2022

QuerTCI: A Tool Integrating GitHub Issue Querying with Comment Classification

Ye Paing
CUNY Hunter College

Tatiana Castro Vélez
CUNY Graduate Center

Raffi T. Khatchadourian
CUNY Hunter College

QuerTCI: A Tool Integrating GitHub Issue Querying with Comment Classification

Ye Paing
CUNY Hunter College
New York, NY, USA
Ye.Paing89@myhunter.cuny.edu

Tatiana Castro Vélez
CUNY Graduate Center
New York, NY, USA
tcastrovelez@gradcenter.cuny.edu

Raffi Khatchadourian
CUNY Hunter College
New York, NY, USA
raffi.khatchadourian@hunter.cuny.edu

ABSTRACT

Empirical Software Engineering (ESE) researchers study (open-source) project issues and the comments and threads within to discover—among others—challenges developers face when incorporating new technologies, platforms, and programming language constructs. However, such threads accumulate, becoming unwieldy and hindering any insight researchers may gain. While existing approaches alleviate this burden by classifying issue thread comments, there is a gap between searching popular open-source software repositories (e.g., those on GitHub) for issues containing particular keywords and feeding the results into a classification model. This paper demonstrates a research infrastructure tool called `QuerTCI` that bridges this gap by integrating the GitHub issue comment search API with the classification models found in existing approaches. Using queries, ESE researchers can retrieve GitHub issues containing particular keywords, e.g., those related to a specific programming language construct, and, subsequently, classify the discussions occurring in those issues. We hope that ESE researchers can use our tool to uncover challenges related to particular technologies using specific keywords through popular open-source repositories more seamlessly than previously possible. A tool demonstration video may be found at: https://youtu.be/fADKSxn0QUk.

CCS CONCEPTS

• **Software and its engineering → Software libraries and repositories**.

KEYWORDS

software repository mining, GitHub, issue comments, classification

1 INTRODUCTION

Issue tracking systems, e.g., GitHub issues, enable the discussion of software problems. Empirical Software Engineering (ESE) researchers have engaged in several activities related to mining software repositories (MSR), including studying (open-source) project issues. Researchers examine comments and threads contained in issues to discover the challenges developers face in writing software. For example, developers may struggle with incorporating new technologies, platforms, and programming language constructs and document and discuss their progress in issue "tickets."

Unfortunately, such issue discussion threads accumulate over time and thus can become unwieldy, hindering insights researchers may gain from them. Moreover, issues can be unlabeled or improperly named, making it difficult to understand the problem at hand. Approaches (e.g., [4]) exist to alleviate this burden by classifying issue thread comments; however, there is a gap between searching popular open-source software repositories (e.g., those on GitHub) for issues containing particular keywords and feeding the results into a classification model. In this paper, we demonstrate `QuerTCI`, a **Quer**y-based **T**ool for **C**lassifying GitHub **I**ssue thread comments that bridges this gap by integrating the GitHub issue comment search API with a classification model. While the default classification model used by `QuerTCI` is the one developed by Arya et al. [4], it can also use other issue comment classification models. `QuerTCI` is a Python-based tool that queries GitHub's search API for issues and comments relating to a query string that the user provides. Then, it automatically preprocesses (i.e., parses, cleans, tokenizes) each line of issue comments retrieved from the GitHub API and runs them through the pre-trained NLP model for classification.

`QuerTCI` is highly-customizable—supporting a wide range of additional functionalities—and works in either interactive and batch (non-interactive) modes. As shown in Fig. 1, users can limit number of issues retrieved (the GitHub API supports up to 1,000), as well as change the sorting criteria (which is important in capped queries like GitHub). Users may also omit particular issue comment classification categories from the results, e.g., retrieving issues that have at least one comment corresponding to a "solution discussion" [4].

Using queries, `QuerTCI` enables ESE researchers to retrieve GitHub issues containing particular keywords, e.g., those related to a certain programming language construct, and subsequently classify the kinds of discussions occurring in those issues in an integrated manor. Our hope is that, by using `QuerTCI` as part of a broader research infrastructure, ESE researchers can uncover challenges related to particular technologies using certain keywords through popular open-source repositories more seamlessly than previously possible. It alleviates the required leg work of data querying and preparation in order for the data to be: (i) compatible with the underlying comment classification model and (ii) related to particular programming language constructs (ala `gitcproc` [5]). `QuerTCI` is open-source and publicly available [13], and a demonstration video may be found at: https://youtu.be/fADKSxn0QUk.

2 ENVISIONED USERS

Since we foresee `QuerTCI` being part of a broader research infrastructure, our envisioned users are mainly ESE researchers seeking to unearth challenges developers face in particular situations or using specific technologies. For example, ESE researchers may be interested in discovering—using a keyword-based search—the kinds of discussions surrounding a particular Application Programmer's Interface (API), programming language feature, or new platform version. Using `QuerTCI`, they are able to receive—using NLP to automatically identify the topics/semantics of what is being discussed—a "quick gist" of otherwise long discussion threads commonly found in GitHub issues. ESE researchers—on a relatively large scale—can

Figure 1: `QuerTCI` interactive command-line interface.

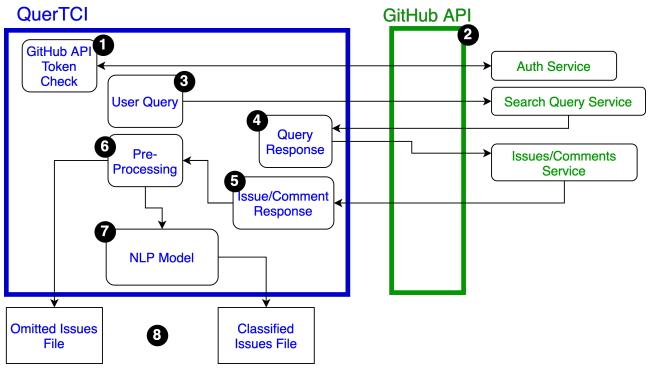

Figure 2: `QuerTCI` high-level architecture.

then quantify categories of discussions taking place under GitHub issues containing keywords of interest.

To further understand developer challenges, ESE researchers may filter for issues containing keywords corresponding to certain language constructs and having particular comment classifications. Then, they may use manual inspection to further investigate. `QuerTCI` empowers ESE researchers to narrow the scope of manual investigation so that they may focus on GitHub issues with the most relevant discussions. For instance, we may be interested in GitHub issues involving Java 8 streams (e.g., `stream()`, `parallelStream()`, `Collectors`) that include no solution discussion. The resulting set can then be further manually inspected by users to understand why issues involving these constructs cannot be solved.

Another class of users may be practicing Software Engineers that are tackling common problems pertaining to certain topics. Using `QuerTCI`, they can summarize issues that may arise for a particular topic/query string of interest. Furthermore, Software Engineers can use `QuerTCI` to filter out certain discussion types (e.g., "Social Discussion" [4]) from the tool's output, thereby saving time that might have been spent combing through large discussion threads.

Lastly, `QuerTCI` may prove useful to (CS) students studying a particular topic—looking to platforms, e.g., GitHub, to gain insight into what experienced Software Engineers are discussing. Using our tool, students can focus on issue threads that pertain to relevant categories (e.g., "Solution Discussion," "Bug Reproduction" [4]). Such expert developer discussion may prove useful to students learning a new programming language, struggling with using a new framework or library (APIs), or adopting a new software platform.

3 ADDRESSED EMPIRICAL SOFTWARE ENGINEERING RESEARCH CHALLENGES

With `QuerTCI`, we aim to reduce the time spent by researchers—and perhaps Software Engineers—in combing through long issue discussion threads for answers relating to their topic of interests. `QuerTCI` was created to help supplement ESE researchers in quantifying and ultimately understanding long issue discussion threads within GitHub issues containing particular keywords (e.g, related to certain APIs). A challenge that ESE researchers face is the time it takes to fully digest and understand issue discussion threads using manual inspection. Although GitHub issues may be "tagged," such tags may either be inaccurate nor reflect how issue discussions evolves. Moreover, issue tags, e.g., "bug," "enhancement," relate to the GitHub issue as a whole and thus may not represent the *conversations* occurring within the issue. As (manual) empirical studies are typically labor and resource intensive, ESE researchers can use `QuerTCI` to reduce the search space needed to discover answers to their (research) questions. Researchers can thus query GitHub issues for keywords and subsequently rely on a pre-trained NLP model for issue thread discussion categorization. Doing so can limit the search space (i.e., issues and discussion threads) necessary for ESE researchers to (manually) inspect.

For example, suppose that we are interested in challenges data scientists face in using a particular TensorFlow API, e.g., `tf.module`. Simply using `QuerTCI` to query for the string "tf.module" would yield a large amount of GitHub issue threads that would automatically be categorized using a pre-trained NLP model, allowing researchers to save time, gain an overview of the nature of the issues surrounding this particular query, and choose a proper subset of GitHub issues to (manually) examine in further detail.

4 IMPLEMENTATION

Figure 2 depicts the overall architecture of `QuerTCI`, which includes querying against GitHub's query service API to search for results pertaining to the input query string, preprocessing the retrieved data locally, and running the retrieved data through a pre-trained NLP model for classification. Finally, `QuerTCI` writes the results to output files on the local file system.

NLP Model, Constraints, Serialization & CLI Prototyping. The `QuerTCI` implementation uses a pre-built, customizable NLP model to classify issue comments. By default, the tool integrates a model provided by Arya et al. [4], but users may provide their own. The models are serialized using the `sklearn` [1] and `pickle` [2] Python libraries—persisting them into a file that is "imported" into `QuerTCI`. The pre-built (input) model should only be responsible for classification; models should not employ any preprocessing or tokenization of the input strings (i.e., those representing individual lines of GitHub issue comment threads). Preprocessing and tokenization—detailed shortly—are done by `QuerTCI` before running them through

the model. This implementation step involved thorough testing to ensure that integrating with the imported model did not generate any errors or data corruption. `QuerTCI`'s CLI was prototyped using an iterative process. To assess usability, we surveyed several ESE researchers in our lab (independent of this project) to understand the options needed and ease-of-use.

Interfacing with the GitHub API. Our tool interfaces with the GitHub API (Fig. 2, step 2). Our implementation lifts several burdens of ESE researchers seeking to programmatically query GitHub, including API query throttling, API key transfer, and HTTP request authentication (Fig. 2, step 1). Then, a query encompassing a search term (Fig. 2, step 3) is used to retrieve a list of GitHub issue threads (e.g., pull/patch requests,[1] issue discussion), where comments for each of the returned issue threads are extracted. This step is accomplished via several REST API endpoints, e.g., `/issues` that, given a query parameter `q`, returns a list of issues that are related to the desired query parameter string [9].

With the results from the previous search query (Fig. 2, step 4), we then query the `/comments` REST API endpoint for each of the retrieved issues. This endpoint is provided as part of the issue results. Querying this endpoint returns the list of comments for each of the issues (Fig. 2, step 5), thus allowing us to extract comment strings for further processing, cleaning, and tokenizing before they are run through the classification model. At this step, we also filter out noisy comments, e.g., for query strings that include punctuation,[2] as well as issues that do not contain any discussion.

Data Preprocessing & Tokenization. With the list of comments previously retrieved from the `/comments` REST API call, we then preprocess and clean the returned data, removing any noise and unnecessary stop words (Fig. 2, step 6). We use the list of stop words included in the `NLTK` library [12], augmenting it with several of our own custom words to further help reduce noise within the comment corpus. Additionally, we tokenize certain (common) strings in order to extract the essence of the GitHub issue text. This process mainly centers around tokenizing screen (GitHub user) names, URLs, quotes (both single and double) and code snippets (strings beginning with back ticks). Each token is then replaced with token names, e.g., `USER_NAME`, `URL`, `QUOTE`, `CODE`. Lastly, due to the way GitHub processes queries—using relaxed matching—`QuerTCI` further filters out issue comments that do not contain the original query string. This is particularly important for queries representing programming language constructs or API calls, which typically include punctuation. Issues not matching this stricter check are omitted and stored in a corresponding "omitted" file for users to inspect further if necessary (Fig. 2, step 8).

Model Classification & Result Output. The retrieved issue comment data—now cleaned and preprocessed—is fed it into the model for classification (Fig. 2, step 7). Classification results are then written to a CSV file. As comments bodies may be lengthy, each comment *line* is classified. We also list the source issue identifiers, including browser- and API-friendly URLs (not shown in Table 1) to help easily navigate to the GitHub issue via a browser for further (manual) inspection.

[1] GitHub treats issues and pull requests similarly.
[2] GitHub ignores punctuation in all query strings.

Table 1 portrays an example result CSV file snippet produced by `QuerTCI`; the complete example file may be found in our dataset [14]. Column **id** represents the unique GitHub issue identifier, column **comment line** the preprocessed, tokenized comment text, and column **category** the classification category as produced using Arya et al. [4]'s pre-built model. The results were obtained for an empirical study on the challenges facing developers in improving the run-time performance of imperative Deep Learning (DL) code using hybridization [15]. The query "`tf.function`" was used (period included) to uncover unsolved GitHub issues mentioning the *TensorFlow* [3] hybridization API keyword. The issues (filtering by `QuerTCI`) were then manually inspected.

5 EVALUATION

As `QuerTCI` is in early development stages, a thorough evaluation is pending. However, `QuerTCI` integrates several successful technologies and approaches. The GitHub API is widely used, both for industry and research. ESE researchers have successfully used the GitHub API at scale, e.g., Dilhara et al. [6] use it to discover 1,000 top-rated Machine Learning (ML) systems comprising 58 million source lines of code (SLOC). Furthermore, through qualitative content analysis of 15 complex issue threads across three GitHub projects, Arya et al.—our default NLP model—uncovered 16 different comment classification types, creating a labeled corpus containing 4,656 sentences [4]. Their model has an F-score of 0.61 and 0.42 for existing and new GitHub issues, respectively. As mentioned in §4, our tool has been used in a prior empirical study.

Our isolated preliminary assessment of `QuerTCI` involved a double-blind open card sort between two authors to independently evaluate our tool's integration with GitHub and the model of Arya et al. [4] and subsequently assess its accuracy. The authors chose a random selection of issue comment threads based on the same query and independently categorized them. The results were then compared to reach an agreed manual classification. We then used `QuerTCI` to classify the issues comments and compare if `QuerTCI` had categorized these issues in a similar way.

While initial results are promising, we plan to expand the evaluation by involving external ESE researchers. Specifically, we will recruit independent ESE researchers to use `QuerTCI` for an empirical study, e.g., one studying particular API usage. Then, we will recruit other independent ESE researchers *not* using `QuerTCI` as a control. To reduce the number of variables, each of the research teams would perform the *same* study *with* and *without* our tool. However, achieving this goal is highly unlikely as the studies will not be novel. More practically, we will use a mass survey among ESE researchers that have not used our tool and then compare the results with those where the researchers *did* use it. Although the comparison will not be completely isolated, if the scale is large enough, we foresee that the results will nevertheless be useful.

6 RELATED WORK

Casalnuovo et al. [5] present `gitcproc`, a tool for processing and classifying GitHub *commits*. Our tool is for processing and classifying GitHub issue *comments*. Like our tool, their tool is also motivated by analyzing programming language constructs using (e.g., API) keywords. However, `gitcproc` does not analyze GitHub SE artifacts at scale; each project repository must be downloaded

Table 1: Example result CSV file snippet. Issue URLs not shown. Column id is the GitHub issue identifier.

id	comment line	category
415902593	however get u step closer running original code actual error message tensorboard propagate ui CODE	Observed Bug Behavior
415902593	i think simplest fix around would call trace_on trace_export separately around graph call so something like	Workarounds
417390174	some detail i using subclassed model complex valued data	Motivation
740456602	removing tf.function decorator viable workaround best practice a related issue URL tensorflow issues/27120	Potential New Issues & Requests
755665148	74 fix acquisition optimizer	Solution Discussion
767685452	SCREEN_NAME still issue latest version coremltools if still issue please share additional code show …	Action on Issue
873531279	i similar issue please help would great	Contribution & Commitment
947976601	situation actually much worse i realised CODE CODE the following test pass	Solution Discussion
947976601	… CODE raised even though value execute error branch perhaps due tracing covering every branch this suggests …	Usage
1004824336	SCREEN_NAME could specify tensorflow version do use docker	Solution Discussion
1004824336	tf version CODE unfortunately i use docker	Usage
1004824336	thx i guess could something wrong pretraining cobblestone because i tested running pre trained cobblestone agent …	Usage
1004824336	hi i've checked sliced_trajectory data part correct may i ask chain used pretraining training part forger most likely one? CODE	Usage
1004824336	SCREEN_NAME I can reproduce reported behavior docker version also i tried reproduce without docker got error …	Bug Reproduction
1004824336	yes CODE trajectory i see problem chain for example agent place additional crafting table creating stone pickaxe look first …	Expected Behavior

locally and subsequently (serially) analyzed. `QuerTCI`, on the other hand, leverages the (indexed) GitHub API *online*, nearly instantly obtaining GitHub issues data from thousands of GitHub projects.

Arya et al. [4] users must sanitize and manually enter the issue comments as input. Our keyword *query*-based approach *automatically* interfaces with GitHub's API *directly* by sending data representing comments *only* from issues matching a particular query string. Moreover, `QuerTCI` cleans, preprocesses, and tokenizes the *automatically* retrieved GitHub data and subsequently runs the sanitized issue comment threads through Arya et al.'s model for classification. Further, the model may be interchanged (q.v. §4).

Karantonis [10] provides a multi-label prediction for GitHub issues using the *RoBERTa* NLP model [11]. Their approach, however, is for automatically assigning issue *labels* (e.g., "bug," "feature"), whereas ours classifies issue *comments* based on a keyword-based query string. Also, unlike Karantonis, who strictly relies on a Python notebook interface, `QuerTCI` provides an (optionally interactive) CLI UI using the `PyInquirer` library, enabling an interactive command menu and command-line arguments. Furthermore, `QuerTCI` automatically authenticates with GitHub's API using a supplied access token and checks for the remaining API query limit.

Fadhel [7] writes a blog post and associated iPython notebook [8] describing how to classifying discussions within code reviews that are part of GitHub pull requests. Similar to Karantonis [10], there is no GitHub integration—users must enter the data manually—and no query feature. Although pull requests are treated similarly to issues in GitHub, Fadhel's classification model is highly-tuned to code review discussions, which my not be entirely amenable to our stated use case of studying, e.g., usage of particular APIs.

7 CONCLUSION & FUTURE WORK

An open-source, publicly available tool [13]—as part of a broader research infrastructure—to help ESE researchers quantify the types of discussions in GitHub issue comment threads around a particular query string of interest has been demonstrated. `QuerTCI` is implemented in Python and uses libraries, e.g., `NLTK`, for string preprocessing and NLP model loading to automatically classify each of the strings. `QuerTCI` also interfaces with GitHub's API and features a (optionally interactive) CLI UI. As the tool is in its early stages, plans for a fuller evaluation were discussed.

In the future, we plan to expand onto other platforms such as Stack Overflow to also process developer Q&A posts. To further enhance performance, we will classify each issue thread as they are retrieved from GitHub's API instead of waiting for all to be retrieved. Other future plans include exploring alternate tool forms, e.g., browser extensions, and performing a thorough evaluation.

REFERENCES

[1] [n. d.] Retrieved 02/16/2022 from http://scikit-learn.org/stable.
[2] [n. d.] Retrieved 02/16/2022 from http://docs.python.org/3/library/pickle.html.
[3] Martín Abadi et al. 2016. TensorFlow: a system for large-scale Machine Learning. In *Symposium on Operating Systems Design and Implementation*.
[4] Deeksha Arya et al. 2019. Analysis and detection of information types of open source software issue discussions. In *ICSE*. DOI: 10.1109/ICSE.2019.00058.
[5] Casey Casalnuovo et al. 2017. GitcProc: a tool for processing and classifying GitHub commits. In *ISSTA*. ACM, 396–399. DOI: 10.1145/3092703.3098230.
[6] Malinda Dilhara et al. 2022. Discovering repetitive code changes in Python ML systems. In *ICSE*. To appear.
[7] Muntazir Fadhel. 2018. Dissecting GitHub code reviews: a text classification experiment. Retrieved 11/24/2021 from http://mfadhel.com/github-code-reviews/mfadhel.com/github-code-reviews/.
[8] Muntazir Fadhel. 2020. What code reviewers talk about. (December 17, 2020). Retrieved 11/24/2021 from https://git.io/JMTiL.
[9] GitHub, Inc. 2021. REST API. Retrieved 11/24/2021 from https://git.io/JMJYC.
[10] Giorgos Karantonis. 2021. Predicting issues' labels with RoBERTa. (March 12, 2021). Retrieved 11/23/2021 from https://git.io/J1jBr.
[11] Yinhan Liu et al. 2019. RoBERTa: A robustly optimized bert pretraining approach. (2019). arXiv: 1907.11692 [cs.CL].
[12] Edward Loper and Steven Bird. 2002. NLTK: the natural language toolkit. (May 2002). Retrieved 02/16/2022 from http://nltk.org. arXiv: cs/0205028 [cs.CL].
[13] Ye Paing, Tatiana Castro Vélez, and Raffi Khatchadourian. 2021. ponder-lab/GitHub-Issue-Classifier. (March 25, 2021). DOI: 10.5281/zenodo.4637636.
[14] Ye Paing, Tatiana Castro Vélez, and Raffi Khatchadourian. QuerTCI: a tool integrating GitHub issue querying with …. DOI: 10.5281/zenodo.6115404.
[15] Tatiana Castro Vélez et al. 2022. Challenges in migrating imperative Deep Learning programs to graph execution: an empirical study. In *MSR*.