CS 510 Project Final Report

LLM-Assisted Open- Source Issue Summarizer and Contributor Finder

Ankit Chavan
auc3@illinois.edu
University of Illinois
Urbana-Champaign
USA

Priyam Sahoo
priyams3@illinois.edu
University of Illinois
Urbana-Champaign
USA

Rohan Patil
rspatil3@illinois.edu
University of Illinois
Urbana-Champaign
USA

Sudarshan Shinde
sshinde5@illinois.edu
University of Illinois
Urbana-Champaign
USA

Sunveg Nalwar
snalwar2@illinois.edu
University of Illinois
Urbana-Champaign
USA

Abstract

Maintainers of large open-source projects often struggle with managing a high volume of incoming GitHub issues. Understanding what each issue is about and identifying the right contributor to address it can be time-consuming and error-prone, especially as projects scale. There is currently no automated way to summarize issues or match them with developers who have worked on similar parts of the codebase.

To address this, we present *LLM-Assisted Open-Source Issue Summarizer and Contributor Finder*, a tool that leverages large language models (LLMs) to generate concise issue summaries and analyze repository history to suggest suitable contributors. By combining LLM-based keyword extraction, textual similarity techniques, and GitHub activity data, the system ranks contributors based on the relevance of their past work.

The tool is implemented as a GitHub CLI extension, making it easy to install and use within existing development workflows. It aims to reduce manual triage effort for maintainers while helping contributors find relevant tasks. We evaluate the tool on a real open-source repository and report encouraging results based on manual verification and early user feedback.

1 Introduction

As open-source projects grow, so does the number of issues reported by users and contributors. Maintainers are often left juggling hundreds of issues, trying to figure out what each one is about and who might be the right person to fix it. This process reading through issue threads, reviewing commit histories, and mentally mapping contributors to relevant parts of the code, requires significant time and effort, especially in large, fast-moving repositories with dozens or even hundreds of active contributors.

Although platforms like GitHub offer robust tools for tracking, labeling, and discussing issues, they do little to assist in understanding the actual content of an issue or routing it to someone familiar with the context. As a result, valuable time is spent on manual triage, and some issues may remain unattended simply because it wasn't obvious who should take ownership. This friction also discourages new contributors, who may be willing to help but lack the context to navigate large and complex codebases.

To address these challenges, we developed *LLM-Assisted Open-Source Issue Summarizer and Contributor Finder*, a tool designed to streamline issue triage and contributor identification using the combined power of large language models (LLMs) and repository history analysis. Given an open issue, our tool generates a short summary of the problem and extracts key technical terms. It then analyzes past commits, pull requests, and file ownership to rank contributors who are most likely to be familiar with the relevant code. These suggestions help maintainers delegate issues more effectively and allow contributors to find tasks that match their skills.

The tool is implemented as a GitHub CLI extension to ensure minimal disruption to developer workflows. It integrates directly into the terminal experience, and can be invoked with a single command. Developers can install it quickly, use it like any other GitHub command, and get immediate suggestions within their familiar development environment. This approach reflects a design philosophy focused on low-friction adoption, simplicity, and real-time utility.

In this report, we present the motivation for building the tool and walk through its system architecture and implementation. We demonstrate how it performs on real repositories, including cases where we were able to verify the relevance of its recommendations. We also provide usage instructions, example outputs, and discuss current limitations and future improvements. Our aim is to contribute a lightweight, developer-friendly solution that supports healthier, more scalable collaboration in open-source communities.

2 Background

Large Language Models (LLMs). Large Language Models are deep learning architectures trained on large-scale text corpora to generate and interpret natural language. Modern LLMs such as GPT-3 and GPT-4 [2] can perform summarization, keyword extraction, and reasoning tasks with minimal supervision. In our tool, LLMs are used to generate concise issue summaries and extract key technical terms from unstructured text.

Open Source Projects. Open source is a collaborative software development model in which the source code is publicly accessible and open to contributions from a distributed community. Developers participate by reporting bugs, opening feature requests, submitting patches, and reviewing code across a wide range of domains. As open-source projects grow in scale and contributor count, maintainers often face challenges in triaging incoming issues and identifying the most suitable contributors to address them. This creates a growing need for tools that can help summarize issue context and match them with developers who have relevant experience within the codebase making contributor recommendation a valuable addition to open-source workflows.

GitHub API. The GitHub REST API [7] allows external applications to query repository metadata, including commits, pull requests, issues, and user activity. It supports authentication via personal access tokens and enforces rate limits to prevent abuse. Our system uses the API to retrieve relevant project and contributor data in real time.

BM25. BM25 [10] is a ranking function used in information retrieval that scores documents based on term frequency and inverse document frequency, adjusted for document length. In our system, BM25 is used to match extracted issue keywords with previously submitted pull requests to identify thematically relevant contributors.

Reciprocal Rank Fusion (RRF). Reciprocal Rank Fusion [4] is a technique for combining multiple ranked lists by assigning scores inversely proportional to the rank position. It is particularly useful when multiple weak rankings need to be aggregated into a stronger consensus. We use RRF to merge contributor rankings derived from pull request relevance and file ownership history.

3 Motivation and Use Cases

In many open-source projects, triaging issues is a constant and often overwhelming task. While existing tools help track issues, the decision making involved understanding what an issue is really about and who might be able to help, still relies heavily on manual effort. This becomes especially challenging when contributors are distributed across time zones and repositories have long histories with hundreds of participants.

Maintainers frequently face the challenge of determining relevance: which issues require urgent attention, and which contributors are best suited to handle them based on recent activity or familiarity with the codebase. At the same time, developers often miss opportunities to work on issues that align with their past contributions simply because no one made the connection.

The idea behind our system is to reduce this coordination burden and make open-source collaboration more efficient. By automating some of the decision making around issue triage and contributor matching, we aim to support more responsive, transparent, and scalable project workflows.

3.1 Use Cases

- **Fast triage in high-activity repositories:** When a popular repository receives a large number of issues in a short time, maintainers can use the tool to quickly filter and route relevant ones to contributors with recent, related code activity.
- **Prioritized assignment during sprints or hackathons:** During time-bound events like release cycles or open-source sprints, contributor suggestions can help prioritize who should take on what, based on current project involvement.
- **Contributor onboarding support:** New developers can better navigate large codebases by identifying people who've worked on similar issues, giving them a starting point for guidance or collaboration.
- **Reducing bottlenecks in review and maintenance:** Instead of depending on a small set of core maintainers, projects can surface active but less visible contributors who are also qualified to handle new issues.

By surfacing the right people for the right problems, the tool aims to lower the cognitive load on maintainers and reduce entry barriers for contributors, without changing the way people already work on GitHub.

4 Implementation and Architecture

The tool is implemented as a GitHub CLI extension that helps maintainers identify the most relevant contributors for a given issue. It performs an end-to-end pipeline that includes fetching issue details, generating summaries, extracting contextual signals, and ranking contributors. The system is written in Python and leverages the GitHub REST API to access repository metadata, and the OpenAI API for LLM-based text analysis.

Figure 1 illustrates the overall architecture of the tool. When a user runs the command `git recommend owner_name/repo_name issue_id` from their terminal or development environment, the tool connects to the specified repository and retrieves metadata for the given issue, including the title, description, and labels.

This data is passed to a large language model (e.g., GPT-3.5) to generate a concise summary and extract relevant keywords. Special attention is given to preserving technical tokens such as camelCase variables and filenames, as they are highly indicative of file/module references in codebases.

The contributor matching process then branches into two ranking strategies:

- **PR-based Matching:** The extracted keywords are used to search past pull requests, which are ranked using a BM25 [10] retrieval model. Contributors associated with top-ranked PRs are extracted and scored accordingly.
- **File-based Matching:** Keywords are also mapped to source files or modules. For each matched file, the tool uses `git blame` and commit history to identify contributors, ranking them by recency and frequency of edits.

The two ranked contributor lists are merged using Reciprocal Rank Fusion (RRF) [4], a simple yet robust method that combines rankings from different sources. The final ranked list is displayed in the terminal, showing GitHub handles and relevance scores for the top suggested contributors.

The tool follows a modular architecture and is designed to be easily extensible. The CLI interface ensures a minimal learning curve for developers, allowing them to use it just like any other GitHub command.

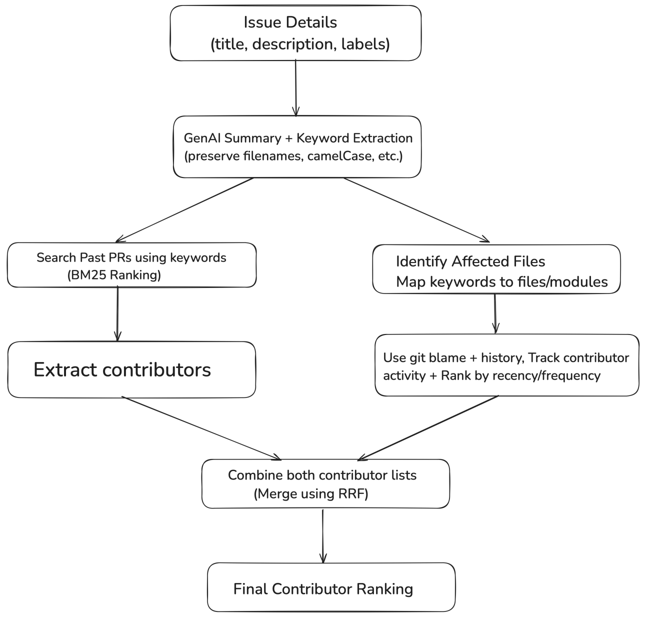

Issue Details
(title, description, labels)
GenAI Summary + Keyword Extraction
(preserve filenames, camelCase, etc.)
Search Past PRs using keywords
(BM25 Ranking)
Identify Affected Files
Map keywords to files/modules
Extract contributors
Use git blame + history, Track contributor activity + Rank by recency/frequency
Combine both contributor lists
(Merge using RRF)
Final Contributor Ranking

Figure 1: Contributor Recommendation Workflow

To summarize the logic of the pipeline, the pseudocode below outlines the main steps in contributor recommendation:

Algorithm 1 Contributor Recommendation Pipeline

```
Require: repo_name, issue_id
1: issue ← Fetch issue details using GitHub API
2: summary, keywords ← LLM_Summarize(issue.title, issue.body)

3: pr_results ← Search past PRs using BM25(keywords)
4: file_matches ← Map keywords to files/modules
5: file_contributors ← Get contributors via git blame, ranked by recency/frequency
6: pr_contributors ← Extract authors from top PRs
7: combined_ranking ← RRF_Merge(pr_contributors, file_contributors)
8: return Top-K contributor suggestions from combined_ranking
```

5 Usage Instructions

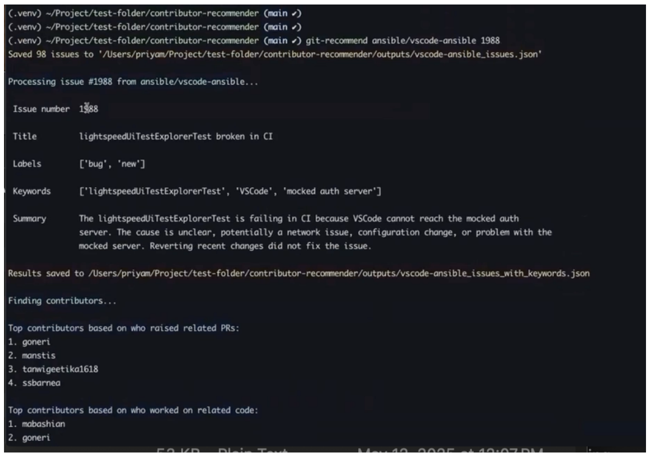
The tool is designed to work entirely from the command line and integrates seamlessly with the GitHub CLI. Users begin by cloning the repository from GitHub and navigating into the project directory. It is recommended to set up a Python virtual environment to manage dependencies cleanly. After activating the environment, the tool can be installed in editable mode using `pip install -e .`, which installs the CLI utility locally.

Once installed, the tool can be invoked using the command `git recommend <owner>/<repo> <issue_id>`. For example, running `git recommend ansible/vscode-ansible 1988` fetches the issue from GitHub, analyzes its content, and produces a ranked list of suggested contributors. The `–help` flag can be used to view supported arguments and functionality.

Figure 2 shows a sample output of the tool. The results are organized into three sections: the extracted keywords, an auto generated summary of the issue, and the final ranked list of recommended contributors. This output is designed to help maintainers quickly understand what the issue is about and identify who is best positioned to address it.

A key feature of the tool is its smart keyword extraction. Beyond standard keyword detection, the tool looks for technical tokens like camelCase identifiers, file names, or hyphenated configuration flags that often signal specific components in the codebase. These keywords are given higher weight during contributor ranking, improving the relevance of the recommendations.

The tool is also useful for new contributors. When applied to issues tagged as `good first issue`, it provides a list of experienced contributors who have previously worked on similar areas. This makes it easier for newcomers to reach out for guidance or collaborate effectively.



```
(.venv) ~/Project/test-folder/contributor-recommender (main ✓)
(.venv) ~/Project/test-folder/contributor-recommender (main ✓)
(.venv) ~/Project/test-folder/contributor-recommender (main ✓) git-recommend ansible/vscode-ansible 1988
Saved 98 issues to '/Users/priyam/Project/test-folder/contributor-recommender/outputs/vscode-ansible_issues.json'

Processing issue #1988 from ansible/vscode-ansible...

 Issue number  1988

 Title         lightspeedUiTestExplorerTest broken in CI

 Labels        ['bug', 'new']

 Keywords      ['lightspeedUiTestExplorerTest', 'VSCode', 'mocked auth server']

 Summary       The lightspeedUiTestExplorerTest is failing in CI because VSCode cannot reach the mocked auth
               server. The cause is unclear, potentially a network issue, configuration change, or problem with the
               mocked server. Reverting recent changes did not fix the issue.

Results saved to /Users/priyam/Project/test-folder/contributor-recommender/outputs/vscode-ansible_issues_with_keywords.json

Finding contributors...

Top contributors based on who raised related PRs:
1. goneri
2. manstis
3. tanwigeetika1618
4. ssbarnea

Top contributors based on who worked on related code:
1. mabashian
2. goneri
```

Figure 2: Sample CLI output: extracted keywords, issue summary, and contributor ranking

6 Evaluation and Results

We evaluated the tool on the `ansible/vscode-ansible` repository [1], which we had previously contributed to and were familiar with. This gave us a solid basis for assessing the quality of the contributor recommendations.

We tested the tool on a range of issues, including both general bug reports and those tagged as `good first issue`. The suggestions were manually verified by checking commit histories and using `git blame` to trace past contributions. In most cases, the top ranked contributors matched those who had worked on similar parts of the codebase, confirming that the ranking logic was effective and reliable.

Our hands on experience with the repository helped us validate the relevance and practicality of the tool's output.

7 Related Work

Automating various parts of the software development lifecycle has been an active area of research and engineering, especially in the context of open-source workflows. Tools like GitHub Copilot [6] have demonstrated the potential of large language models (LLMs) to assist developers with code completion and inline suggestions. However, such tools are primarily aimed at individual developer productivity and do not address broader project management tasks such as issue triaging or contributor matching.

There has also been prior work on automating issue labeling and reviewer assignment using rule-based bots and GitHub-native tools such as Probot [8] and GitHub Actions [5]. While useful in specific use cases, these systems generally rely on predefined rules or static mappings and lack a deeper semantic understanding of issue content or contributor history. Our approach differs by using LLMs to interpret issue descriptions and applying a hybrid matching strategy that incorporates both textual similarity and commit-level activity.

In the research domain, developer expertise modeling has been explored using commit graphs, file authorship, and contribution frequency, particularly in the mining software repositories (MSR) community [9]. Our tool builds on this line of work by combining file-based and PR-based signals with contextual keyword extraction to generate contributor recommendations. Additionally, recent advances in LLMs for code understanding and generation, such as those in LLM4Code [3], have shown promise in question answering and code summarization. Our system applies similar language model capabilities to a new context: augmenting open-source project maintenance workflows through intelligent and lightweight CLI tooling.

8 Limitations and Challenges

While the tool performs well in practice, several limitations and technical challenges were encountered during development:

- **Non-Deterministic LLM Responses:** The summaries and keyword extractions generated by the LLM can vary across runs, leading to slightly different recommendations. To mitigate this, we combined ranking strategies (e.g., PR-based and file-based) to stabilize the final results.
- **API Request Limits:** GitHub imposes rate limits on its API, which constrained the number of requests we could make in a given session. As a workaround, we restricted keyword-based deep searches to a maximum of three high-priority keywords per issue in this proof of concept.
- **Keyword Variations and Normalization:** Handling different forms of technically equivalent keywords such as `Auto-complete`, `Autocomplete`, or `auto_complete` proved to be non trivial. These variations impact the accuracy of similarity scoring and increase the computational overhead of normalization.

These challenges suggest useful directions for improvement, particularly in making the tool more robust and scalable across repositories with diverse styles and naming conventions.

9 Future Work

There are several directions in which the tool can be improved:

- **LLM Prompt Tuning:** We plan to enhance the prompts used for keyword extraction and potentially fine tune the language model with domain-specific data to produce more consistent and deterministic outputs. This could reduce variability across runs and improve the quality of extracted keywords, especially for technically nuanced issues.
- **Keyword Normalization:** More advanced normalization techniques will be incorporated to handle variations in keyword formatting (e.g., `Auto-complete`, `Autocomplete`, `auto_complete`) and improve matching accuracy. This includes token similarity scoring, camelCase parsing, and context-based weighting of technical terms.
- **Improved Contributor Metrics:** Future versions of the ranking algorithm will include more detailed contributor performance signals, such as the number of previously resolved issues, code review participation, commit recency, and responsiveness. These metrics will help improve the fairness and relevance of suggestions, especially in projects with overlapping contributor domains.
- **Support for Non-Code Contributions:** Many meaningful contributions happen outside of source code such as editing documentation, configuration files, and test frameworks. Future updates will include support for tracking and recommending contributors who have worked on files like `README.md`, `setup.py`, and CI/CD configuration files.
- **CI/CD Integration:** We plan to integrate the tool with GitHub Actions or similar CI/CD pipelines to trigger contributor recommendations automatically when a new issue is opened. This would enable real-time triage without manual CLI invocation, making the tool more scalable and proactive.
- **Recommendation Explainability:** To help users better trust and interpret contributor suggestions, we aim to include short rationale summaries alongside each ranked contributor e.g., "Last modified `tasks.py` in PR #1432"—based on the evidence used in ranking. This would improve transparency and user confidence in the tool's output.

10 Conclusion

This project presents a command-line tool that uses large language models and repository history to recommend contributors for open GitHub issues. By automating issue summarization and contributor matching, the tool significantly reduces the burden on maintainers and speeds up issue triage.

In addition to supporting maintainers, the tool also helps contributors especially newcomers—discover relevant issues and find meaningful ways to participate in open-source projects. The integration with the GitHub CLI ensures minimal disruption to existing developer workflows.

While our initial evaluation has shown promising results, there is still room for improvement in areas like keyword normalization, ranking heuristics, and prompt consistency. Moving forward, we aim to make this tool an essential part of open-source collaboration workflows and expand its reach through integration with IDEs and project dashboards.

Project Repository

The full source code and documentation for the tool is available at:
https://github.com/priyamsahoo/contributor-recommender

References

[1] Ansible. 2023. vscode-ansible. https://github.com/ansible/vscode-ansible. Accessed: 2024-05-16.

[2] Tom Brown, Benjamin Mann, Nick Ryder, et al. 2020. Language Models are Few-Shot Learners. *Advances in Neural Information Processing Systems* 33 (2020), 1877–1901.

[3] Mark Chen, Jerry Tworek, Heewoo Jun, et al. 2021. Evaluating Large Language Models Trained on Code. *arXiv preprint arXiv:2107.03374* (2021).

[4] Gordon V. Cormack and Maura R. Grossman. 2009. Reciprocal Rank Fusion Outperforms Condorcet and Individual Rank Learning Methods. In *Proceedings of the 32nd international ACM SIGIR conference.* 758–759.

[5] GitHub. 2021. GitHub Actions. https://docs.github.com/en/actions.

[6] GitHub. 2021. GitHub Copilot. https://github.com/features/copilot.

[7] GitHub. 2023. GitHub REST API v3. https://docs.github.com/en/rest.

[8] GitHub Probot. 2020. Probot: GitHub Apps to automate and improve your workflow. https://probot.github.io/.

[9] Shane McIntosh, Yasutaka Kamei, Bram Adams, and Ahmed E. Hassan. 2017. An empirical study of the impact of modern code review practices on software quality. In *Proceedings of the 2017 Mining Software Repositories.* 235–246.

[10] Stephen Robertson and Hugo Zaragoza. 1994. The Probabilistic Relevance Framework: BM25 and Beyond. *Foundations and Trends in Information Retrieval* 3, 4 (1994), 333–389.