propagating information through the graph, enabling the model to capture complex, multi-hop relationships invisible to traditional matrix factorization. While the results demonstrate state-of-the-art performance on benchmark datasets, the model's complexity and computational requirements make real-time deployment challenging.

E. Summary of Literature Survey

Table I summarizes the reviewed literature. The review reveals a consistent pattern: prior work predominantly focuses on repository-level recommendations optimized for users with established contribution histories. The gap addressed by the

proposed system is the provision of fine-grained, issue-level recommendations that are personalized, computationally efficient, and effective for users with no prior GitHub activity.

Ref.	Title	Authors	Method	Advantages	Limitations
[1]	Personalized Project Recommendation on GitHub	Sun et al., 2021	Collaborative filtering + behavior ranking	High precision; continuous feedback	Cold start; requires large history
[2]	Suggestive Approaches for GitHub Recommender	Sharma & Mahajan, 2020	Survey: content-based, collab., hybrid methods	Broad theoretical foundation	No empirical evaluation
[3]	Sequential Recommendations on GitHub Repository	Kim et al., 2022	Deep learning sequential transformer model	Captures dynamic preferences	Cold start; large data required
[4]	LEGION: Pre-trained LMs for Recommendation	Dang et al., 2023	Pre-trained LM semantic matching	Superior semantic understanding	Computationally expensive
[5]	GCN-Based Open-Source Recommendation	Anon., 2023	Developer-repo graph + GCN embeddings	Rich community relationships	High computational cost

TABLE I.
Summary of Reviewed Literature on Open-Source Recommendation.

III. EXISTING METHODOLOGY AND LIMITATIONS

Existing open-source recommendation systems can be broadly categorized into five paradigms, each with characteristic strengths and limitations.

1) Behavior-History-Based Systems:

The dominant paradigm relies on developer behavior signals — repository stars, forks, watches, commits, and pull requests — to infer user preferences. These systems perform well for active users with rich interaction histories but fail catastrophically for new users who have not yet accumulated sufficient behavioral data. The cold start problem is a fundamental and largely unresolved challenge for this class of systems.

2) Content-Based Filtering:

Content-based approaches match users with repositories based on textual similarity between user-expressed interests and repository metadata such as README files, descriptions, and topic tags. While these methods can function without interaction history, they typically rely on shallow keyword matching and fail to capture semantic relationships between skills and project content.

3) Collaborative Filtering:

Collaborative filtering systems identify users with similar interaction patterns and recommend repositories that similar users have engaged with. These systems are sensitive to data sparsity and perform poorly in the long tail of the repository distribution, where few users have interacted with a given project.

4) Deep Learning Sequential Models:

Recent advances in deep learning have enabled sequential recommendation models that capture temporal user preference dynamics. However, these models require large-scale training data, involve significant computational overhead, and lack interpretability, making them difficult to deploy and debug in production environments.

5) Graph Neural Network Models:

GCN-based approaches construct developer-repository graphs and learn graph embeddings that capture structural relationships. While powerful, these models require the construction and maintenance of large heterogeneous graphs and are computationally expensive at both training and inference time.

A. Key Limitations of Existing Systems

Based on the literature review, the following critical limitations of existing systems are identified:

- Granularity: Existing systems recommend repositories or projects, not specific contributing issues. For a beginner, knowing which repository to engage with is insufficient; they also need to know which issue within that repository is appropriate for their skill level.
- Cold Start Problem: All behavior-history-based systems perform poorly for new users. Explicit user profiling through structured onboarding is not employed in any existing system.
- Difficulty Awareness: No existing system incorporates issue difficulty classification as a ranking factor, leaving beginners to manually filter through issues of widely varying complexity.
- Onboarding Guidance: Existing recommendation systems present recommendations without providing any guidance on how to proceed with a contribution.
- Dynamic Profile Adaptation: Most existing systems use static user profiles and do not update recommendations in response to a user's actual contribution activity.
- Computational Efficiency: Deep learning and graph-based approaches impose significant infrastructure requirements disproportionate for individual user recommendation tasks.

IV. PROPOSED METHODOLOGY

The proposed system is a web-based application that addresses the identified limitations through a modular, lightweight, and user-centric architecture. The system is designed around three core principles: personalization (recommendations are tailored to individual user profiles), accessibility (the system functions effectively for users with no prior GitHub contribution history), and adaptability (user profiles and recommendations evolve in response to actual contribution behavior).

A. System Architecture

The system architecture is organized into three primary layers: the Frontend Layer, the Backend Services Layer, and the External Systems Layer, as described below.

1) Frontend Layer:

The frontend is implemented as a responsive React.js web application providing four primary interface components: (a) User Interface (UI) — the primary interaction surface for end users providing navigation, profile management, and recommendation browsing; (b) Context Collection UI — a structured onboarding form capturing user skills, programming languages, domain interests, experience level, and learning goals at registration; (c) Recommendation Display — a ranked list interface presenting recommended issues with repository name, issue title, difficulty label, programming language, and similarity score; and (d) Onboarding and Feedback UI — a guided workflow module walking users through their first contribution steps and collecting feedback on recommendation quality.

2) Backend Services Layer:

The backend is organized as a set of microservices coordinated through a central API Gateway implemented in Node.js with Express. The backend comprises: API Gateway / Web Server for request routing and authentication management; GitHub Data Extraction Service interfacing with the GitHub REST API to fetch real-time repository and issue data; User Context Service managing user profiles; Matching and Ranking Engine computing TF-IDF representations and cosine similarity scores; Recommendation Service aggregating and formatting results; Onboarding Service managing step-by-step contribution guidance; and Contribution Tracking Service monitoring user activity and triggering profile updates.

3) External Systems Layer:

The system interfaces with two external systems: the GitHub REST API for repository and issue data collection, and a persistent database comprising MongoDB for document storage and PostgreSQL for structured relational data, used for user profile and recommendation history storage.

B. User Profiling and Context Collection

User profiling is a two-stage process. In the first stage, users provide explicit context during registration, including: (a) primary programming languages (e.g., Python, JavaScript, Java); (b) domain interests (e.g., web development, machine learning, DevOps); (c) self-assessed experience level (beginner, intermediate, advanced); and (d) contribution type preference (bug fixes, feature development, documentation, testing). In the second stage, users optionally connect their GitHub accounts, enabling the system to automatically extract implicit profile signals including most frequently used programming languages, contribution frequency and recency, star and fork patterns, and existing open-source contribution history. This two-stage approach ensures the system functions effectively even in the absence of a GitHub account.

C. Data Collection via GitHub REST API

The GitHub Data Extraction Service employs the GitHub REST API to collect the following data types in real time: (a) Repository Metadata — name, description, primary language, topics, star count, fork count, open issue count, last commit date, and maintainer activity metrics; (b) Issue Data — title, body text, labels (including difficulty labels such as 'good first issue' and 'help wanted'), creation date, assignee status, comment count, and linked pull request status; and (c) Repository Activity Metrics — commit frequency over the past 30/90 days, pull request merge rate, average issue resolution time, and contributor count. Issues are pre-filtered to exclude those already assigned, stale (no activity in 90+ days), or labeled as requiring specialized expertise beyond the user's stated skill level.

D. NLP Processing and TF-IDF Vectorization

Issue content is processed through the following NLP pipeline: (1) Text Preprocessing — tokenization, stop word removal, punctuation removal, and stemming/lemmatization applied to issue titles, descriptions, and labels; (2) Feature Concatenation — processed title, body, and label texts concatenated into a single document representation; (3) TF-IDF Vectorization — the TF-IDF algorithm is applied to the corpus of processed issue documents, producing sparse vector representations that emphasize terms distinctive to individual issues; and (4) User Profile Vectorization — the user's skill profile comprising programming language names, domain keywords, and stated interests is vectorized using the same TF-IDF vocabulary, enabling direct comparison with issue vectors.

E. Cosine Similarity-Based Matching

Recommendation scores are computed using cosine similarity between the user profile vector and each issue vector. Cosine similarity is defined as:

$$similarity(u, i) = (u \cdot i) / (\|u\| \times \|i\|) \quad ...(1)$$

where u is the user profile vector and i is the issue document vector. Cosine similarity is invariant to vector magnitude, focusing purely on the angular distance between representations and thus measuring alignment of thematic content rather than document length.

F. Multi-Factor Ranking Mechanism

The final recommendation score for each issue is computed as a weighted combination of three factors:

- Semantic Similarity Score (weight: 0.50): The cosine similarity between the user profile vector and the issue TF-IDF vector, capturing content-level relevance.
- Difficulty Compatibility Score (weight: 0.30): A graded score derived from issue labels (e.g., 'good first issue' = 1.0 for beginner users, 'intermediate' = 0.7, 'advanced' = 0.3), adjusted based on the user's stated experience level.
- Repository Health Score (weight: 0.20): A composite score reflecting repository activity, maintainer responsiveness, and community engagement.

The composite score formula is:

$$Score(i) = 0.5 \times Sim + 0.3 \times Diff + 0.2 \times RepoHealth \quad ...(2)$$

Issues are ranked in descending order of composite score, and the top-N recommendations are presented to the user.

G. Contribution Tracking and Dynamic Profile Update

When a user selects a recommended issue and subsequently makes a contribution (verified via GitHub pull request creation or merge events), the system updates the user's dynamic profile by: (a) incrementing skill confidence scores for technologies

used in the contribution; (b) recording the contribution in the user's history; (c) adjusting the user's experience level classification if sufficient contributions have been made; and (d) re-vectorizing the user profile to reflect accumulated experience. This dynamic update mechanism enables the system to provide increasingly accurate recommendations as the user's contribution history grows.

V. NOVELTY AND CONTRIBUTIONS

A. Issue-Level Personalization

The most distinctive contribution of the proposed system is its focus on issue-level recommendation, as opposed to the repository-level or project-level recommendations that dominate existing literature. Repository-level recommendations tell a user which project to engage with but provide no guidance on how to begin contributing. Issue-level recommendations directly identify specific contribution opportunities that match the user's skills, dramatically reducing the cognitive load and time required for a beginner to make their first contribution. This granularity of recommendation is, to the best of the authors' knowledge, not addressed by any prior system in the literature.

B. Hybrid Explicit-Implicit User Profiling

The proposed system combines explicit user-provided skill declarations with implicit GitHub profile signals, creating a hybrid profiling approach that addresses the cold start problem without sacrificing personalization quality. Existing systems rely exclusively on one or the other: purely implicit systems fail for new users, while purely explicit systems may become stale as user skills evolve. The hybrid approach provides baseline recommendations from the moment of registration and continuously improves as the user engages with the platform.

C. Lightweight NLP Pipeline

Unlike recent deep learning and graph-based approaches that require significant computational resources, the proposed TF-IDF and cosine similarity pipeline is computationally efficient, interpretable, and easily deployable on standard web server infrastructure. This design choice makes the system accessible to institutions and individuals who do not have access to GPU computing resources while still delivering recommendation quality that exceeds behavior-history-based baselines in cold-start scenarios.

D. Difficulty-Aware Multi-Factor Ranking

The integration of difficulty classification into the ranking mechanism represents a significant practical advance over existing systems, which rank repositories purely by behavioral similarity or semantic relevance without regard for the user's readiness. By calibrating difficulty scores to the user's stated and inferred experience level, the system ensures that beginners are matched with appropriately scoped contribution opportunities, maximizing the probability of successful contribution and positive early experience.

E. Structured Onboarding Module

The inclusion of a structured onboarding module that guides users through the contribution workflow is a practical innovation absent from existing recommendation systems. By providing step-by-step guidance on forking, branching, committing, and submitting pull requests, the system addresses not only the discovery challenge but also the execution challenge that prevents many beginners from converting a recommendation into an actual contribution.

F. Comparative Summary

Table II provides a comparative summary of the proposed system against existing approaches across key feature dimensions.

TABLE II.

Comparison of Proposed System Versus Existing Approaches.

Feature	Existing Systems	Proposed System
Target	Repository/project recommendation	Issue-level personalized recommendation
User Profiling	Based on historical activity (stars, forks)	Signup inputs + GitHub API + dynamic updates
NLP Usage	Limited or absent	TF-IDF vectorization + cosine similarity
Cold Start	Poor — requires large history	Resolved via explicit skill input at signup
Difficulty Ranking	Not considered	Integrated difficulty label + activity scoring
Onboarding	Absent	Built-in structured step-by-step module
Profile Updates	Static	Dynamic tracking and re-vectorization
Compute Cost	High (deep learning / GCN)	Moderate (TF-IDF + cosine)

VI. RESULTS AND DISCUSSION

A. System Implementation

The proposed system was implemented using a modern full-stack web architecture. The frontend was developed using React.js, providing a responsive and interactive user interface. The backend services were implemented using Node.js with Express, organized as modular API endpoints. The GitHub Data Extraction Service interfaces with the GitHub REST API v3, with rate limit management handled via authenticated API tokens. NLP processing was implemented using the Python scikit-learn library's *TfidfVectorizer* for TF-IDF computation and *cosine similarity* for matching. User profile and recommendation data are stored in MongoDB, while PostgreSQL is used for structured relational data. The system is containerized using Docker for deployment consistency.

B. Experimental Setup

The system was evaluated using a dataset of 500 GitHub repositories across five technology domains: Web Development, Machine Learning, DevOps, Mobile Development, and Systems Programming. A corpus of 15,000 open issues was collected from these repositories using the GitHub REST API, with issues filtered to include only those labeled 'good first issue', 'help wanted', or 'beginner-friendly'. A user study was conducted with 50 participants representing diverse skill levels and backgrounds, from computer science students to working professionals transitioning to open-source development.

C. Evaluation Metrics

System performance was evaluated using the following standard metrics: Precision@K (the fraction of the top-K recommendations rated as relevant by the user), Recall@K (the fraction of all relevant issues that appear in the top-K recommendations), and Mean Reciprocal Rank (MRR), which measures the rank of the first relevant recommendation. Relevance was determined through user feedback: after receiving recommendations, participants rated each recommended issue on a 5-point scale of perceived appropriateness to their skills and interests.

D. Performance Analysis

The proposed system achieved a Precision@5 of 0.78 and Precision@10 of 0.71, demonstrating that the majority of top-ranked recommendations were rated as relevant by users. The MRR of 0.84 indicates that the most relevant recommendation typically appeared within the top two positions. Performance was particularly strong for cold-start users (those with no GitHub contribution history), where the proposed system outperformed a behavior-history-based collaborative filtering baseline by a margin of 34% in Precision@5, confirming the effectiveness of the hybrid explicit-implicit profiling approach.

User satisfaction surveys indicated that 82% of participants found the recommended issues appropriate for their skill level, and 76% reported that the structured onboarding guidance helped them understand the contribution process more clearly than unguided exploration of GitHub. Average time to first contribution attempt was 3.2 days for users of the proposed system, compared to a self-reported average of 12.7 days for participants who had previously attempted open-source contribution without the platform.

E. Limitations and Future Directions

While the proposed system demonstrates strong performance in the evaluated settings, several limitations remain. First, the TF-IDF-based matching may underperform for issues whose relevance to a user's skills is primarily semantic rather than lexical — for example, an issue involving 'gradient descent optimization' may not be matched to a user who lists 'neural networks' as a skill. Future work will explore the integration of pre-trained language model embeddings (e.g., Sentence-BERT) for improved semantic matching, potentially in a cascaded architecture where TF-IDF provides candidate generation and a PLM provides re-ranking.

Second, the current recommendation model does not incorporate social signals such as the reputation and responsiveness of repository maintainers. Future extensions will integrate maintainer quality metrics into the repository health score. Third, expansion beyond GitHub to other platforms such as GitLab and Bitbucket would increase the scope and utility of the system.

VII. CONCLUSION

This paper presented an AI-Driven Recommendation Engine for Open-Source Contributions, a web-based intelligent platform that addresses a critical gap in existing tools and research: the provision of personalized, issue-level contribution recommendations for developers at all experience levels, with particular effectiveness for beginners entering the open-source ecosystem for the first time.

The proposed system integrates GitHub REST API data collection, NLP-based TF-IDF vectorization, cosine similarity-based matching, and a multi-factor ranking mechanism combining semantic relevance, difficulty classification, and repository health metrics. The hybrid explicit-implicit user profiling strategy effectively resolves the cold start problem that limits existing behavior-history-based recommendation systems. The structured onboarding module and dynamic contribution tracking system further distinguish the proposed platform from prior work.

Experimental evaluation with 50 users and a corpus of 15,000 open issues across 500 repositories demonstrated Precision@5 of 0.78 and MRR of 0.84, with a 34% improvement over collaborative filtering baselines in cold-start scenarios. User study results indicate high satisfaction with recommendation quality and a significant reduction in time to first contribution attempt.

By lowering the barriers to open-source participation, systems of this kind have the potential to diversify the open-source contributor community, bringing in perspectives and contributions from developers who would otherwise have been deterred by the complexity of the onboarding process. Future work will focus on integrating transformer-based semantic embeddings, expanding to multiple version control platforms, incorporating social and reputation signals, and conducting large-scale longitudinal studies of contributor retention and skill development outcomes.

REFERENCES.

[1] X. Sun, W. Xu, X. Xia, X. Chen, and B. Li, “Personalized project recommendation on GitHub,” in *Proc. IEEE/ACM Int. Conf. Softw. Eng. Companion (ICSE-C)*, 2021.

[2] S. Sharma and A. Mahajan, “Suggestive approaches to create a recommender system for GitHub,” *Int. J. Mod. Educ. Comput. Sci. (IJMECS)*, MECS Press, 2020.

[3] J. Kim, J. Wi, and Y. Kim, “Sequential recommendations on GitHub repository,” *Appl. Sci., MDPI*, 2022.

[4] Y.-T. Dang et al., “LEGION: Pre-trained language models for open-source repository recommendation,” *arXiv preprint*, 2023.

[5] Anonymous Authors, “Graph convolutional network-based open-source project recommendation,” *Expert Syst. Appl., ScienceDirect, Elsevier*, 2023.

[6] P. Resnick and H. R. Varian, "Recommender systems," *Commun. ACM,* vol. 40, no. 3, pp. 56–58, 1997.

[7] G. Salton and C. Buckley, "Term-weighting approaches in automatic text retrieval," *Inf. Process. Manage.,* vol. 24, no. 5, pp. 513–523, 1988.

[8] W. Maalej and H. Nabil, "Bug report, feature request, or simply praise?" in *Proc. IEEE 23rd Int. Requirements Eng. Conf. (RE),* 2015.

[9] GitHub Inc., "GitHub REST API documentation," *GitHub Developer Docs,* 2024. [Online]. Available: https://docs.github.com/en/rest

[10] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.,* vol. 12, pp. 2825–2830, 2011.