# Automatically Augmenting GitHub Issues with Informative User Reviews

Arthur Pilone*, Marco Raglianti†, Michele Lanza†, Fabio Kon*, Paulo Meirelles*
*Institute of Mathematics and Statistics — University of São Paulo, Brazil
†REVEAL @ Software Institute — USI, Lugano, Switzerland

***Abstract*—Development teams for mobile applications can receive thousands of user reviews daily. At the same time, these developers use different communication channels, such as the GitHub issue tracker. Although GitHub issues are accessible and manageable for developers, their content often differs starkly from what users write in app reviews. Issues may lack steps to reproduce bugs or insights that justify the priority of new feature requests. The sheer volume of user reviews for a popular app, combined with their heterogeneity and varying quality, makes manual integration into issue trackers unfeasible.**

**We present an approach that automatically augments GitHub issues with informative user reviews to bridge the gap between user feedback and developer-managed issues. Using a state-of-the-art large language model (LLM), our approach automatically retrieves user reviews with high semantic textual similarity (STS) to the issue content and suggests reviews that augment developers' understanding of the issue. In this paper, we present large-scale quantitative and qualitative analyses to assess the feasibility of enriching development workflows with user-written information. Using over 37,000 issues and 750,000 reviews from 19 popular Free/Libre/Open Source Software (FLOSS) mobile applications, our approach augments 3,017 (8%) issues with 7,287 (1%) potentially informative reviews. In addition to providing insights into user-reported bugs and feature requests, the information from these matches points toward a novel and promising way to leverage user reviews for concerted app evolution.**

***Index Terms*—Semantic Textual Similarity, User Feedback Mining, GitHub Issues, Information Retrieval, Software Repository Mining**

## I. Introduction

With over 1.5 million apps,[1] the Google Play Store is not only home to software products but also a repository for millions of user reviews posted daily. Previous works have explored classifying, understanding, and using these reviews to support app development [5], [24], [26], but the sheer volume of reviews presents both a curse and a blessing to software engineers. The abundant user input constitutes a potential source of valuable insights rarely elicited by the experience of a single user. These insights might never be found in a pool of millions of messages, reviews, and comments.

To enhance the information available to developers with the users' perspective, some teams openly expose users to their backlogs by integrating service desks into their workflows.[2] Other teams encourage users to file inquiries directly in their issue trackers [10], often with a high technical entry barrier.

[1]https://www.appbrain.com/stats/number-of-android-apps
[2]https://docs.gitlab.com/user/project/service_desk/

These solutions ignore the immense volume of reviews that users willingly and collectively provide for popular apps. Consequently, these reviews remain overlooked in app stores.

The potential of user review integration in development workflows is still largely untapped, but before developers can use the contents of such reviews in actionable ways, two problems need to be addressed. First, valuable information might be hidden among mountains of irrelevant noise [24], [37]. Second, there is a disconnect between the language of users in reviews and the one developers use to write issues in trackers [14], [40]. Making *informative* reviews promptly available to developers allows to augment issues with additional details. For example, for a bug-fixing task, reviews might provide information for a specific software and hardware configuration [32], [40]. Similarly, a feature request [12] could benefit from insights directly from the intended audience.

We propose a novel approach that leverages the growing potential of dense vector embeddings for information retrieval [16], [22] to automatically augment issues with related reviews on a large scale. Our approach extracts all potential connections between issues and reviews by computing semantic textual similarity (STS) [2] based on the distance between their dense vector embeddings, obtained using a state-of-the-art Large Language Model (LLM) [34].

We evaluated our approach on a diverse dataset to understand how STS-based retrieval can be tailored to support software maintenance and evolution. Based on the DATAR dataset [1], we analyzed 19 Free/Libre/Open Source Software (FLOSS) mobile apps from different categories (*e.g.,* tools, games, media players), each with over 10,000 reviews and 100 issues.

Additionally, we performed a qualitative analysis with two objectives: (*i)* To determine whether high similarity scores correspond to reviews highly pertinent to an issue and (*ii)* to investigate whether the highest similarity matches contain valuable information to augment the corresponding issues. A case study with the Brave browser Android app illustrates how informative matches can complement both bug reports and feature requests.

Our approach excels at identifying related user reviews, achieving a precision of 88% for matches above a threshold of 85% of similarity. One in every three issues is matched with reviews deemed informative. Our analyses confirm the feasibility of supporting software maintenance tasks by automatically augmenting issues with informative reviews.

 DOI: 10.1109/ICSME64153.2025.00045

## II. Background and Motivation

The potential of augmenting issues with information from user reviews relies on the assumption that user reviews contain information useful for the development process. While manageable for small apps, manually reading and filtering user reviews can be unprofitable or simply unfeasible for teams maintaining apps receiving thousands of them weekly. In a survey by van Oordt and Guzman [37], 83% of practitioners reported analyzing user feedback manually, and 70% of the study participants agreed that analyzing feedback is a time-consuming task. We aim to automate the retrieval part by selecting the relevant reviews among the others.

Part of the complexity of analyzing user reviews comes from their varying quality. Not all user reviews contain valuable information for the developers. Previous works have found that only around 33% of user reviews can be of immediate value to development teams [24], and that only around 35% of them contain meaningful information [5]. Hence, the design of an automatic retrieval approach must take into account the different grades of usefulness any review might have.

### A. Related, Pertinent, and Informative Reviews

We propose three classes, depicted in Figure 1, to characterize the relationship between an issue and a review.

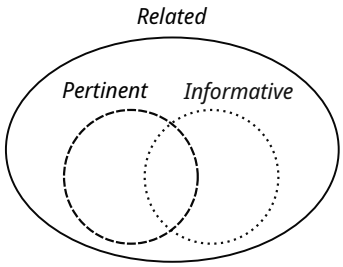


Fig. 1. Euler diagram with the review classes we propose.

We define reviews as **related** if they pertain to one of the topics mentioned in the issue. For instance, considering an issue about the authentication mechanism of an app, any review describing actions such as signing up, logging in, or logging out is related.

In practice it is common to find issues at a narrow and well defined intersection of different topics. Accordingly, we define reviews as **pertinent** if they are closely related to all the topics mentioned in an issue. For example, an issue about the occurrence of a crash when a user interacts with any text field would be related to reviews commenting on problems occurring after interacting with a text field after a fresh install. As shown in Figure 1, any pertinent review is also related to the same issue.

Different developers perceive the relatedness between issues and reviews with varying strength. To partially mitigate the subjective measure of pertinence as the strength of perceived relatedness, we prefer the use of relevance for a partial overlap of a broader topic and pertinence when a narrower focus is present in both reviews and issues.

Finally, we define **informative** reviews as those related to an issue and containing meaningful information absent from the issue title and description. We hypothesize that the majority of the informative reviews are also pertinent to the issue, we verify this hypothesis in Section V. However, a partly related but not pertinent review can still be informative, containing details that developers could abstract into useful insights. Not only is it common to find user reviews with incomplete information [37], [40], but developers might also gain unexpected insights while reading reviews on related topics. For example, two crashes happening for different actions might have a common root cause in the same malfunctioning user interface component.

Ultimately, the new insights present in informative reviews can augment the issue and support developers to tackle and resolve it. Consequently, **any approach that intends to augment issues must be capable of automatically retrieving potentially informative reviews**. While there is no quantitative metric to how informative a review is, we argue that pertinence itself is a metric of semantic similarity and that a high pertinence is a necessary condition for being informative. As such, we obtain informative reviews by retrieving those highly pertinent to the issue, by leveraging text embeddings.

### B. Related Work

While our approach uses text embeddings to bring to light the insights present in informative reviews, previous works have explored eliciting insights from the distribution of large amounts of user comments into categories and topics. There are methods to aggregate reviews of similar usefulness for the developers [17], [26], [33]. Others cluster reviews by the mentioned topics, supporting the identification of recurring trends among large sets of reviews [5], [9], [24], [28], [30].

In particular, Chen *et al.* [5] proposed AR-Miner, which clusters reviews about the same topic, filtering and ranking reviews by how "informative" they are. While our definition of informative consists of reviews that augment issues with new information, Chen *et al.* use the same term to characterize reviews that might simply contain actionable information, with no links to any issue in particular.

Panichella *et al.* [27] presented AR-Doc, which classifies app reviews using feature extraction and sentiment analysis. Similarly, Scalabrino *et al.* [30] proposed CLAP, a tool that suggests possible emergent user insights through fine-grained review clustering. Wang *et al.* [39] advance the study on app review clustering using their BERT+Attr-CRF approach, which leverages advanced deep learning techniques and the BERT model [7] for feature extraction.

Previous work has also explored different techniques that augment development issues, for example by analyzing and supporting issue comprehension [8], [10], [12]. Hooimeijer and Weimer [13] and Zimmerman *et al.* [40] investigated the characteristics that assist the comprehension and resolution of bug reports. Other works studied how artifacts such as issue links [19] and tags [31] reflect themselves on how a project is developed.

Li *et al.* [20] analyzed the impact of issue templates, and Kuramoto *et al.* [18] studied whether visual elements (images, videos) influence how long developers take to close an issue.

Palomba *et al.* [25] proposed CRISTAL, an approach matching reviews, pre-filtered as informative, with issues and commits. Their approach identifies links using the dates of reviews and issues and the asymmetric Dice similarity coefficient [3] on their texts. However, using dates to filter out relevant issues can hide valuable insights on recurring bugs and requests, which our approach can capture.

Recent work has started exploring how text embeddings might support software maintenance. Haering *et al.* [11] proposed the DEEPMATCHER approach that, although restricted to using only issue titles and pre-filtered bug reports, was the first to bring the use of text embeddings to look for links between app reviews and bug reports. DEEPMATCHER identifies related reviews and issues describing a problem only using the embeddings of the nouns found in the review. Our approach, in comparison, uses a unified embedding for each review and issue, is unrestricted on user feedback type, and focuses on identifying informative reviews.

Also using text embeddings for matching bug reports and app reviews, Tang *et al.* proposed BUGRMSYS to support *collaborative bug finding* [35]. Under the premise that apps of the same category often suffer from common bugs, BUGRMSYS uses text embeddings to search for resolved bug reports from similar apps with high similarity to reviews from a target app.

Other authors studied the potential of text embeddings for identifying related artifacts in different contexts. Liu *et al.* [21] used text embeddings to identify links between app reviews and app release notes. Meanwhile, Tizard *et al.* [36] used the embeddings generated by the Universal Sentence Encoder (USE) [4] model to identify links between forum posts, issue tracker entries, Frequently Asked Questions (FAQs), and other documentation sources.

It is still unclear how embeddings-based artifact retrieval can be affected by factors such as the project size or the target audience. Similarly, no previous study has investigated whether STS can be used to retrieve reviews that are sufficiently related to an issue while still containing new and valuable information it might lack. Our work investigates whether highly pertinent reviews are as informative as those partially pertinent. We analyze if there is any tendency of highly pertinent reviews to be informative more frequently. In this case, an approach identifying highly pertinent reviews could provide a reliable automation for retrieving the informative ones.

**Text Embeddings:** The recent popularization and ensuing evolution of Large Language Models (LLMs) have fostered the use of a new mathematical representation for the semantic value of words and sentences: Dense vector embeddings, popularly known as *text embeddings*.

Domain analysis and topic modeling techniques have been widely used to identify reviews that pertain or relate to one another [5], [9], [30]. Vector representations such as Term Frequency-Inverse Document Frequency (TF-IDF) [29] were used to quantify the similarity between app reviews [33]. However, these techniques fail to account for semantically equivalent expressions (synonyms) [16], which, paired with the language gap between the vocabulary used by users and that of developers [14], [40], constitutes a challenge to bridging texts produced by the two groups.

The mathematical representations internally used by LLMs have recently become an attractive alternative for supporting natural language processing tasks with heavy emphasis on semantics [11], [36]. As LLMs encode their inputs, each text fragment is embedded into a dense vector composed of hundreds to thousands of floating-point values.

The dense vector embeddings adequately store semantic values, so that words or fragments with similar meanings are encoded onto neighboring vectors [15], [16]. As such, the proximity between embeddings can be used to quantify, with a value ranging from 0 to 1, the Semantic Textual Similarity (STS) between words, sentences, or documents [22].

Following the popularization of the self-attention mechanism [38] and of bidirectional encoders, such as `BERT` [7], the text embeddings computed by LLMs have become a straightforward and reliable means of manipulating semantic information. Models implementing both mechanisms compute *contextualized embeddings*. After an embedding is computed for each token (text fragment), the attention mechanism updates the embeddings using the values of surrounding input tokens, imbuing the embedding of each token with the context in which it appears. Contextualized embeddings overcome the barriers of synonyms and word sense ambiguity [16].

## III. RETRIEVING PERTINENT USER REVIEWS

In Figure 2, we show an overview of our approach to identify pertinent reviews and augment GitHub issues. We estimate the Semantic Textual Similarity (STS) between app reviews and software repository issues via their text embeddings.

The first step is to collect all the project issues from the issue tracking system of the repository, in this case GitHub Issues. We filter issues based on parameters such as labels and tags (*e.g.,* consider issues on AndroidOS in case of multi-OS repositories). We gather titles and bodies and perform a textual cleaning step (*e.g.,* removing said tags when used for filtering). We remove markdown syntax elements, including attachments and typesetting marks (*e.g.,* bold, italics). Besides shortening the content for the embedding context, this step reduces spurious semantic similarities due to recurring patterns in the markdown syntax. In parallel, we gather all English written reviews from the app page on the Google Play Store.

Next, we concatenate issue title and body in a single text and compute the *text embedding* of every review and issue using a specialized encoder, which computes a single embedding for every document. For our implementation we referred to the MTEB benchmark[3] [22] and chose the `jina-embeddings-v3` [34] model, a multi-lingual model based on `XLM-RoBERTa` [6].

[3]https://huggingface.co/spaces/mteb/leaderboard_legacy

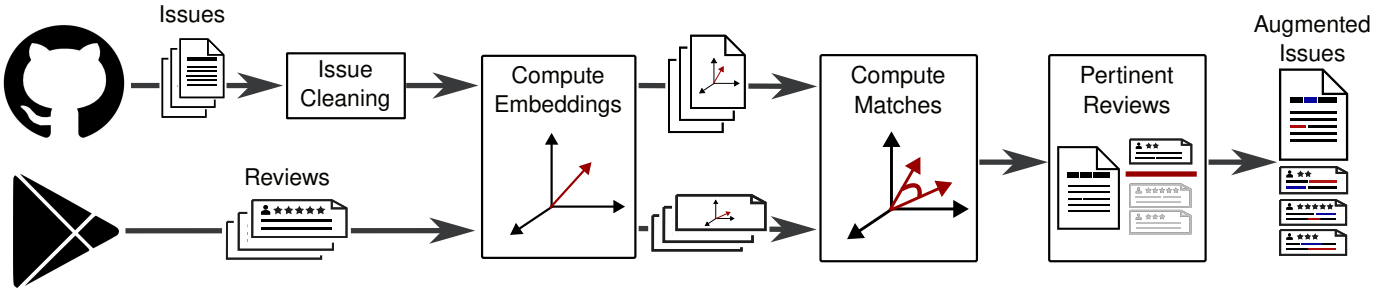


Fig. 2. Approach Overview.

At the time of the study, the model appeared in the MTEB benchmark as the second highest ranked model in the English written datasets of the STS task. We chose `jina-embeddings-v3` instead of the highest ranked model (`bilingual-embedding-large`[4]) because of its longer context length, capable of embedding issue bodies or even whole discussions with up to 8,194 tokens, and due to its multi-lingual capabilities, not being restricted to English or French, to explore potential benefits for the different languages used by users and developers.

The time taken to compute all embeddings grows significantly with the project popularity (number of reviews) and development history (issues), so we store them in a vector database, where each document is indexed by its embedding. The database supports efficient lookups of similar embeddings when given a metric to compute similarity. For every issue, we obtain highly pertinent reviews by retrieving the five reviews with the highest cosine similarity to the issue.

Ideally, the retrieved reviews are presented to the developer tackling the chosen issue, along with data such as app version and review publication date. In a completely automated context, the approach could include the potentially informative reviews directly on the GitHub issue description or discussion. Then, the developer could also use the reviews to understand and solve the issue at hand.

## IV. Research Design and Empirical Evaluation

As informative reviews can only augment issues they are related to, a mechanism that fails to identify related reviews cannot be used to augment issues automatically. Similarly, if highly pertinent reviews are rarely informative, then a mechanism that reliably retrieves pertinent matches will also fail at automatically augmenting issues.

Our approach relies on two assumptions: STS can be used to retrieve pertinent reviews, and highly pertinent reviews are informative. To validate the feasibility of our approach, we derive the two research questions that guide our empirical evaluation:

***$RQ_1$: What is the relationship between the similarity score and the frequency of* pertinent *matches?***

***$RQ_2$: How often are highly* pertinent *reviews also* informative*?***

[4]https://huggingface.co/Lajavaness/bilingual-embedding-large

Our evaluation aims to identify where reviews with high STS lie on the Euler diagram from Figure 1. Moreover, part of our analysis investigates how the classes shown in the diagram interact and intersect one another. We analyze these questions on a varied sample to test if they apply to apps of different categories, project sizes, and popularity. Additionally, we are also interested in observing how these factors impact the approach results.

### A. Data Collection

We start from a set of popular apps with openly available issue trackers. While there are alternative app repositories exclusively dedicated to FLOSS apps, such as F-Droid,[5] they rarely support user reviews. Moreover, these alternative app repositories commonly lack popular FLOSS apps present on the Google Play Store.[6]

Since there is no category or filter on the Google Play Store for retrieving FLOSS apps, we start from lists of FLOSS Android apps using a dataset published by Abedini *et al.*, DATAR [1]. The dataset contains over 1,300 apps, including the GitHub repository identifier, Android package name, and number of Play Store reviews of each app.

While DATAR includes apps from hundreds to millions of users, our approach is designed to support the maintenance of apps with a high volume of incoming user feedback. Therefore, we progressively filter the dataset entries until we obtain a small collection of large enough apps.

We filter based on the number of reviews. To exclude apps with a low influx of user comments, we chose to filter only apps with at least 10,000 English-written reviews. Since the DATAR dataset includes the number of reviews on all languages, we had to collect each app's reviews to obtain those written in English.[7] We obtained a potential sample of 21 apps.

We collected open and closed GitHub issues from the remaining apps and excluded two projects with fewer than 100 issues, resulting in a final dataset of 19 projects (Table I).

After having obtained the dataset of apps for our analyses, we removed issues written in languages other than English.

[5]https://f-droid.org/

[6]https://gitlab.com/fdroid/fdroiddata/

[7]To conservatively account for the possible outdatedness of the DATAR statistics, we screened all the apps (193) with at least 100 user reviews worldwide according to DATAR, some of which we found to have an effective number of reviews one order of magnitude higher than the reported one [1].

TABLE I
DESCRIPTIVE STATISTICS OF THE APPS IN THE ANALYZED DATASET.

| | App | Category | Reviews | Issues | Commits | Contributors |
|---|---|---|---|---|---|---|
| $A_1$ | DuckDuckGo Private Browser | Tools | 235,741 | 664 | 5,308 | 92 |
| $A_2$ | PPSSPP - PSP emulator | Games | 104,652 | 331 | 42,309 | 397 |
| $A_3$ | Quran for Android | Books | 69,637 | 1,052 | 4,629 | 71 |
| $A_4$ | Sky Map | Reference | 45,683 | 228 | 666 | 35 |
| $A_5$ | Barcode Scanner | Tools | 41,555 | 1,201 | 3,823 | 129 |
| $A_6$ | Kodi | Video Players | 41,109 | 189 | 67,628 | 899 |
| $A_7$ | WordPress - Website Builder | Productivity | 31,424 | 8,758 | 96,786 | 214 |
| $A_8$ | Proton VPN: Private, Secure | Tools | 27,904 | 122 | 3,279 | 19 |
| $A_9$ | lichess • Free Online Chess | Games | 20,125 | 2,017 | 6,965 | 78 |
| $A_{10}$ | K-9 Mail | Communication | 17,204 | 4,769 | 15,178 | 410 |
| $A_{11}$ | AnkiDroid Flashcards | Education | 16,961 | 8,101 | 20,262 | 465 |
| $A_{12}$ | SMS Backup+ [8] | Tools | 16,309 | 874 | 1,788 | 53 |
| $A_{13}$ | Shattered Pixel Dungeon | Games | 14,362 | 1,882 | 8,036 | 2 |
| $A_{14}$ | Loop Habit Tracker | Productivity | 13,093 | 700 | 2,535 | 64 |
| $A_{15}$ | Olauncher. AF Launcher | Personalization | 12,317 | 427 | 890 | 26 |
| $A_{16}$ | Mindustry | Games | 12,177 | 1,189 | 18,698 | 655 |
| $A_{17}$ | AntennaPod | Music & Audio | 11,695 | 4,215 | 9,036 | 298 |
| $A_{18}$ | Bitwarden Password Manager | Productivity | 10,618 | 974 | 3,998 | 38 |
| $A_{19}$ | Sabbath School & PM | Books | 10,109 | 114 | 2.102 | 8 |

As a consequence of the predominance of English in technical issues, this restriction had a weak impact on the size of our dataset: Out of the 54,957 issues, only 713 (1.3%) had titles written in languages other than English.

A manual inspection of the GitHub repositories from the 19 projects revealed that four of them were shared for targets other than Android. Based on information from the issue templates and labels, we were able to identify 16.437 issues that only affected desktop or IOS users. The 19 selected apps comprise a total of 37,807 issues and 752,675 reviews, all of which are shared in our replication package.[9] Table I shows the apps we used for the evaluation, along with their number of issues, commits, contributors, reviews, and category as reported from the Google Play Store. It is worth noting that our dataset includes apps from 10 different categories, as well as three different orders of magnitude in the number of commits and contributors. Representing 1.5% of the projects from the original DATAR dataset, our sample is specific to the app popularity that we sought to encompass. Our sample is diverse [23] enough to favor reflections on how the approach may generalize to other projects of similar popularity.

### B. Study Methodology

Both $RQ_1$ and $RQ_2$ involve quantitative and qualitative aspects. Our large sample offers our quantitative analysis insights on how the STS computed by the approach may change from project to project. Subjective aspects such as how pertinent or informative a review is require manual qualitative analysis: Our study consists of two different parts.

First, we applied the approach to the 19 apps from our dataset, storing the five reviews with higher STS for every issue. Our replication package includes all 189,035 review suggestions and the computed STS value.

[8]SMS Backup+ is currently unavailable on Google Play and F-Droid as its maintainers adapt to the loss of a core maintainer.

[9]https://figshare.com/s/4d65282d72bb47c22ec3

Then, we conducted a qualitative analysis over the suggested reviews. Following $RQ_1$ and $RQ_2$, our analysis must first assess, for every issue in the sample, whether each suggested review is related, pertinent, and informative.

At an average of 2 minutes per issue, an estimate found with a preliminary pilot experiment, reviewing all 189k matches suggested for the dataset would take over 2,520 hours. Instead, we adopt a statistically significant sample of randomly selected issues. Our sample includes 400 issues and grants our analyses a confidence level of 95% with a margin of error of less than 5%. We preserved the proportion of issues per project constant between the original population and our sample.

A survey presents the reviews found for each issue to two different respondents, the first author and 1 of 10 external participants, comprising 7 mobile application developers and 3 software engineering researchers. Every external participant was given a stratified sample of 40 issues. After reading the title, description, and timeline of an issue, the respondents read the five user reviews with highest STS to the issue. For every review registered as *related*, the respondents checked whether the issue is *informative* and chose a value on a 5-point rating scale to indicate its *pertinence* to the issue.

We consider as informative only the reviews that were marked as informative by both respondents. A single pertinence score is attributed to each review by the arithmetic mean of the pertinence scores of the respondents that reviewed it. If the respondent marked the review as unrelated, this score is 0. Otherwise, it corresponds to the pertinence value between 1 and 5 as chosen by the respondent.

The pertinence score provides a rough gauge to how intensely a review pertains to a related issue. We expect a notable number of minor disagreements between respondents as each subconsciously develops their own criteria. A review with a pertinence score of 5 for one might be only scored a 4 from another.

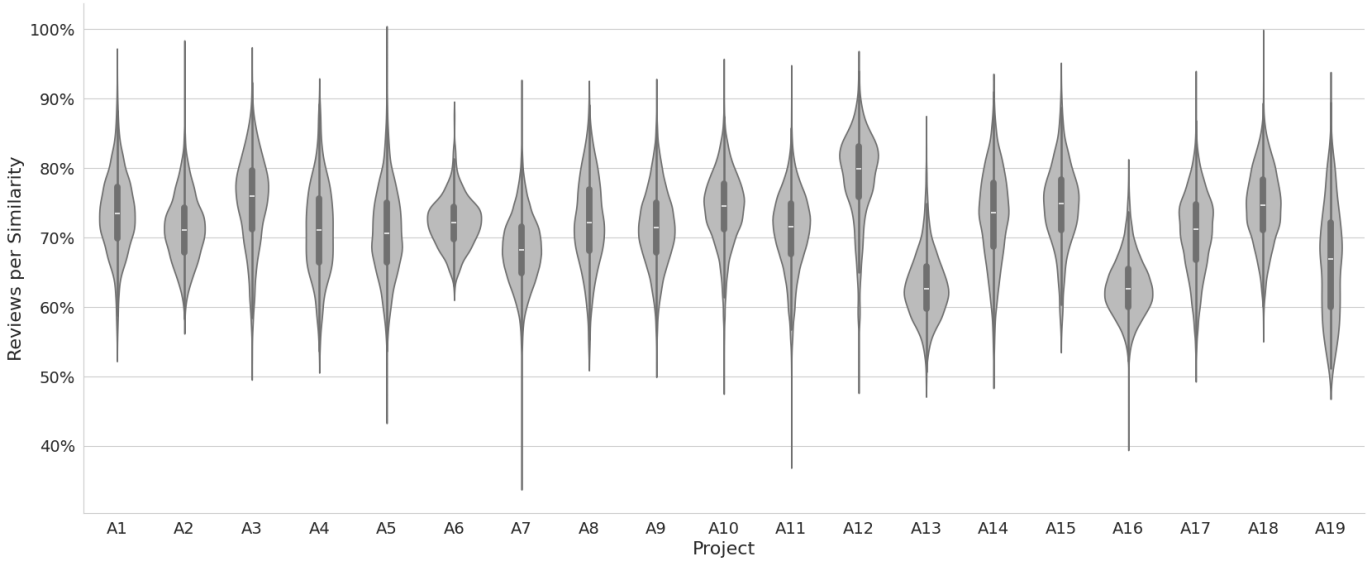


Fig. 3. Distribution of STS values for the matches of different projects.

Scores given on opposite ends of the pertinence scale, however, indicate a major difference between the participants' opinions, as a highly pertinent review to one might become a slightly pertinent review to another.

We observed disagreements on whether some reviews were highly or slightly pertinent and, simultaneously, whether they were informative. For these cases, one of the other authors served as a third respondent. Each review with such disagreements was considered informative if 2 out of the 3 respondents marked it as informative. The pertinence score for the conflicted reviews corresponds to the mean of the 2 closest pertinence values given, which necessarily fit the same side of the 5 point rating scale.

## V. Evaluation Results

Despite the inherent subjectivity of the perception of pertinence, the respondents disagreed on no more than 2 points on the pertinence scale for 78.9% of all the 2,000 reviews. Similarly, they agreed on whether the reviews were informative or not in 85.9% of the cases. Only 83 (4.2%) reviews had substantial disagreements and were reviewed thrice.

After the third review for disagreements, 69 (3.5%) reviews were marked as informative by two respondents in 56 issues (14.0%), but a total of 282 (14.1%) reviews were marked as informative by at least one respondent. Although our analyses consider only the reviews marked as informative twice, the number of informative reviews could be up to four times larger.

After pondering the factors that may influence the distribution of similarity values, we use the results of our quantitative and qualitative analyses to answer our two research questions.

### A. Overall Metrics

Figure 4 depicts the histogram for the similarity values of all matches, including every issue and review from our dataset. We note its close resemblance to the bell shape typical of normally distributed variables.

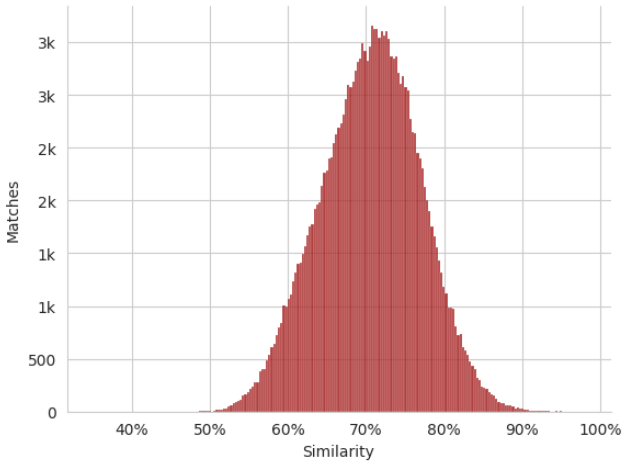


Fig. 4. Histogram of similarity scores on matches from our dataset.

The normality of the similarity values is confirmed with a statistic of 0.999 on the Shapiro-Wilk test. Although the similarity value could, in theory, range from 0 to 100%, the mean of the similarity distribution for the top-5 matches was not centered on the interval middle point. The matches suggested by the approach have a mean similarity of 70.5% and a standard deviation of 6.4% similarity.

We analyzed tendencies of the similarity distribution in each project and characteristics such as the number of commits, contributors, issues, and app reviews. Figure 3 contains the similarity distribution for the 19 apps in our dataset, ordered by decreasing number of app reviews from left to right. While the variance of the similarity values appears to differ from project to project, most projects have a similar mean.

We used our extensive dataset to investigate the relationships between issue and review length and the similarity values computed by our approach. We observed the top matches, revealing no significant correlation between the length of issues or reviews and the similarity values of the matches. Similarly, no correlation was found between the similarity scores and the differences in length between issues and reviews. This lack of meaningful correlation suggests that the STS computed for these artifacts does not exhibit a clear tendency to increase or decrease as the length of the reviews and issues varies.

When analyzing our dataset for correlations with the project mean similarity values, most variables exhibit very weak correlation. These include the number of reviews (0.18), commits (0.13), and contributors (-0.19). All accounting for small Pearson correlation coefficients.

The only significant correlation we have identified is between the mean similarity value for an app and its number of issues (0.57). The number of projects in our sample is sufficient to identify a tendency for those with more issues to have a higher average similarity for their suggestions.

The standard deviation of the project similarities is strongly correlated with two other project variables: The number of contributors (-0.72) and the mean issue length (-0.67). Projects with more contributors or longer issues tend to have more predictable similarity values for their matches.

***Overall Metrics: Key Results***

- $R_1$ *The similarity of the top-5 highest matches follows a normal distribution with mean of 70.5% similarity.*
- $R_2$ *Issue and review length appear to have little to no effect on increasing or decreasing the STS values.*
- $R_3$ *Projects with more issues tend to have a higher mean similarity for their matches.*
- $R_4$ *The variance of the similarities tends to decrease for projects with more contributors or longer issues.*

### *B. $RQ_1$: What is the relationship between the similarity score and the frequency of pertinent matches?*

With a 0.478 Pearson correlation coefficient, we identify a moderate positive correlation between match similarity and the pertinence perceived by the participants. We proceed to analyze how many reviews were considered highly pertinent, with a score of 4 or 5 by both respondents, on different levels of similarity. If we consider intervals of 5% (0.05) similarity, we obtain the values depicted in Figure 5.

Beyond the moderate correlation between the pertinence score and the review similarity, we observe that reviews with high similarity are more frequently considered pertinent by the participants. Specifically, only 9% of reviews with a similarity score between 65% and 70% were highly pertinent. However, the percentage of pertinent matches rises exponentially with higher similarity values, increasing from 20% to 89% between 75% and 90% similarity.

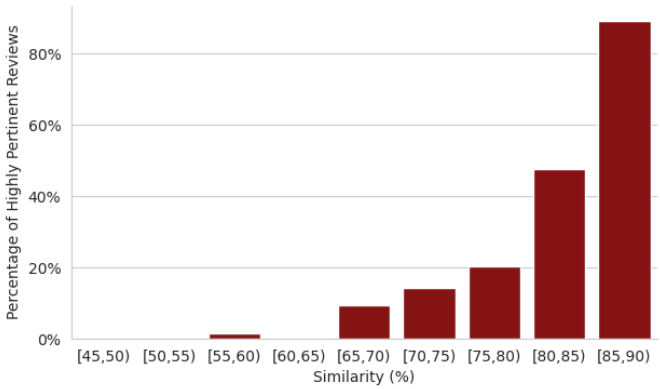


Fig. 5. Percentage of highly pertinent reviews across similarity.

***$RQ_1$: Key Results***

- $R_5$ *There is a moderate (0.478) correlation between the similarity value and the match pertinence scores.*
- $R_6$ *The percentage of pertinent reviews grows significantly with increases in similarity.*

### *C. $RQ_2$: How often are highly pertinent reviews also informative?*

We observe a clear tendency for highly pertinent reviews to be informative more often than slightly pertinent ones, as shown in Figure 6.

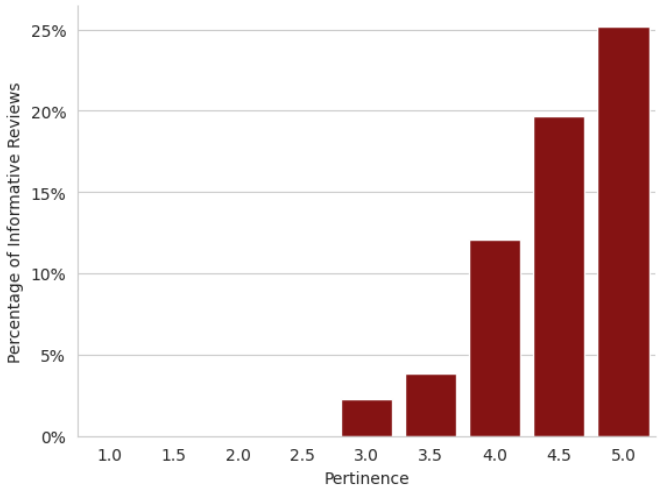


Fig. 6. Percentage of informative reviews across pertinence scores.

Only 2.3% of reviews with a pertinence score of 3 were marked as informative, with respect to the 25% of reviews with the highest pertinence score. The percentage of informative reviews is directly proportional to the pertinence score assigned by the respondents.

A similar phenomenon is observed on the similarity values computed by the approach for informative and non-informative reviews: We found that reviews marked as informative had a higher average similarity than those not marked as informative.

Over 75% of non-informative reviews had less than 75% similarity to the corresponding issue, while 75% of informative reviews had at least 71% similarity to the issue. The short overlap between the two classes suggests the feasibility of using a similarity threshold to differentiate between them.

***RQ$_2$: Key Results***

*$R_7$ Highly pertinent matches are significantly more frequently informative than low pertinence matches.*

*$R_8$ Informative reviews tend to have higher similarity to the issue than non-informative ones.*

## VI. Informative Reviews in Practice

Although our empirical evaluation has demonstrated that our approach can retrieve informative reviews, it has not assessed how these reviews complement the information in the matched issues. To validate whether reviews with informative matches can augment the information present in an issue, we manually reviewed the highest similarity matches for a single project. We chose to conduct this separate case study with a project not included in the DATAR dataset to further test the generalizability of our approach. We chose the Brave browser Android app,[10] a popular FLOSS application with more than 100 million downloads on the Google Play Store.[11]

We collected all the GitHub issues of the app (9,800) and 184,000 reviews written in English, and computed the text embeddings of every review and issue using the `jina-embeddings-v3` model [34]. We observed the matches found for the 100 issues with the highest top-1 similarity. Sorting matches by decreasing similarity, we included the first 100 unique issues matched.

We present two examples with their top-3 scoring matching reviews to show that they are not only pertinent but also informative, as they add meaningful information for the developers.

### A. A Prematurely Closed Bug Report

In Figure 7, we show the matches for issue #29130.[12] This bug report describes the repeating occurrences of crashes after the users tap a search bar.

The top three matches are pertinent to the issue, as they all mention the browser crashing after interacting with search bars. The author of the oldest matching review mentions that the problem also happens for other text input fields, not only search bars. This information is missing from the issue text and discussion, hinting at a potentially latent related bug. Considering that the issue was closed 14 months after it was opened, for being *stale*, the temporal distribution of the matching reviews is interesting.

[10]https://github.com/brave/brave-browser

[11]https://play.google.com/store/apps/details?id=com.brave.browser

[12]https://github.com/brave/brave-browser/issues/29130

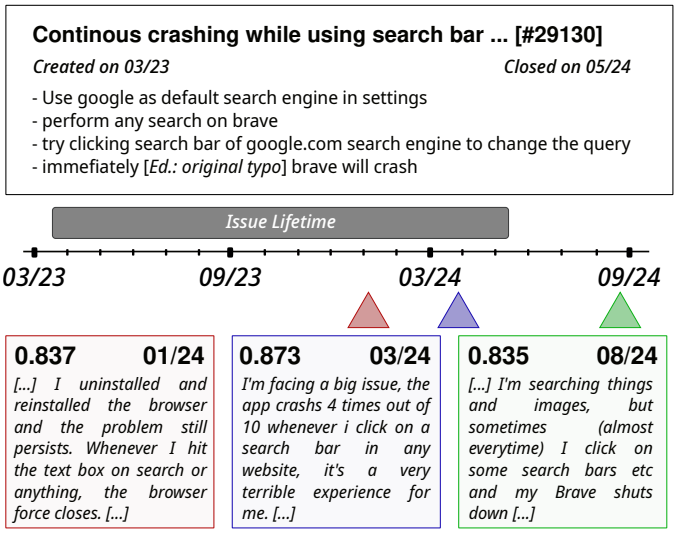


Fig. 7. Issue #29130 (bug report). Lifetime and top-3 relevant reviews.

The first review appears 10 months after the issue was opened, yet the occurrence of crashes for other text input fields is missing in the issue contents and discussion. The second review in order of time, and highest ranking, reports the problem 3 weeks before the issue was closed. The third match found by our approach reports the occurrence of crashes related to search bars 4 months after the issue was closed, also mentioning searching images as causing crashes. The user tried uninstalling and reinstalling the app, thus implying having the most up-to-date version, letting the developers infer that the bug is still present in the current version, at least for some hardware, software, and OS configurations.

### B. A Frequently Requested Feature

In Figure 8, we show the issue #25863.[13] This contains a feature implementation proposal for the option to toggle on or off the behavior of opening links with external apps (actually, a feature already present in Brave for the iOS operating system).

All top-scoring relevant reviews revolve around the expected behavior for opening links in external apps, the core topic of the issue. Each review brings a new insight into how or why the developers should implement the feature.

The oldest matching review proves that the feature has been desired for more than 10 months before the issue was opened. The second one brings a concern about the configurability of the default external apps and a misconfiguration, overriding global phone settings. Finally, the most recent review asks for a possible change in the behavior of the feature after the developers have already implemented it, questioning the default value chosen for the feature. The matched reviews come from different moments in time, all outside the lifetime of the issue, each one providing useful information to inform and affect the development process and decisions.

[13]https://github.com/brave/brave-browser/issues/25863

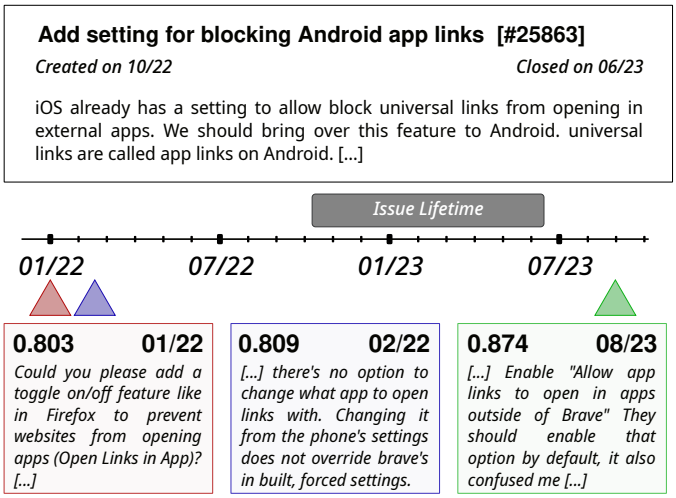


Fig. 8. Issue #25863 (feature request). Lifetime and top-3 relevant reviews.

## VII. Discussion

Overall, our findings seem to confirm the potential and generality of our approach. The results of our quantitative analysis ($R_1, R_2, R_3, R_4$) hint at how the approach results may differ for apps based on their popularity and size.

The normal distribution of the similarity values ($R_1$) provides predictable results of the number of matches filtered when thresholding based on similarity, and allows practical comparisons with other normally distributed variables. The repeating pattern for the similarity distribution of our projects, as seen in Figure 3, illustrates how little difference there is between projects of same popularity.

Although our sample of apps did not identify any correlation between the number of reviews of an app and the mean similarity value for its matches, we presume this correlation could still exist, much like is the case for apps with more issues ($R_3$). Possibly, an even larger sample including apps with hundreds or millions of reviews could help highlighting this tendency.

The increased similarity variance for projects with longer issues ($R_4$), coupled with the absence of a notable correlation between issue or review length and similarity values ($R_2$), indicates that the embeddings from the model used might be sensitive and unstable for long issues. Intricate issue templates or long stack traces seem to impact the reliability of the embeddings produced, consequently affecting the STS values. Future improvements of the approach should tackle this hurdle by removing meaningless or generic information from issue descriptions prior to the embedding process.

Our qualitative analysis, supported by the evaluation of our survey respondents, was essential to answering $RQ_1$ and $RQ_2$. Its positive results consolidate the feasibility of using STS to augment issues with informative reviews. Following the assumptions that based our research questions, $RQ_1$ results ($R_5$ and $R_6$) attest that **the STS based on text embeddings can be used to filter pertinent reviews**.

Comparing the number of suggestions considered related to that found by Haering *et al.* [11], our approach offered related suggestions more frequently. Suggesting three candidate bug reports for each user comment, Haering *et al.*'s DeepMatcher offered 167 related suggestions out of 600 (27.8%). Considering exclusively the first three reviews our approach suggests for each issue, we identified 520 related suggestions out of the 1200 (43.3%) top-three reviews with highest similarity to each issue. While this may be due to the model we chose being more capable, it is fair to note that the previous authors experimented with a smaller sample of issues and reviews, and included only bug reports on their evaluation.

Similarly to previous work, such as [11], our findings support the assumption that STS values tend to be higher for pertinent user reviews. However, our results go one step further by quantifying the relation between higher similarity and higher perceived relatedness (pertinence).

The results for $RQ_2$ ($R_7$) prove these **highly pertinent reviews are more likely to contain informative matches**, thus validating our approach. In particular, the finding that informative matches tend to have higher similarity values ($R_8$) justifies a key aspect of our approach: The addition of a minimum similarity threshold for the matches suggested to any given issue. The existence of such a threshold dramatically reduces the number of low pertinence reviews incorrectly suggested to the developers. This, in turn, will increase the approach precision and improve its helpfulness when integrated into development workflows. At the same time, this increase in precision would allow to extract a subset of issues with reliably pertinent and informative suggestions.

### *Filtering Suggestions By Similarity*

Experimenting with possible values for the minimum similarity threshold alongside the dataset annotated in our survey, we can observe changes in how many informative reviews are left in the approach top-5 matches.

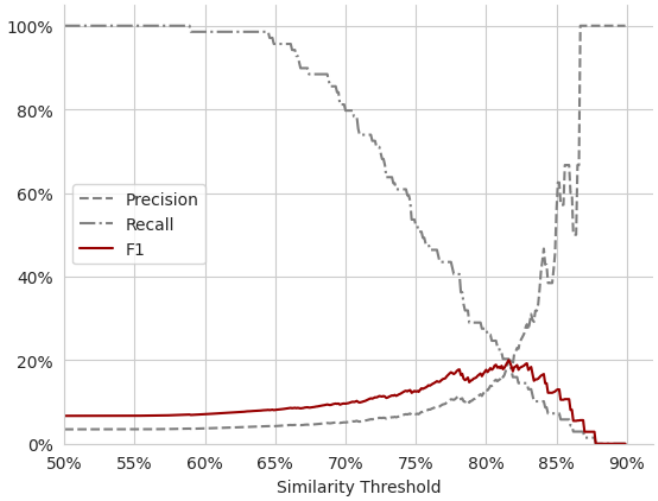


Fig. 9. Precision, recall, and F1 score of varying minimum similarity thresholds on the informative reviews of our annotated dataset.

In Figure 9, we show how different thresholds would impact the approach precision, recall, and F1 score. Choosing a value for the similarity threshold that maximizes the F1 score, we pick 81.6% similarity, which corresponds to a F1 score of 0.2. With our minimum similarity threshold candidate set at 81.6%, we can use it to deepen our quantitative analysis on the full set of 37,000 issues. Out of the over 750,000 reviews, 7,287 (1%) satisfy the threshold, which would pertain to 3,017 (8%) different issues.

Back to the annotated subset of issues, our similarity threshold would filter pertinent matches for 35 of the 400 issues, out of which 13 would include informative matches. In other words, we expect the given similarity threshold to augment over 37% of issues with pertinent matches.

**We argue that augmenting 8% of issues with reviews that have a high potential to be informative is an encouraging result**. Choosing a lower similarity threshold would severely reduce the precision of the approach in identifying informative reviews, requiring further manual analysis by developers and thus defeating its purpose of *automatically* augmenting GitHub issues. The recurring occurrence of internal or excessively technical issue descriptions, coupled with the rarity of useful user reviews, makes augmenting 8% of issues with **pertinent** and potentially **informative** user reviews an achievement that could merit a follow-up study with project developers, which we will consider as future work.

Besides adequately filtering out the noise of thousands of useless user reviews with praise or no actionable information [24], our approach has overcome the language differences between the technical writing of developer issues and users reviews [14], [40]. We hypothesize that this could be a consequence of our choice of a multi-lingual model.

The Brave case study shows that the highest-ranking matches can bring new insights to support the developers' comprehension of the context of an issue. The most pertinent matches to a bug report often include additional information on the context in which the bug manifests itself. Similarly, for new feature requests, the most pertinent matches contain insights on what might better suit the users interests and needs. This case study also highlights the importance of presenting issues and reviews with proper contextual information (*e.g.,* dates on a timeline) to ease and speed up assessing the usefulness and insights provided by the suggested reviews. Ultimately, our evaluation serves as a guide to future applications that might use STS to support software evolution, and supports the assumption that **STS serves as a good basis for filtering highly pertinent reviews and supporting development workflows with user-written information**.

## VIII. Threats to Validity

**Construct Validity.** Our results might differ from the ones of using different LLMs for computing text embeddings. Although our implementation of the approach used the `jina-embeddings-v3` model [34], changing it for other mechanisms could affect the mean, variance, and overall distribution of the similarity values used for our analyses.

**Internal Validity.** The inherent subjectivity of pertinence and informativeness constitutes the main threat to our analyses. Factors such as the varying prior experience on software development and the lack of prior knowledge about the projects analyzed reflect on the number of disagreements of our survey. To mitigate this threat, we measured review pertinence following practices of previous works that also measured text relatedness to decrease the influence of biases and external factors on the pertinence value chosen by the respondents. Our choice of a 6-valued pertinence score follows the interface typically adopted by annotations and experiments that capture human-rated text relatedness [2]. Similarly, our choice of only considering informative the reviews that the respondents agreed on reduces the influence of individual biases.

As another consequence of the subjective measurement of review pertinence and informativeness, our choice of a similarity threshold done in Section VII could be affected by our participants' biases and how we chose to treat their disagreements. To mitigate this threat in practice, the implementation of our approach into a team's development workflow should automatically adjust the minimum threshold following the developers feedback to the suggested matches.

The choice of issue templates adopted by the development teams could also exert a non negligible influence on the STS values. While, overall, longer templates could correspond with more varied STS distributions, factors such as how strictly the maintainers adhere to the template or what fields it contains could influence the similarity values in a currently unexplored manner. Our analysis circumvents this threat by evaluating only broad tendencies from our complete set of projects.

**External Validity** As we have devised our approach as a means to support the development of popular apps, how it could generalize to apps with fewer reviews is out of scope of the current study. However, the choice of restricting ourselves to the DATAR dataset [1] could impact how our approach generalizes to even more popular FLOSS projects.

The restriction of conducting our analysis on a set of projects with openly accessible issue trackers could also have a significant impact on generalizability, due to the fundamentally different usage of issue trackers in closed-source software and industrial environments.

## IX. Conclusion

We presented a novel approach that augments the comprehension of issues by complementing them with information from automatically identified pertinent reviews. Using the embeddings produced by an LLM, our approach identifies pertinent reviews with high semantic textual similarity.

Our quantitative and qualitative analyses demonstrate that similarity-based retrieval can be used to identify highly pertinent reviews that have a high probability of being informative. Coupled with the Brave case study, our evaluation shows the potential of our approach at automatically augmenting GitHub issues with information from user reviews, partially overcoming technical and non-technical language differences.

Future extensions to our approach and evaluation will explore how different embedding models and issue cleaning techniques can impact the similarity values computed. It might be relevant to investigate how the boilerplate used for some types of issues (*e.g.,* bug report templates) could be used to extract the most important sections of each issue body. Moreover, future studies should continue exploring how to effectively present the insights from the user reviews in a clear, automated, and informative manner to the developers, magnifying the issue augmentation potential. For example, the similarity values and dates involved play a significant role in decoding and utilizing the information presented in the matches suggested.

While our results demonstrate the generalizability of the approach for large and popular repositories – starting with those containing tens of thousands of reviews – its straightforward design provides a foundation for other mechanisms aimed at supporting software maintenance. Ultimately, our study sheds light on a promising approach to enriching development workflows with timely and cost-effective insights derived from the collective wisdom of millions of user reviews.

## Replication Package

Our replication package contains the issues, reviews, embeddings, and matches collected and computed by our implementation of the approach for the 19 projects used in our analyses. Additionally, the package contains all survey answers produced for our qualitative analysis, along with the source code used for our analyses. We also share the issues, reviews, and matches we found for the Brave app case study.

The replication package is accessible at:

**https://figshare.com/s/4d65282d72bb47c22ec3**

## Acknowledgments

This research is supported by CAPES (Finance Code 001), the University of São Paulo — USP (22.1.9345.1.2), the São Paulo Research Foundation — FAPESP (2024/00957-8), the São Paulo State Data Analysis System Foundation — SEADE (FAPESP 2023/18026-8), and is part of the EcoSustain project (FAPESP 2023/00811-0). The authors would also like to thank the Swiss National Science Foundation (SNSF) for their financial support through the project "FORCE" (SNF Project Number 232141).

## References

[1] Y. Abedini, M. H. Hajihosseini, and A. Heydarnoori, "DATAR: A dataset for tracking app releases," in *Proceedings of the International Conference on Mining Software Repositories (MSR)*. IEEE, 2024, pp. 384–388.

[2] E. Agirre, D. Cer, M. Diab, A. Gonzalez-Agirre, and W. Guo, "SEM 2013 shared task: Semantic textual similarity," in *Proceedings of the Joint Conference on Lexical and Computational Semantics (SEM)*. ACL, 2013, pp. 32–43.

[3] R. Baeza-Yates and B. Ribeiro-Neto, *Modern information retrieval*. ACM Press New York and Addison-Wesley, 1999.

[4] D. Cer, Y. Yang, S.-y. Kong, N. Hua, N. Limtiaco, R. St. John, N. Constant, M. Guajardo-Cespedes, S. Yuan, C. Tar, Y.-H. Sung, B. Strope, and R. Kurzweil, "Universal sentence encoder," arXiv:1803.11175, 2018. [Online]. Available: https://arxiv.org/abs/1803.11175

[5] N. Chen, J. Lin, S. C. H. Hoi, X. Xiao, and B. Zhang, "AR-Miner: Mining informative reviews for developers from mobile app marketplace," in *Proceedings of the International Conference on Software Engineering (ICSE)*. ACM, 2014, pp. 767–778.

[6] A. Conneau, K. Khandelwal, N. Goyal, V. Chaudhary, G. Wenzek, F. Guzmán, E. Grave, M. Ott, L. Zettlemoyer, and V. Stoyanov, "Unsupervised cross-lingual representation learning at scale," arXiv:1911.02116, 2020. [Online]. Available: https://arxiv.org/abs/1911.02116

[7] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. ACL, 2019, pp. 4171–4186.

[8] A. Fiechter, R. Minelli, C. Nagy, and M. Lanza, "Visualizing GitHub issues," in *Proceedings of the Working Conference on Software Visualization (VISSOFT)*. IEEE, 2021, pp. 155–159.

[9] C. Gao, J. Zeng, M. R. Lyu, and I. King, "Online app review analysis for identifying emerging issues," in *Proceedings of the International Conference on Software Engineering (ICSE)*. ACM, 2018, pp. 48–58.

[10] L. Grammel, H. Schackmann, A. Schröter, C. Treude, and M.-A. Storey, "Attracting the community's many eyes: An exploration of user involvement in issue tracking," in *Proceedings of the Workshop on Human Aspects of Software Engineering (HAoSE)*. ACM, 2010, pp. 1–6.

[11] M. Haering, C. Stanik, and W. Maalej, "Automatically matching bug reports with related app reviews," in *Proceedings of the International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 970–981.

[12] K. Herzig, S. Just, and A. Zeller, "It's not a bug, it's a feature: How misclassification impacts bug prediction," in *Proceedings of the International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 392–401.

[13] P. Hooimeijer and W. Weimer, "Modeling bug report quality," in *Proceedings of the International Conference on Automated Software Engineering (ASE)*. ACM, 2007, pp. 34–43.

[14] L. Hoon, M. A. Rodriguez-García, R. Vasa, R. Valencia-García, and J.-G. Schneider, "App reviews: Breaking the user and developer language barrier," in *Trends and Applications in Software Engineering*. Springer International Publishing, 2016, pp. 223–233.

[15] Q. Jiao and S. Zhang, "A brief survey of word embedding and its recent development," in *Proceedings of the Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*. IEEE, 2021, pp. 1697–1701.

[16] D. Jurafsky and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models*, 3rd ed., 2025, online manuscript released January 12, 2025. [Online]. Available: https://web.stanford.edu/~jurafsky/slp3/

[17] G. Khlifi, I. Jenhani, M. B. Messaoud, and M. W. Mkaouer, "Multi-label classification of mobile application user reviews using neural language models," in *Proceedings of the European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU)*. Springer, 2024, pp. 417–426.

[18] H. Kuramoto, M. Kondo, Y. Kashiwa, Y. Ishimoto, K. Shindo, Y. Kamei, and N. Ubayashi, "Do visual issue reports help developers fix bugs? A preliminary study of using videos and images to report issues on GitHub," in *Proceedings of the International Conference on Program Comprehension (ICPC)*. ACM, 2022, pp. 511–515.

[19] L. Li, Z. Ren, X. Li, W. Zou, and H. Jiang, "How are issue units linked? Empirical study on the linking behavior in GitHub," in *Proceedings of the Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 2018, pp. 386–395.

[20] Z. Li, Y. Yu, T. Wang, Y. Lei, Y. Wang, and H. Wang, "To follow or not to follow: Understanding issue/pull-request templates on GitHub," *IEEE Transactions on Software Engineering*, vol. 49, no. 4, pp. 2530–2544, 2023.

[21] T. Liu, C. Wang, K. Huang, P. Liang, B. Zhang, M. Daneva, and M. van Sinderen, "RoseMatcher: Identifying the impact of user reviews on app updates," *Information and Software Technology*, vol. 161, p. 107261, 2023.

[22] N. Muennighoff, N. Tazi, L. Magne, and N. Reimers, "MTEB: Massive text embedding benchmark," arXiv:2210.07316, 2023. [Online]. Available: https://arxiv.org/abs/2210.07316

[23] M. Nagappan, T. Zimmermann, and C. Bird, "Diversity in software engineering research," in *Proceedings of the Joint Meeting on Foundations of Software Engineering (FSE)*. ACM, 2013, pp. 466–476.

[24] D. Pagano and W. Maalej, "User feedback in the AppStore: An empirical study," in *Proceedings of the International Requirements Engineering Conference (RE)*. IEEE, 2013, pp. 125–134.

[25] F. Palomba, M. Linares-Vásquez, G. Bavota, R. Oliveto, M. Di Penta, D. Poshyvanyk, and A. De Lucia, "User reviews matter! Tracking crowdsourced reviews to support evolution of successful apps," in *Proceedings of the International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2015, pp. 291–300.

[26] S. Panichella, A. Di Sorbo, E. Guzman, C. A. Visaggio, G. Canfora, and H. C. Gall, "How can I improve my app? Classifying user reviews for software maintenance and evolution," in *Proceedings of the International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2015, pp. 281–290.

[27] S. Panichella, A. Di Sorbo, E. Guzman, C. A. Visaggio, G. Canfora, and H. C. Gall, "ARdoc: App reviews development oriented classifier," in *Proceedings of the International Symposium on Foundations of Software Engineering (FSE)*. ACM, 2016, pp. 1023–1027.

[28] I. Saidani, A. Ouni, M. Ahasanuzzaman, S. Hassan, M. W. Mkaouer, and A. E. Hassan, "Tracking bad updates in mobile apps: A search-based approach," *Empirical Software Engineering*, vol. 27(4), no. 81, pp. 1–42, 2022.

[29] C. Sammut and G. I. Webb, Eds., *Encyclopedia of Machine Learning. TF–IDF*. Springer US, 2010, pp. 986–987.

[30] S. Scalabrino, G. Bavota, B. Russo, M. D. Penta, and R. Oliveto, "Listening to the crowd for the release planning of mobile apps," *IEEE Transactions on Software Engineering*, vol. 45, no. 1, pp. 68–86, 2019.

[31] M. Seiler and B. Paech, "Using tags to support feature management across issue tracking systems and version control systems: A research preview," in *Proceedings of the International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ)*. Springer, 2017, pp. 174–180.

[32] Y. Song, J. Mahmud, Y. Zhou, O. Chaparro, K. Moran, A. Marcus, and D. Poshyvanyk, "Toward interactive bug reporting for (Android app) end-users," in *Proceedings of the Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, 2022, pp. 344–356.

[33] C. Stanik, M. Haering, and W. Maalej, "Classifying multilingual user feedback using traditional machine learning and deep learning," in *Proceedings of the International Requirements Engineering Conference Workshops (REW)*. IEEE, 2019, pp. 220–226.

[34] S. Sturua, I. Mohr, M. K. Akram, M. Gunther, B. Wang, M. Krimmel, F. Wang, G. Mastrapas, A. Koukounas, A. Koukounas, N. Wang, and H. Xiao, "jina-embeddings-v3: Multilingual embeddings with task LoRA," arXiv:2409.10173, 2024. [Online]. Available: https://arxiv.org/abs/2409.10173

[35] X. Tang, H. Tian, P. Kong, S. Ezzini, K. Liu, X. Xia, J. Klein, and T. Bissyandé, "App review driven collaborative bug finding," *Empirical Software Engineering*, vol. 29(5), no. 124, pp. 1–32, 2024.

[36] J. Tizard, P. Devine, H. Wang, and K. Blincoe, "A software requirements ecosystem: Linking forum, issue tracker, and FAQs for requirements management," *IEEE Transactions on Software Engineering*, vol. 49, no. 4, pp. 2381–2393, 2023.

[37] S. van Oordt and E. Guzman, "On the role of user feedback in software evolution: A practitioners' perspective," in *Proceedings of the International Requirements Engineering Conference (RE)*. IEEE, 2021, pp. 221–232.

[38] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in Neural Information Processing Systems*, vol. 30, pp. 5998–6008, 2017.

[39] Y. Wang, J. Wang, H. Zhang, X. Ming, L. Shi, and Q. Wang, "Where is your app frustrating users?" in *Proceedings of the International Conference on Software Engineering (ICSE)*. ACM, 2022, p. 2427–2439.

[40] T. Zimmermann, R. Premraj, N. Bettenburg, S. Just, A. Schroter, and C. Weiss, "What makes a good bug report?" *IEEE Transactions on Software Engineering*, vol. 36, no. 5, pp. 618–643, 2010.