# On the Helpfulness of Answering Developer Questions on Discord with Similar Conversations and Posts from the Past

Alexander Lill
University of Zurich
Zurich, Switzerland
lill@ifi.uzh.ch

André N. Meyer
University of Zurich
Zurich, Switzerland
ameyer@ifi.uzh.ch

Thomas Fritz
University of Zurich
Zurich, Switzerland
fritz@ifi.uzh.ch

## ABSTRACT

A big part of software developers' time is spent finding answers to their coding-task-related questions. To answer their questions, developers usually perform web searches, ask questions on Q&A websites, or, more recently, in chat communities. Yet, many of these questions have frequently already been answered in previous chat conversations or other online communities. Automatically identifying and then suggesting these previous answers to the askers could, thus, save time and effort. In an empirical analysis, we first explored the frequency of repeating questions on the Discord chat platform and assessed our approach to identify them automatically. The approach was then evaluated with real-world developers in a field experiment, through which we received 142 ratings on the helpfulness of the suggestions we provided to help answer 277 questions that developers posted in four Discord communities. We further collected qualitative feedback through 53 surveys and 10 follow-up interviews. We found that the suggestions were considered helpful in 40% of the cases, that suggesting Stack Overflow posts is more often considered helpful than past Discord conversations, and that developers have difficulties describing their problems as search queries and, thus, prefer describing them as natural language questions in online communities.

## CCS CONCEPTS

• **Human-centered computing** → **Empirical studies in HCI**; **Field studies**; • **Software and its engineering** → **Collaboration in software development**; • **Information systems** → **Question answering**; **Recommender systems**.

## KEYWORDS

Developer Questions, Chat Community, Semantic Similarity

## 1 INTRODUCTION

Software developers spend a big part of their time at work seeking relevant information, either by searching the web for documentation or tutorials, communicating with their co-workers, or asking questions on Question & Answer (Q&A) communities such as Stack Overflow (e.g., [32]). Software developers also increasingly use chat communities such as Gitter and Discord to ask questions on various topics, ranging from troubleshooting to implementing specific features or even learning how to program [2, 15]. While these chat communities often allow developers to find answers to their questions quickly and collaboratively [56], many questions stay unanswered [27] or are repeated several times by different developers, imposing time and effort on both, the developers asking the questions and the ones answering them.

Automatically identifying previously asked similar questions and suggesting their answers to developers could potentially increase developer productivity. Research on supporting developers in receiving answers to their questions has predominantly focused on Q&A forums, with Stack Overflow (SO) being the most prominent one (e.g., posting links to similar questions [45] or providing summaries of previous questions as answers [65]). Besides leveraging SO as a resource, approaches have also extracted information from documentation such as JavaDocs [40, 60] and chat communities such as Slack and Gitter [22, 56] to suggest them to developers when answering questions on SO. Instead of recommending similar questions, Nascimento Vale et al. used the large language model (LLM) GPT-2 to train and answer questions on SO [26]. To our knowledge, the only approach that aims to support question answering in a chat community is GitterAns [53]. However, GitterAns does not utilize the knowledge from within its own community on Gitter but rather only recommends SO posts to answer troubleshooting questions. In addition, little is known about the helpfulness of the related approaches in the field since most were evaluated in lab settings using test datasets, and only one approach for SO was preliminarily evaluated in a small field setting [45].

In our research, we aim to evaluate the helpfulness of suggesting either previous Stack Overflow posts or Discord conversations to developers in the field. For the Discord suggestions, we thereby utilize the information available in the software development chat communities by suggesting past conversations that are similar to the newly asked question by the developer. In particular, we are examining the following two research questions:

**RQ1** (a) How frequently do developers repeat similar questions on Discord, and (b) how accurately can we detect similar questions using an approach based on SBERT?

**RQ2** (a) Can we support developers in answering their questions on Discord with our SBERT-based approach by providing similar previous Discord conversations and Stack Overflow posts, and (b) which source for suggestions is more helpful?

To address our first research question, we created a dataset from two popular Discord communities and *manually* labeled 465 conversations to examine the frequency of similar questions. Based on SBERT, we then developed an approach that considers the semantic similarity between questions to find similar conversations for a given input question. In an empirical analysis, we assessed the precision of our approach in *automatically* detecting similar questions using the labeled data. Our analysis showed that similar questions frequently occur on Discord, with between 44% and 74% of the analyzed questions being similar to at least one other question. Our approach can identify these similar questions with a precision within the first three results (notation: *precision@3*) of 43% on the labeled dataset and 59% on the overall exported dataset.

To address our second research question, we conducted a field experiment in four Discord communities where we posted links to similar previous Discord conversations and SO posts as an answer to a total of 277 questions. We gathered data via emoticon reactions and textual responses and received additional feedback from our participants through 53 surveys and 10 interviews. Our analysis showed that our approach was able to provide helpful suggestions in 40% of the cases. We further found that while suggestions pointing to SO posts were more often helpful (63%) than suggestions referring to conversations on Discord (37%), the helpfulness for each varies by community. When attempting to find an answer to their question, developers usually first leverage existing information, including documentation websites, Q&A platforms, and, more recently, generative AI solutions such as ChatGPT [6]. Challenges with crafting effective search queries that generalize one's problem and encapsulate the problem context often prohibit finding a suitable answer. In these cases, developers resort to posting their questions to a chat community like Discord, which is often considered more constructive and helpful, allows for clarification questions, and has fast response times. We discuss the helpfulness of our approach and how developers could be better supported when formulating search queries and questions.

This paper makes three main contributions: First, an approach and findings from an empirical evaluation on the accuracy of the approach and the frequency of similar questions based on manually labeled data from two popular Discord communities. Second, a quantitative and qualitative evaluation of the approach in a field experiment with software developers from four Discord communities that provides evidence of the accuracy of our approach in a real-world setting. And third, insights into software developers' current practices of answering their questions using online communities.

## 2 RELATED WORK

Finding answers to coding questions online is very time-consuming, as developers are spending roughly one-third of their time seeking information overall [32], with about 11% of their workdays spent browsing online for information [43]. Answers to these questions can come from an expert in the developer's personal or company network, from documentation, internal or public knowledge bases, or from more generally researching the public web for previous similar questions on the topic [34, 35, 39, 42, 64]. In case a question still remains unanswered after considering one or several of the aforementioned sources, developers often decide to post a new question in either a Q&A forum or an online chat community [66]. Other developers in the community might then self-select to ask clarifying questions, provide suggestions to similar online discussions, and/or provide ready-to-use answers. Q&A forums, and Stack Overflow in particular (e.g., [17, 19, 61]), as well as online chat communities, such as IRC [57], Slack [22], Gitter [27, 47, 55] and Discord [50, 58], have received a fair amount of research that investigated the covered topics, as well as community-specific behaviors and rules.

After Discord was originally mainly adopted by the gaming community, it has recently gained traction among open-source software development communities [13, 15], often after the communities listed Discord as their preferred source of discussion and Q&A. At the time of writing, Discord hosted almost 1300 active Discord communities which, in most cases, either focus on a specific framework or programming language or on learning how to program. The chat-based nature of Discord and similar communities increases the velocity and volatility of the contents of these platforms as content is often less structured, more nested, and more difficult to retrieve at the cost of receiving quick, almost instant, responses to questions [50, 51, 58]. One way to better understand these communities, their topic structure and history, as well as their members is through visualization, as investigated by Raglianti et al. [50]. The work presented in this paper seeks to better understand why and how developers are posting their questions to Discord communities and how they could be better supported in satisfying their information needs in chat-based communities like Discord.

***Supporting Question Asking and Answering in Online Communities.*** One challenge faced by software development online communities is that asking a good question that is detailed and at the same time concise enough to attract voluntary respondents is hard [18, 21]. Therefore, many online communities provide guidelines on asking good questions and employ moderators to enhance quality. Stack Overflow, for example, is actively closing questions that are near or exact duplicates of prior questions, off-topic or "pointless" [18, 23, 48]. To reduce the need to close low-quality questions and provide askers with an increased chance of receiving a response, several approaches tried to support developers during the formulation of questions, for example, by providing recommendations on how to write good questions [21], asking a clarifying question [30], or by providing ad-hoc feedback to the asker when formulating a question, either manually by experts [29] or automatically through a web browser extension [46].

Other approaches that focused on Stack Overflow aim to help askers or respondents more directly by posting links to previous similar questions [45, 53, 60], posting links to the relevant documentation [40], or providing summaries of previous answers on similar questions [59, 65]. A relatively new set of approaches aims to provide automatically generated responses or code snippets that are personalized to the asker's question using large language models (LLMs), such as GPT [5] and BERT [25]. These include bots, either specifically trained to answer how-to questions using Stack

Overflow posts [26] or trained for more general purposes (e.g. Chat-GPT [36]), as well as tools integrated into the developer's IDE (e.g. GitHub Co-Pilot[49] and other programming assistants [54]).

Closest to our work is GitterAns, a bot developed by Romero et al. that detects troubleshooting questions on the chat platform Gitter, queries a search engine to find relevant previous Stack Overflow answers, and posts them to the user in the Gitter chat [53]. A preliminary *manual* evaluation of the answers to 20 randomly sampled questions showed that the bot suggested relevant suggestions for 9 of them. In contrast, our work focuses on Discord as the chat-based community which serves as the source for not only the question but also prior conversations as potentially relevant answers. In addition, our approach is evaluated through an experiment *in the field*, on 277 questions in four different software development Discord communities of varying sizes.

***Detecting Similar Questions in Online Communities.*** A core challenge for many of the aforementioned approaches is the detection of similar or duplicate questions and answers that were previously posted in an online community, given an asker's newly posted question. Approaches to identify relevant *similar* questions range from manual (e.g., [56]) to fully automated, for example, by using the "related questions" feature of Stack Overflow [45] or querying an online community's prior responses using an API [53]. In case a prior question is exactly the same as a new question content-wise, it is considered a *duplicate* [28]. To support large online communities with detecting and closing duplicate questions almost instantly and avoiding inefficiencies stemming from answering the same questions repeatedly, several approaches have been developed to automatically detect duplicates, the best one performing with a recall-rate of around 80% [16, 62, 69].

Many of these tools to detect similar or duplicate questions are based on approaches such as LSTM, RNN, or logistic regression (e.g. [16, 62]) and need to compare a newly created question to all other questions to calculate a relevance score [38]. This results in challenges in providing timely detection. One novel way to overcome these disadvantages and still achieve state-of-the-art performance for detecting duplicates is to avoid these comparisons by calculating sentence embeddings using SBERT [33, 37] to represent the question's semantic meaning as a vector and using cosine similarity to find the most similar questions. SBERT is an extension of BERT using siamese and triplet network structures, allowing to calculate semantically meaningful sentence embeddings and avoiding the need to always feed individual question pairs to the model for calculating a similarity score [52].

Given the promising results of SBERT, this work leverages the same technology to find similar questions. While the above approaches have been evaluated with Stack Overflow data, we have not been able to find work for chat communities like Discord. Additionally, while these initial approaches focused on detecting exact duplicates, we are interested in learning how well the technology works to detect conversations similar to a question and how helpful these conversations are to developers.

## 3 APPROACH

To support developers in finding answers to their questions in a chat community, we developed an approach that identifies past conversations that are semantically similar to a developer's question. Since such semantically similar conversations contain answers to previous questions in the community, they might answer the developer's question or at least provide guidance.

Our approach entails three main steps: extracting and cleaning conversations from the Discord community (3.1), creating sentence embeddings of conversations using pre-trained models to capture semantic meaning (3.2), and determining semantic similarity between conversations and a developer's question based on the embeddings (3.3). To address our research questions, we further created a modified version of the approach that retrieves Stack Overflow posts that are semantically similar to a developer's question (3.4).

### 3.1 Extracting & Cleaning Conversations

***Extracting and Grouping.*** For a given Discord channel, we first extract all messages from the channel using the official Discord API [1] and group them into conversations that consist of the question and one or more answers. Our approach focuses on communities with organized help channels that keep messages of different conversations separate through threaded conversations or a bot assigning new questions to single-user help channels, thus not requiring us to disentangle messages.

***Filtering.*** To ensure that conversations are focused on a question and an answer, we filter out conversations with no answer and more than 200 messages. On average, each of the more than 50,000 conversations exported has 13 messages. A manual inspection of the 170 conversations with more than 200 messages showed that these conversations contained multiple, overlapping question discussions mostly due to malfunctions of the bot assigning help channels, which is why we excluded them. We further filter out conversations for which the question author never replied after an initial answer since an analysis of 100 random conversations showed that authors generally respond to initial answers, e.g., by thanking or following up on the question, unless they abandoned it.

***Cleaning Bot Messages.*** Next, we analyze all messages with the bot flag and categorize them to remove unrelated messages. The categorization is based on heuristics we identified in a manual analysis of all bot messages from one community. Specifically, while bot messages that were requested by a chat participant to either format a code snippet or to link often-used documentation are kept, bot messages related to organizing the help channels (e.g., archival reminders) or the bot administration, and bot messages unrelated to the conversation, are removed (also see replication package [41]).

***Optimizing for Embeddings.*** Since only the first 256 words of a conversation are used for creating sentence embeddings, we apply several heuristics to better capture the semantic meaning of the whole conversation in the 256 words. In particular, we automatically identified and extracted code blocks in the messages using the markdown syntax for code, i.e., one or three backticks. If the code block is shorter than 20 characters, we move it to the start. If it is longer, we move it to the end of the conversation. We identified this heuristic based on our manual inspection of 100 conversations that showed that short inline code snippets are generally used to specify highly relevant classes or method names, while longer code blocks usually contain a lot of noise and less relevant information than the natural description. Additionally, we remove text that contains

no semantic value, such as line breaks, user and channel names, emoticons, and hyperlinks. Finally, we normalize all contractions, such as "it's" to "it is" using the Python `contractions` package.

***Generating Conversation Titles.*** To present a list of the most relevant past conversations to users, we further generate a brief summary of each conversation. For some communities, we are able to use the title developers specified when asking a question. In communities without titles, we generate a summary of the conversation using the language model `t5-base-en-generate-headline` [8], which is optimized for generating headlines based on the Text-To-Text Transfer Transformer (T5) [3]. We chose this model after comparing models on a set of randomly selected conversations.

## 3.2 Creating Sentence Embeddings

To extract semantic meaning from a conversation, we used SentenceBERT (SBERT) to create a vector for each cleaned conversation or the first 256 words of it in case of longer ones. SBERT is an extension of the deep learning transformer network approach BERT that relies on a self-attention mechanism taking into account long-range dependencies and contextual information to better capture the semantic meaning of a sentence. It achieves state-of-the-art performance on sentence classification and sentence-pair regression tasks [52]. For our task of ranking conversations based on their semantic similarity to a developer's question, SBERT improves upon BERT by creating semantically meaningful embeddings of conversations that can be compared using cosine similarity with a simple matrix multiplication at runtime. BERT, on the other hand, requires a one-by-one comparison in which each pair of conversations has to be fed into the model to determine similarity.

Various pre-trained SentenceBERT models are available [12] that differ in the data they were trained on, their fine-tuning, and the number of dimensions they use for the vectors. For our Q&A task, we focused on models pre-trained on 215 million question-answer pairs, including commonly known similarity and duplicate datasets from StackExchange (including Stack Overflow) and Quora, thus, avoiding the need to train the models ourselves. Specifically, we selected the two models `multi-qa-MiniLM-L6-cos-v1` (MiniLM) [9] and `multi-qa-mpnet-base-cos-v1` (MPNet) [10] for their high performance on semantic textual similarity tasks both in terms of similarity metrics and prediction speed, reaching a Massive Text Embedding Benchmark (MTEB) [44] score of 78.90 (MiniLM) and 80.28 (MPNet), compared to ST5-XXL for example, which is slightly more accurate (82.63), but significantly slower [24].

## 3.3 Determining Semantic Similarity

To determine past conversations that are most semantically similar to a developer's question, our approach first cleans the developer's question using the same steps as listed in Section 3.1 before creating a sentence embedding of the question as described in Section 3.2. Using the vector representations of the question and the past conversations, the approach then calculates the semantic similarity based on the cosine similarity of each pair of vectors.

It has been shown that a retrieve & re-rank approach that first retrieves results based on cosine similarity and then re-ranks the top results based on a more computationally expensive cross-encoder approach can boost the relevance of the retrieved similar documents [31]. Therefore we further refined our approach and added a re-ranking step of the top 32 conversations most similar to the input question using the cross-encoder `ms-marco-MiniLM-L-6-v2`, chosen due to its high similarity-score performance and speed [12].

## 3.4 A Version for Stack Overflow

To examine how our approach for retrieving past conversations compares to retrieving similar Stack Overflow posts, we created a modified version of it. For this version, we used the official Stack Exchange archives [11] downloaded in September 2022 and pre-selected specific tags corresponding to the topics discussed in the Discord communities to which we applied our approach. Details about the used tags can be found in the section about the participating communities (5.3) and Table 2. We then removed posts that did not contain any of the pre-selected tags or had no answers. For each remaining post, we used the Python `bs4` package to retrieve the visible text from the HTML code of the post, removed line breaks, and created a concatenated representation of the post containing (i) the title of the post, (ii) the list of the tags assigned to the post, and (iii) the question asked in the post. We then created a sentence embedding for each post (or the first 256 words if longer) and determined semantically similar posts as described in Section 3.3.

## 4 EMPIRICAL ANALYSIS

To examine how frequently developers repeat similar questions on Discord (RQ1a) and how accurately we can retrieve these similar repeated questions (RQ1b), we empirically analyzed two popular software development communities on Discord. In particular, we first curated a dataset of 500 conversations that we manually labeled (4.1). We then analyzed the curated dataset (4.2) and finally applied our approach to evaluate its precision (4.3).

## 4.1 Selected Communities & Collected Data

We selected two communities for our analysis: the Electron community that focuses on integrating web applications into the Electron framework, and the TypeScript community focusing on the use of the TypeScript programming language. We chose these two communities due to their high activity and popularity and their representativeness of the various communities on Discord, which can roughly be grouped into (a) projects and frameworks, (b) programming languages, and (c) others. Specifically, we focused on the channels of these communities that are exclusively used for Q&A.

For each community, we extracted 250 conversations going back in time from our export day, January 24th, 2022. This reflects the total number of asked questions within one to four weeks, depending on the community's activity. On average, each of the 500 conversations consisted of 11 messages. We excluded 35 conversations for being off-topic, not having a topic (e.g., just a greeting), or the content largely not being readable for our approach due to it being only available as an attached image. We further excluded exact duplicates when a user asked the exact same question twice due to not receiving an answer. Overall, the analyzed questions that askers initially posed in a conversation have an average word count of 67 in the Electron and 72 in the TypeScript community.

## 4.2 RQ1a: Counting Similar Questions

*Method.* To answer RQ1a, we grouped questions into higher-level topic areas before comparing questions in each topic area for similarity. For this, one author labeled an initial set of 100 questions and identified topic areas, e.g., "build commands and settings", that were then reviewed and discussed with another author. The first and the third author then labeled a further set of 30 questions. Since an agreement of 93% was achieved, the remaining questions were all labeled by the first author. Next, we analyzed all questions within the same topic area and labeled them with specific topics. In case two questions described the same problem or asked for the same information, and we considered the answers for one question also helpful for the other, we labeled both questions with the same topic.

*Results.* For the 465 analyzed questions, we identified a total of 75 topic areas. 40 topic areas contained more than one question, and the remaining 35 only had one question per topic area. The analysis of the 431 questions that were categorized into the 40 topic areas with multiple questions resulted in 153 individual questions, and 278 questions being part of a total of 45 question topics with at least two questions describing the same problem or asking about the same information. Thus, we found similar questions for 60% (278 of 465) of all cases, where the answer for one of the questions could benefit another question about the same topic. The 45 identified topics contain between 2 and 79 questions (mean=6.2±11.7), with most of them containing two.

The majority of similar questions were found in the Electron community (180 out of a total of 243, 74%), and significantly less in the TypeScript community (98 of 222, 44%). The most common topics in the Electron community were questions on how to communicate with the operating system (79×) and how to control the application window from the web application running inside Electron (10×). In the case of the communication with the operating system from Electron, the common answer is explaining the intricacies of context isolation that prevents web applications from accessing resources outside of their browser window, and how inter-process communication (IPC) works in Electron. For TypeScript, the most common questions asked how to export (20×) and import (17×) modules and how to handle asynchronous operations using promises (11×).

> **RQ1a:** Similar questions are frequently repeated in the Q&A channels of Discord communities. Out of a total of 465 analyzed questions, 278 are similar to each other (60%). The amount of repeated similar questions differs between communities and makes up between 44% and 74% of the analyzed questions.

## 4.3 RQ1b: Detecting Similar Questions

*Method.* To evaluate our approach for detecting and retrieving similar questions, we used the labeled dataset for calculating its *precision@k* and *recall@k* metrics, and conducted an additional manual evaluation to determine the similarity of the retrieved questions.

For the automated analysis, we used the dataset of labeled similar conversations created for RQ1a. We used each of the 278 questions belonging to topics with similar questions as input to our approach and evaluated how many of the questions retrieved by our approach are labeled as similar in our dataset. For this evaluation, we queried

**Table 1: Model Comparison per Community for Automated$_A$ and Manual$_M$ Analysis showing *Precision@3* and *Recall@3***

|             | Electron (E) | | TypeScript (TS) | | E & TS (Avg.) | |
|-------------|------|------|------|------|------|------|
|             | P@3  | R@3  | P@3  | R@3  | P@3  | R@3  |
| MiniLM$_A$  | 49%  | 12%  | 36%  | 11%  | 43%  | 12%  |
| MPNet$_A$   | 47%  | 11%  | 35%  | 12%  | 41%  | 12%  |
| MiniLM$_M$  | 65%  | -    | 53%  | -    | 59%  | -    |

only the labeled dataset for finding similar conversations and not the complete set of exported questions, allowing us to calculate both *precision@k* and *recall@k*. The *precision@k* metric indicates how many of the first $k$ returned results are relevant for the given input, and the *recall@k* metric shows how many of the overall relevant results were returned for a given input in the first $k$. We set k=3 as we are interested in the performance of the first three results, which will be shown to users as part of our experiment (see section 5), with the goal to provide at least one relevant suggestion out of three to limit user effort while also ensuring value. This evaluation was conducted for both pre-trained models selected in subsection 3.2 to compare their performance and determine which one to use for the experiment.

Additionally, we manually analyzed a total of 100 questions to evaluate our approach and the chosen pre-trained model under realistic conditions. For this, we randomly sampled 100 questions from our labeled dataset created for RQ1a, and used our approach to find similar questions within the complete exported dataset. The retrieved conversations were then manually analyzed to determine whether they are similar to the input question.

*Results.* The evaluation results show that our approach achieves an average *precision@3* of 43% and an average *recall@3* of 12% across both communities in the labeled dataset of 278 questions using the MiniLM model (see Table 1). The *precision@3* metric is 49% for the Electron, and 36% for the TypeScript community, where we also found significantly fewer similar questions (44% similar questions for TypeScript, 74% for Electron). While we report the *recall@3* scores for completeness (see Table 1), their meaningfulness is limited since the score is highly affected by a large number of similar questions for some topics and the low k=3. For instance, for the topic of communicating with the OS (i.e., IPC) in Electron, there are 79 similar questions resulting in a best-case *recall@3* score of 4% (3/79) for three out of three relevant results. Experimenting with the retrieve & re-rank approach using the cross-encoder described in subsection 3.3 yielded no performance improvement.

Our manual analysis of 100 questions using the MiniLM model shows that the performance improves when we do not restrict our search to the 465 labeled questions but search for similar questions in the overall dataset of 11,171 questions. We achieved an average *precision@3* of 59% (65% for Electron and 53% for TypeScript) and were able to identify 169 additional similar questions which were not included in the labeled dataset. These results provide evidence that our approach can successfully retrieve one to two similar conversations when given an input question, and thus satisfies our aim of providing at least one relevant suggestion in three.

To determine a threshold for showing the retrieved similar questions to users we analyzed the similarity scores generated by our approach during both the automated and manual evaluation. Starting from the average threshold for the correctly identified similar questions of the automatic evaluation (0.60 for TypeScript, and 0.59 for Electron), we analyzed the scores of the suggested similar questions during the manual evaluation and aimed to lower this threshold to include as many of the relevant suggestions as possible while avoiding too many non-relevant suggestions. The lowest scores for relevant suggestions were 0.43 for TypeScript and 0.32 for Electron. In the end, we defined our minimum threshold for the similarity score as 0.46 after a case-by-case analysis of the suggestions with scores that are lower than the average of 0.60.

> **RQ1b:** Our approach achieves an average *precision@3* of 43% across both communities in the labeled dataset, and a *precision@3* of 59% when tasked to find similar questions in the overall exported dataset of 11,171 questions. This shows that, in case there are similar conversations with relevant answers, our approach is able to recommend one to two such conversations within the top three.

## 5 EXPERIMENT IN THE FIELD

To evaluate our approach and answer research questions RQ2a and RQ2b, we applied it to several Discord communities in the field, which allowed us to receive users' feedback about our suggestions for their real-world questions. In addition, the field experiment allowed us to better understand how developers would want to integrate automated suggestions into their workflows.

### 5.1 Experiment Procedure

During the experiment, we monitored participating Discord communities for newly asked questions, generated suggestions for qualifying questions using our approach (section 3), and posted them to the conversation in case they fulfilled pre-defined criteria. To evaluate the helpfulness of the suggestions, their presentation on Discord, and how these suggestions could best be integrated with developers' workflows, we collected participants' feedback through Discord's "reactions"-feature as well as surveys. A few participants further opted to answer clarifying questions in a follow-up interview. The experiment procedure is visualized in Figure 2 and was approved by our institutional ethics board prior to its execution.

***Monitoring for new Questions.*** During the experiment, we monitored the Q&A channels of participating communities for continuous sessions of 1 to 5 hours during a total of 29 different days over 7 weeks, or a total of 85 hours, and processed all new questions asked during these time-windows.

To avoid evaluating the approach on questions it could not succeed, we only posted suggestions to questions that fulfilled the following criteria: the question (a) is longer than 15 words (not including code), (b) does not only focus on the code itself (e.g., asking to refactor a code snippet, or to fix an error in a code snippet), (c) does not contain attached images that contain error messages or other information necessary for understanding the question (not easily extractable from the image), and (d) should not be too specific to a particular environment and user (e.g., describing a unique problem using only the user's domain-specific terminology without any abstraction of the problem).

***Posting Suggestions.*** Whenever an asker posted a new question, we waited at least two minutes before generating and posting our suggestion to avoid interrupting them in the process of formulating their question and adding necessary context. To generate suggestions for qualifying questions, we applied our approach to the question. The output was suggestions for both, Discord and Stack Overflow, including a link to the suggested conversation or post and a summary of the conversation or the title of the post. In general, we alternated between posting suggestions to Discord and Stack Overflow. In case the similarity scores for any of the generated suggestions for the particular platform were above the threshold of 0.46 (see section 4.3), we posted them to the Discord conversation as a response to the question. In case the scores for the generated suggestions were lower than the threshold for the selected platform, we checked the scores for the other platform and posted these suggestions in case they were above the threshold. In case the scores for both platforms were lower than the threshold, we did *not* post any suggestions. In case the approach yielded less than three suggestions, only these were posted.

***Receiving Helpfulness Ratings.*** As one way to receive feedback on the quality of the posted suggestions, we asked question askers and responders to indicate whether a suggestion is helpful by selecting an emoticon as a "reaction": the "✓" emoticon for helpful suggestions and the "×" emoticon for not helpful suggestions. An example of the posted suggestions and one participant's reaction can be found in Figure 1. In that case, the user indicated that the first suggestion was helpful, while the second and third suggestions were not helpful[1]. Participants were further given the option to opt out of the experiment by selecting the "⊖" emoticon.
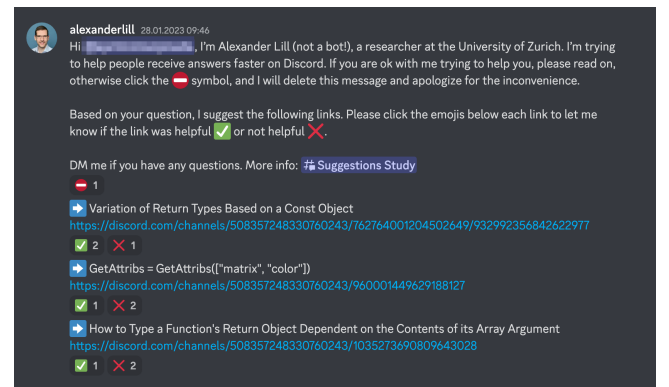


**Figure 1: Example of Posted Suggestions**
The screenshot shows our invitation to participate as well as three suggestions. In this case, a participant rated the first suggestion as helpful, while the second and third were rated as *not* helpful[1].

---

[1]Note that we pre-selected all reactions to allow participants to click the reaction without having to search for a particular emoticon. This means that the count '1' is shown when no reaction is selected. This count is incremented by '1' for every additional user that rates a suggestion.

**Table 2: Participating Communities & Datasets (Averages are rounded to the closest increment of 500)**

| Community | Topics | Avg. Members | Avg. Online | Questions Asked 2022 | Discord Conversations | Stack Overflow Posts | Stack Overflow Filtered Tags |
|---|---|---|---|---|---|---|---|
| Java | Java | 21,000 | 3,500 | 19,004 | 12,884 | 570,260 | java |
| TypeScript | TypeScript | 42,000 | 7,000 | 14,235 | 6,024 | 159,515 | typescript |
| Reactiflux | TypeScript, React, Redux | 210,000 | 14,500 | 8,576 | 2,091 | 354,978 | reactjs, redux |
| Vuetify | TypeScript, Vue.js, Vuetify | 48,500 | 3,000 | 3,294 | 921 | 6,817 | vue.js, vuetify.js |

***Feedback Survey.*** Once participants either reacted with an emoticon or answered with a textual response and did *not* opt out of the study, we invited them to answer a feedback survey. Before participants could start answering the survey questions, they were shown an overview of the experiment and asked to review and agree to the consent form. The survey consisted of four parts: In part 1, participants were asked to give feedback on the suggestions. Part 2 focused on the presentation of suggestions. Part 3 asked how participants approach finding answers to their questions and how they use Discord for the task, and part 4 collected demographics and invited participants to an interview and to enter a raffle for a digital gift card as compensation for participating in the experiment. The survey questions can be found in the replication package [41].

***Follow-up Interview.*** Participants who opted to participate in a follow-up interview were asked clarifying questions about their survey responses, as well as a few additional questions on their preference for receiving Discord versus Stack Overflow suggestions and their requirements for a Discord bot that could post suggestions automatically. The semi-structured interview questions are available in the replication package [41].

### 5.2 Experiment Preparation

***Pilot.*** To test the experiment procedure, we ran a pilot study in the Discord community Astro, a web framework. Overall, we monitored 20 questions and posted suggestions to 13 conversations, for which we received 3 survey responses. In the pilot, we invited participants to our survey via direct messages (DMs) on Discord. Since our messages were mostly ignored due to the lack of notifications for messages from users who are not approved connections, we adjusted the study procedure accordingly by posting invitations to the survey directly into the respective Discord conversations. Since several moderators of Discord communities also stated their interest in supporting the experiment, we adapted our procedure to invite both, question askers and responders, to provide feedback through the survey and interview. Finally, study participants' feedback led us to shorten the length of the study description that we posted to the conversations when asking for feedback.

***Community Eligibility.*** We considered several publicly available lists of software development Discord communities for recruiting (see [41]). To evaluate a community's eligibility the first author joined and checked the following criteria: The community should (a) be concerned with software development, (b) have at least 5,000 members, and more than 1,000 online users to ensure that new questions are regularly asked, (c) have at least 900 questions in the export of their help channels, and (d) should have Stack Overflow

posts concerned with the community's topic(s). To avoid conversations being mixed with different active discussions, (e) communities should either use the "help-forums" feature [4], where each question creates a separate channel, or use a bot to organize channels to avoid overlapping conversations (see section 3.1). For eligible communities, we contacted the moderators through Discord, to explain and discuss the experiment and obtain permission to conduct it.

### 5.3 Experiment Data

***Participating Communities.*** Of all eligible communities, four agreed to participate in the experiment. Two of them, Java and TypeScript, are focused on a programming language, while Reactiflux and Vuetify are concerned with web frameworks. As shown in Table 2, they have between 3,000 and 14,000 online members on average, and users posted a total of 45,109 questions (124 on average per day) to these communities in 2022. For the selected communities, we applied the data preparation steps as described in section 3. Table 2 provides an overview of the exported datasets from both Discord and Stack Overflow, as well as the tags that were used to filter Stack Overflow posts. The export sizes of the two platforms are diverse: while the Java export includes 12,884 exported Discord conversations, Vuetify is much smaller, with 921. Similarly, the Java export from Stack Overflow contains 570,260 posts, while the Vuetify export contains 6,817 posts. Since the export of the Stack Overflow posts tagged with "java" counted more than 1.8M posts, we decided to use only popular questions to both reduce noise and computational complexity and discarded posts with less than 1,000 views. The limit of 1,000 views was selected based on Stack Overflow's definition of a "popular question" [14].

***Participant Demographics.*** We received demographic information from a total of 49 survey participants, which were mostly located in Europe (25×) or North America (15×). Participants indicated that they had completed a high school education (10×) or a bachelor's (17×) or master's (11×) degree. Of the survey participants, 44 identified as male, 2 as female, and 3 chose not to disclose their gender. The majority of survey participants were software developers (35×) or students (6×), with an average age of 27.7 (±6.9). Participants responded having an average of 6.2 years of non-professional (0-29, ±6.1) and 4.2 years of professional software development experience (0-25, ±5.3). On average, they reported having worked for 3.4 years (0-18, ±3.6) with the respective programming language and 1.5 years (0-8, ±1.6) with the used web framework (for web frameworks, both the programming language and framework experience were collected).
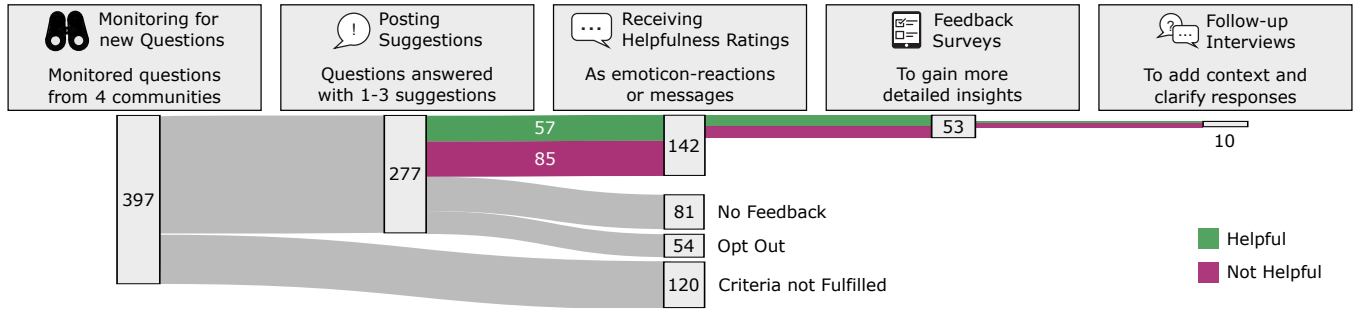
Figure 2: Experiment Procedure & Collected Data

**Collected Data.** Figure 2 visualizes the collected data, starting from the questions that were monitored and answered and ending with the received surveys and conducted interviews. In total, we recorded 397 new questions that were posted to the four Discord communities during the monitoring time windows. Of these, no suggestions were posted to 120 questions that did not fulfill the qualifying criteria defined above: suggestions were below our threshold (42×), or questions that only focused on the code snippet (54×), included key information only as part of an image (33×), were too short (32×), were deleted before or after we posted the suggestions (10×), or asked for an opinion or plan (13×). Of the 277 questions we posted suggestions to, 54 question askers opted out. From the remaining 223 questions, we received feedback in 142 cases and did not receive any feedback in 81 cases, corresponding to a response rate of 64%. Thus, the following analysis is based on the 142 cases that we received feedback for (35.5 ± 7.4 per community). We additionally received survey responses from 53 participants (42 askers A** and 11 responders R**) and conducted 10 follow-up interviews (7 askers and 3 responders).

**Qualitative Analysis.** To analyze the collected qualitative data stemming from surveys and interviews, we applied the reflexive thematic analysis approach by Braun and Clarke [20]. Two authors independently familiarized themselves with the data and coded two-thirds of the open-text survey responses and interviews. To reach an agreement on a final coding tree and saturation, the two authors discussed and consolidated the codes together. Due to the high initial agreement between the authors, the first author then coded the remaining one-third of the data using the previously identified codes, which resulted in no new codes. A subsequent discussion of the codes with all authors led to higher-level themes that are described as results in the following section.

### 5.4 Experiment Results

**Helpfulness of Suggestions.** Of the 142 suggestions that we received feedback for, participants **rated 57 as 'helpful' (40%)** and 85 as 'not helpful' (60%). Participants' feedback was evaluated as one rating for each group of suggestions, meaning that we counted the entire group of suggestions as helpful when one of the up to three suggested links was rated as helpful. In case a participant gave feedback in the conversation and the survey, as well as in case a group of suggestions received feedback from the question asker and responder, they were counted as one feedback.
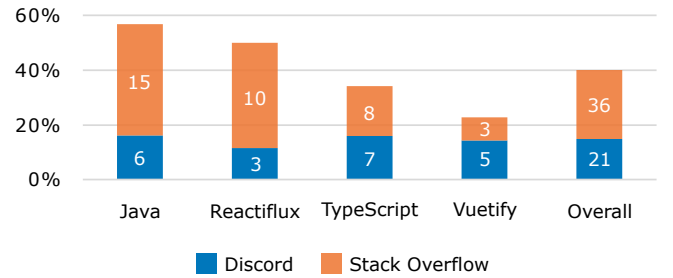


Figure 3: Percentage of Suggestions Rated as Helpful, Distinguished by Suggestion Source and Community

**Helpful Suggestions.** For the 57 suggestions that participants rated as *helpful*, we received additional feedback through 21 surveys and three interviews to better understand participants' ratings on suggestion helpfulness. In 7 of these 21 cases, participants stated that the suggestions *directly* answered their question, meaning they did not need to perform further research. In 14 cases, the suggestions were *indirectly* helpful to participants since they brought them to a *"path where [they could] find a solution to the problem"* (A10), for example, by pointing them towards additionally relevant information or by identifying new keywords for additional web searches that led to a result: *"By providing keywords that I had not thought of when formulating my previous search queries"* (A03). Finally, one participant mentioned that the suggestions led them into a completely new direction that eventually resolved their question: *"It lead to few new topics and old ones too that got overlooked. Ultimately the amalgamation of answers lead me to think in a new direction to reach at a reasonable thought"* (A24).

**Not Helpful Suggestions.** 21 participants who rated the suggestions as *not helpful* provided additional feedback through the survey, and seven of them also in the interviews. The main reasons why participants rated suggestions as not helpful were that they were either unrelated to the specific problem (12×), the problem context was unclear or different (8×, e.g. wrong or not matching software version or environment), or because the suggested resources provided only a partial solution to their question (4×): *"The suggestions scratched the surface and technically answered the question as asked, but didn't address the whole problem."* (A21), *"My question was very*

*nuanced, so the suggestions ended up being too specific."* (A07). Another reason for participants rating a suggestion as not helpful was that they had already found the suggested link before posting their question to Discord, and thus the suggestion did not provide them with any new information (7×): *"I have come across the links that you have shared, and some of those proposed "workarounds" work but don't scale well for many fields with different use cases."* (A31).

> **RQ2a:** We can support developers in answering their questions on Discord by suggesting similar previous Discord conversations and Stack Overflow posts in 40% of the cases.

***Stack Overflow is not always the Better Information Source.***
Participants generally rated suggestions of previous Stack Overflow posts more often as helpful compared to suggestions of previous Discord conversations. From the 57 suggestions that were rated as helpful, 21 (37%) linked to previous Discord conversations, while 36 (63%) linked to previous Stack Overflow posts. As visualized in Figure 3, the helpfulness of each suggestion source differs by community: While suggestions of Discord conversations were more often helpful in the Vuetify community (63% of all helpful suggestions referred to Discord), suggestions linking to Stack Overflow posts were more often helpful in the Java (71%) and Reactiflux (77%) communities, and almost equally helpful in the TypeScript community (53% Stack Overflow and 47% Discord).

In the interviews and surveys, we further asked participants about their preference for these two suggestion sources, yielding 17 responses. Participants (9 of the 17) generally prefer suggestions of Stack Overflow posts since *"Stack Overflow is the main place where you can find answers as a developer"* (A11). More nuanced responses received in the interviews revealed that Stack Overflow questions are often easier to generalize and adapt to one's own problem, while **Discord is preferred for problems that are very specific and require more context**: *"Generally, Stack Overflow has the advantage that it has better responses and that they usually are more generic."* (R02). *"[...] often on Discord, the question is [...] very context specific"* (R05). In addition, participants emphasized that Discord conversations often consist of a more detailed description of the context, helping to better understand and troubleshoot the problem and that it is especially useful to receive these suggestions because this knowledge is inaccessible via search engines: *"Discord threads hold unique value in that [...] the problem could be similar, and you can read through the thought process of the people trying to help and them trying to figure [the problem] out. You have a history to read, which is useful for problems that are hard."* (A18). For 5 of the 17 participants, the source for the suggestions does not matter as long as they help them solve their problem: *"I don't care where the links go as long as they are clean sites (not filled with ads) and have the information I'm seeking."* (A41). Further work is required to study how well these opinions generalize to other developers.

> **RQ2b:** Suggesting Stack Overflow posts was generally more often considered helpful (63%) than Discord conversations (37%). Helpfulness varies between communities and depends on how well the problem can be generalized vs. how much context is required.

***Discord as "Last Resort" for Finding a Solution.*** Of the 35 participants who answered the survey question on the steps they usually take to resolve their problems, **no participant mentioned using Discord first**. Instead, the majority reported first employing a search engine to find existing information (27×), or trying to find a solution using a generative AI such as ChatGPT (8×). The web searches usually directed participants to reading documentation (16×), checking Q&A forums (10×), checking GitHub for related issue reports (5×), sifting through source code (3×), or checking other websites such as blogs (2×). Only after unsuccessfully combing through existing information online participants considered asking a question, either directed at another person (4×) or online on Discord (20×). The main reasons for choosing Discord to ask are the very fast response times (8×) and the ability to communicate back and forth with the responder in case of follow-up questions (3×): *"On Discord, you can have fast, direct talking with people instead of creating a new post and waiting for [a] long time before having someone to answer."* (A01)

Another reason for choosing to post questions to Discord rather than Stack Overflow was mentioned by 13 participants who described **Discord communities to have a very constructive and helpful atmosphere**, while Stack Overflow was described to pose a greater barrier due to the rules, the risk of being down-voted and the need to provide all the required context at the time of posting the initial question: *"Ease of use, no downvote, less trollings, [and] less people trying to get points for their future job."* (A21).

Participants further commented on their strategies when they do not receive answers from Discord or when the answers do not resolve their problem. Out of 27 participants, 12 emphasized that **Discord is usually their last chance for solving their problem** before they move on (6×) or try to find a workaround (7×): *"Discord is kind of my last resort."* (A08), *"If I resort to asking on Discord, it usually means that I've exhausted all my options."* (A18). Other participants stated reformulating their search queries to search again (12×) or posting their unresolved question to another Discord community or a Q&A platform like Stack Overflow or Reddit (6×): *"Asking on Stack Overflow and Quora after that. Should neither of those yield results finding a workaround."* (A03).

***Providing Adequate Context when Searching.*** As previously noted, participants often fall back to posting a new question when they are unable to find a resolution to their problem from existing information. As the main reason for unsuccessful searches, 15 of 25 participants mentioned **difficulties in crafting effective search queries that match their problem's context**. Participants described that they either lacked the necessary knowledge to write effective queries using the correct keywords (6×) or that it was difficult to summarize their problem within a limited and brief query (6×): *"[I am having] difficulty to summarize my problem for a search engine, it is [easier] to ask humans, where I can share code samples."* (A25). As a result, participants reported that their unsuccessful searches led to information that was either unrelated to their problem (4×) or on topic but with inadequate context, such as being too specific, too vague, outdated, or incorrect (11×): *"The most common case that my question was not answered is because the situation is too specific or because the answer is too old, compared to the versions I'm using."* (A09).

***Presenting Suggestions to Users.*** In the survey, participants further elaborated on the ideal form of receiving suggestions, as well as how they should integrate with their existing workflows as askers and responders. In terms of visualizing the information necessary to decide whether to click a suggested link, 21 of the 45 participants who answered the survey question requested a question summary, as well as either a summary of all answers (17×) or the accepted answer in case a suggestion is pointing to a Stack Overflow post (8×). In addition, 14 participants requested to see relevant code snippets from the suggested conversations/posts.

To integrate link suggestions into their workflow, 37 of the 50 answering participants asked for the suggestions to be visible to everyone in the conversation. Not limiting the visibility just to the question asker benefits responders who can then consider these suggestions before providing their answer (7×) and who can better gauge which solutions the question asker might have already considered (4×). It further allows future readers to benefit from the same suggestions to discover solutions to their own problems (27×). In contrast, 10 of the 50 participants stated that they want the suggestions to be visible only to themselves and *before* posting their question, as this could help to reduce the number of similar questions asked (5×): *"I think showing suggestions beforehand can help people quickly solve common issues without having to wait for responses or even submit their questions in the first place."* (A18).

## 6 DISCUSSION

***Helpfulness of Suggestions.*** Based on 142 collected ratings, our results show that participants considered around 40% of the suggestions as helpful. The suggestions thereby either directly answered their question, and thus, reduced the time to find an answer, or pointed them towards additional relevant information. When discussing the helpfulness of our suggestions, moderators hinted that the actual number of helpful suggestions could be higher since participants might have rated questions as "not helpful" in case the answers were difficult to understand or map to the problem: *"The [asker] did not display enough familiarity with coding to understand the problem via indirection. The [suggested] Stack Overflow link only makes sense if you are able to translate the [asker]'s convoluted problem into your own domain, which at the level the [asker] seemed to operate on seemed not very likely."* (R08).

The helpfulness of a suggestion further depends on the particular Discord community (Section 5.4) and dataset size. A Pearson correlation analysis showed a significant (p<0.0005) correlation of 0.95 between the percentage of helpful suggestions and the dataset size (see Table 2). One exception was the Vuetify community, where suggestions of previous Stack Overflow (SO) posts performed worse than expected, given the dataset size. A manual analysis showed that in this community, responses often referred to another major release of Vuetify that was no longer relevant. Previous work showed that outdated code snippets and responses are important issues for developers [63, 68]. While SO is currently evaluating ways to mark outdated solutions to counter the issue [7], this problem is not specific to SO but to all content in online communities. Future work could investigate how communities can better convey and persist context and thus more easily mark questions that are embedded in a different or outdated context.

While related work has suggested several approaches to answer questions on SO by recommending previous SO posts or to improve the way the SO posts are presented, little is known about the helpfulness of these approaches in the field, how they apply to questions in chat communities, and how SO posts compare to previous conversations in the chat communities. Also, while we try to examine a more general-purpose approach, several of the related approaches have been trained and fine-tuned for specific types of questions, such as Java API questions [60] or how-to-questions with code snippets [26], making a comparison of the helpfulness difficult. One of the closest related works is by Murgia et al., which evaluated an approach for answering the 50 most common Git error messages on SO using SO's existing recommendation feature. A preliminary field study resulted in positive feedback for 19 out of 29 (66%) evaluated cases, 5 of which were answers accepted by users and 14 were upvoted [45]. Yet, while the approach by Murgia et al. achieved a higher score of 'helpfulness', it was limited to a specific set of questions, compared to the evaluation in which we applied our approach to the questions that were asked in the Discord communities during the 7-week experiment. Overall, our results indicate that there is potential and value in a more general-purpose approach that recommends prior conversations and/or SO posts to answer questions in chat communities. At the same time, the results show that aspects such as the way the answer is presented (e.g., SO post vs. Discord conversation), the topics discussed in the community, and the details required in the answer affect the helpfulness.

***Providing Question Context.*** While developers usually first rely on existing information to find answers to their questions, many eventually turn to Discord as their last resort for resolving their problems. Discord is preferred for specific problems requiring more context, as it allows fast follow-up questions to provide more context until a problem is resolved. Our results indicate that many developers struggle to formulate their problem as a question and to generalize it appropriately while still providing enough context for responders. These difficulties may discourage them from posting questions to Stack Overflow, as suggested by Ford et al. [29]. Future work could further investigate how to better support developers in abstracting and describing their specific problem as natural language questions while including adequate context for potential responders. In particular, we imagine a system that could initially and continuously support developers in formulating search queries when they try to answer their questions. Besides capturing context from source code inside the IDE (see e.g. [54]), it could capture the developer's search queries, as well as websites and communities that were considered. In case the problem remains unresolved and the developer turns to an online community such as Discord, the automatically captured context could then be suggested as amendments when posing a question, thus supporting the developer with the formulation of their question.

***Using Generative AI for Question Answering.*** Recently, developers started leveraging generative AI tools, such as ChatGPT and GitHub Copilot. These, however, do not fully replace the need to post questions in online communities, such as Discord or Stack

Overflow, nor do they solve all coding problems [36]. The continuous use of Discord, also during our experiment, shows that developers still turn to Discord to find answers to their questions. Several participants even mentioned using ChatGPT first to try and find an answer to their question, yet eventually gave up and asked their question on Discord. One reason why developers are not always successful when using generative AI to answer their questions might again be the difficulty of crafting effective search queries, even in natural language. Prior work on generative AI tools further highlighted that the generated answers and code snippets are often incorrect or do not follow the programming language-specific idioms [36, 49, 67]. Additionally, challenges in the up-to-dateness of the training data and the necessity of re-training generative AI tools might result in unspecific or outdated answers for emerging and fast-changing technologies [36]. A semantic-search approach like the one proposed in this paper, one that only requires indexing new content rather than re-training, could be combined with a generative AI approach to overcome some of these challenges and ensure up-to-date and correct answers.

*Limitations*. The participating Discord communities might not be representative of all software-development-related Discord communities due to eligibility criteria and communities' willingness to participate. The resulting four communities focus on two programming languages and different web frameworks and are of varying sizes of 21,000 to 210,000 members. In addition, our participation criteria explicitly excluded minors. However, messages we received from developers who were willing to participate in the survey but could not due to their age suggest that minors are participating in these conversations. Although the feedback on helpful and not helpful suggestions was equally distributed (see Figure 2), there may be a self-selection bias towards taking the feedback survey only when the question was answered since developers otherwise might have wanted to continue finding a solution instead.

## 7 CONCLUSION

Developers frequently rely on receiving answers to their coding questions by means of chat communities such as Discord, whenever their searches through available online documentation and Q&A forums are unsuccessful. One way to help save time and effort to resolve those questions is to suggest links to previously asked similar questions. To better understand the feasibility of suggesting previous Discord conversations, we conducted an empirical analysis that showed that similar questions occur frequently, with between 44% to 74% of all analyzed questions being similar to at least one other question. We then developed an approach to retrieve similar questions automatically and evaluated it both in isolation, using a manually labeled dataset, and in the field by conducting an experiment. In the experiment, we collected 142 ratings on the helpfulness of suggestions for answering 277 real-world questions posted by developers in four Discord communities. By alternating between suggestions from previous Discord conversations and Stack Overflow posts, we found that Stack Overflow suggestions were generally considered more helpful. Overall, 40% of the suggestions were deemed helpful, and helpfulness varied across communities and dataset sizes. Participants' qualitative feedback also highlighted developers' challenges in finding answers to

their questions from existing information, voicing difficulties with crafting search queries that capture their problem in an abstract way that still encapsulates the necessary context. Future work is needed to better support developers with composing web search queries as well as formulating questions that adequately capture the problem and required context, such as software versions.

## REFERENCES

[1] 2023. Discord API Reference. (March 2023). https://discord.com/developers/docs/reference
[2] 2023. Discord Communities categorized as Educational with keyword Coding. (March 2023). https://discord.com/servers/education?query=coding
[3] 2023. Exploring Transfer Learning with T5. (March 2023). https://ai.googleblog.com/2020/02/exploring-transfer-learning-with-t5.html
[4] 2023. Forum Channels: A Space for Organized Conversations - Discord Blog. (March 2023). https://discord.com/blog/forum-channels-space-for-organized-conversation
[5] 2023. GPT2 GitHub Repository. (March 2023). https://github.com/openai/gpt-2
[6] 2023. Introducing ChatGPT - OpenAI Blog. (March 2023). https://openai.com/blog/chatgpt
[7] 2023. Introducing Outdated Answers project - Stack Overflow Meta. (March 2023). https://meta.stackoverflow.com/questions/405302/introducing-outdated-answers-project
[8] 2023. Model for generating headlines. (March 2023). https://huggingface.co/Michau/t5-base-en-generate-headline
[9] 2023. multi-qa-MiniLM-L6-cos-v1 model. (March 2023). https://huggingface.co/sentence-transformers/multi-qa-MiniLM-L6-cos-v1
[10] 2023. multi-qa-mpnet-base-cos-v1 model. (March 2023). https://huggingface.co/sentence-transformers/multi-qa-mpnet-base-cos-v1
[11] 2023. Official StackExchange Archives. (March 2023). https://archive.org/download/stackexchange
[12] 2023. Pre-trained SentenceBERT models. (March 2023). https://www.sbert.net/docs/pretrained_models.html#sentence-embedding-models
[13] 2023. Publicly maintained list of Discord SE Communities. (March 2023). https://github.com/mhxion/awesome-discord-communities
[14] 2023. Stack Overflow Badges Overview. https://stackoverflow.com/help/badges
[15] 2023. Stack Overflow Developer Survey 2023. https://survey.stackoverflow.co/2023/
[16] Muhammad Ahasanuzzaman, Muhammad Asaduzzaman, Chanchal K. Roy, and Kevin A. Schneider. 2016. Mining duplicate questions in stack overflow. In *Proceedings of the 13th International Conference on Mining Software Repositories (MSR '16)*. Association for Computing Machinery, New York, NY, USA, 402–412. https://doi.org/10.1145/2901739.2901770
[17] Miltiadis Allamanis and Charles Sutton. 2013. Why, when, and what: Analyzing Stack Overflow questions by topic, type, and code. In *2013 10th Working Conference on Mining Software Repositories (MSR)*. 53–56. https://doi.org/10.1109/MSR.2013.6624004
[18] Piyush Arora, Debasis Ganguly, and Gareth J.F.Jones. 2015. The Good, the Bad and their Kins: Identifying Questions with Negative Scores in StackOverflow. In *2012 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*. IEEE, Istanbul, Turkey.
[19] Anton Barua, Stephen W. Thomas, and Ahmed E. Hassan. 2014. What are developers talking about? An analysis of topics and trends in Stack Overflow. *Empirical Software Engineering* 19, 3 (June 2014), 619–654. https://doi.org/10.1007/s10664-012-9231-y
[20] Virginia Braun and Victoria Clarke. 2006. Using Thematic Analysis in Psychology. *Qualitative research in psychology* 3 (01 2006), 77–101.
[21] Fabio Calefato, Filippo Lanubile, and Nicole Novielli. 2018. How to ask for technical help? Evidence-based guidelines for writing questions on Stack Overflow. *Information and Software Technology* 94, C (Feb. 2018), 186–207.
[22] Preetha Chatterjee, Kostadin Damevski, Lori Pollock, Vinay Augustine, and Nicholas A. Kraft. 2019. Exploratory Study of Slack Q&A Chats as a Mining Source for Software Engineering Tools. In *2019 IEEE/ACM 16th International*

*Conference on Mining Software Repositories (MSR)*. IEEE, Montreal, QC, Canada, 490–501. https://doi.org/10.1109/MSR.2019.00075

[23] Denzil Correa and Ashish Sureka. 2014. Chaff from the wheat: characterization and modeling of deleted questions on stack overflow. In *Proceedings of the 23rd international conference on World wide web*. ACM, Seoul Korea, 631–642. https://doi.org/10.1145/2566486.2568036

[24] Peter Devine and Kelly Blincoe. 2022. Unsupervised extreme multi label classification of stack overflow posts. In *2022 IEEE/ACM 1st International Workshop on Natural Language-Based Software Engineering (NLBSE)*. IEEE, 1–8.

[25] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[26] Liliane do Nascimento Vale and Marcelo de Almeida Maia. 2021. Towards a question answering assistant for software development using a transformer-based language model. In *2021 IEEE/ACM Third International Workshop on Bots in Software Engineering (BotSE)*. 39–42. https://doi.org/10.1109/BotSE52550.2021.00016

[27] Osama Ehsan, Safwat Hassan, Mariam El Mezouar, and Ying Zou. 2021. An Empirical Study of Developer Discussions in the Gitter Platform. *ACM Transactions on Software Engineering and Methodology* 30, 1 (Jan. 2021), 1–39. https://doi.org/10.1145/3412378

[28] Mathias Ellmann. 2019. Same-same but different: on understanding duplicates in stack overflow. *Informatik Spektrum* 42, 4 (2019), 266–286.

[29] Denae Ford, Kristina Lustig, Jeremy Banks, and Chris Parnin. 2018. ˮWe Don't Do That Hereˮ: How Collaborative Editing with Mentors Improves Engagement in Social Q&A Communities. https://doi.org/10.1145/3173574.3174182 Pages: 12.

[30] Zhipeng Gao, Xin Xia, David Lo, and John Grundy. 2021. Technical Q&A Site Answer Recommendation via Question Boosting. *ACM Transactions on Software Engineering and Methodology* 30, 1 (Jan. 2021), 1–34. https://doi.org/10.1145/3412845

[31] Gregor Geigle, Jonas Pfeiffer, Nils Reimers, Ivan Vulić, and Iryna Gurevych. 2022. Retrieve fast, rerank smart: Cooperative and joint approaches for improved cross-modal retrieval. *Transactions of the Association for Computational Linguistics* 10 (2022), 503–521.

[32] Márcio Gonçalves, Cleidson Souza, and Victor Gonzalez. 2011. Collaboration, Information Seeking and Communication: An Observational Study of Software Developers' Work Practices. *J. UCS* 17 (Jan. 2011), 1913–1930.

[33] Thi-Thanh Ha, Van-Nha Nguyen, Kiem-Hieu Nguyen, Kim-Anh Nguyen, and Quang-Khoat Than. 2021. Utilizing SBERT For Finding Similar Questions in Community Question Answering. In *2021 13th International Conference on Knowledge and Systems Engineering (KSE)*. 1–6. https://doi.org/10.1109/KSE53942.2021.9648830 ISSN: 2694-4804.

[34] Morten Hertzum and Annelise Pejtersen. 2000. The information-seeking practices of engineers: searching for documents as well as for people. *Information Processing & Management* 36 (Sept. 2000), 761–778. https://doi.org/10.1016/S0306-4573(00)00011-X

[35] Andre Hora. 2021. Googling for Software Development: What Developers Search For and What They Find. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. IEEE, Madrid, Spain, 317–328. https://doi.org/10.1109/MSR52588.2021.00044

[36] Samia Kabir, David N Udo-Imeh, Bonan Kou, and Tianyi Zhang. 2023. Who Answers It Better? An In-Depth Analysis of ChatGPT and Stack Overflow Answers to Software Engineering Questions. *arXiv preprint arXiv:2308.02312* (2023).

[37] Arthur Kamienski, Abram Hindle, and Cor-Paul Bezemer. 2022. Analyzing Techniques for Duplicate Question Detection on Q&A Websites for Game Developers. *Empirical Software Engineering* 28, 1 (Dec. 2022), 17. https://doi.org/10.1007/s10664-022-10256-w

[38] Dehami Deshan Koswatte and Saman Hettiarachchi. 2021. Optimized Duplicate Question Detection in Programming Community Q amp;A Platforms using Semantic Hashing. In *2021 10th International Conference on Information and Automation for Sustainability (ICIAfS)*. 375–380. https://doi.org/10.1109/ICIAfS52090.2021.9606030 ISSN: 2151-1810.

[39] Annie Li, Madeline Endres, and Westley Weimer. 2022. Debugging with Stack Overflow: Web Search Behavior in Novice and Expert Programmers. In *2022 IEEE/ACM 44th International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*. IEEE, Pittsburgh, PA, USA, 69–81. https://doi.org/10.1109/ICSE-SEET55299.2022.9794240

[40] Jing Li, Aixin Sun, and Zhenchang Xing. 2018. Learning to answer programming questions with software documentation through social context embedding. *Information Sciences* 448-449 (June 2018), 36–52. https://doi.org/10.1016/j.ins.2018.03.014

[41] Alexander Lill, André N. Meyer, and Thomas Fritz. 2023. *Replication Package for "On the Helpfulness of Answering Developer Questions on Discord with Similar Conversations and Posts from the Past"*. https://doi.org/10.5281/zenodo.8341017

[42] Jiakun Liu, Sebastian Baltes, Christoph Treude, David Lo, Yun Zhang, and Xin Xia. 2021. Characterizing search activities on stack overflow. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2021)*.

Association for Computing Machinery, New York, NY, USA, 919–931. https://doi.org/10.1145/3468264.3468582

[43] André N Meyer, Laura E Barton, Gail C Murphy, Thomas Zimmermann, and Thomas Fritz. 2017. The work life of developers: Activities, switches and perceived productivity. *IEEE Transactions on Software Engineering* 43, 12 (2017), 1178–1193.

[44] Niklas Muennighoff, Nouamane Tazi, Loïc Magne, and Nils Reimers. 2022. MTEB: Massive Text Embedding Benchmark. http://arxiv.org/abs/2210.07316 arXiv:2210.07316 [cs].

[45] Alessandro Murgia, Daan Janssens, Serge Demeyer, and Bogdan Vasilescu. 2016. Among the Machines: Human-Bot Interaction on Social Q&A Websites. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*. ACM, San Jose California USA, 1272–1279. https://doi.org/10.1145/2851581.2892311

[46] Nicole Novielli, Fabio Calefato, Federico De Laurentiis, Luigi Minervini, and Filippo Lanubile. 2021. A Virtual Mentor to Support Question-Writing on Stack Overflow. In *2021 IEEE/ACM 13th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. 125–126. https://doi.org/10.1109/CHASE52884.2021.00027 ISSN: 2574-1837.

[47] Shengyi Pan, Lingfeng Bao, Xiaoxue Ren, Xin Xia, David Lo, and Shanping Li. 2021. Automating Developer Chat Mining. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 854–866. https://doi.org/10.1109/ASE51524.2021.9678923 ISSN: 2643-1572.

[48] Luca Ponzanelli, Andrea Mocci, Alberto Bacchelli, and Michele Lanza. 2014. Understanding and Classifying the Quality of Technical Forum Questions. In *2014 14th International Conference on Quality Software*. 343–352. https://doi.org/10.1109/QSIC.2014.27 ISSN: 2332-662X.

[49] Rohith Pudari and Neil A Ernst. 2023. From Copilot to Pilot: Towards AI Supported Software Development. *arXiv preprint arXiv:2303.04142* (2023).

[50] Marco Raglianti, Csaba Nagy, Roberto Minelli, and Michele Lanza. 2022. DiscOrDance: Visualizing Software Developers Communities on Discord. In *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 474–478. https://doi.org/10.1109/ICSME55016.2022.00062 ISSN: 2576-3148.

[51] Marco Raglianti, Csaba Nagy, Roberto Minelli, and Michele Lanza. 2022. Using Discord Conversations as Program Comprehension Aid. (2022), 5.

[52] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Association for Computational Linguistics, Hong Kong, China, 3982–3992. https://doi.org/10.18653/v1/D19-1410

[53] Ricardo Romero, Esteban Parra, and Sonia Haiduc. 2020. Experiences Building an Answer Bot for Gitter. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*. Association for Computing Machinery, New York, NY, USA, 66–70. https://doi.org/10.1145/3387940.3391505

[54] Steven I Ross, Fernando Martinez, Stephanie Houde, Michael Muller, and Justin D Weisz. 2023. The Programmer's Assistant: Conversational Interaction with a Large Language Model for Software Development. *arXiv preprint arXiv:2302.07080* (2023).

[55] Lin Shi, Xiao Chen, Ye Yang, Hanzhi Jiang, Ziyou Jiang, Nan Niu, and Qing Wang. 2021. A first look at developers' live chat on Gitter. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, Athens Greece, 391–403. https://doi.org/10.1145/3468264.3468562

[56] Lin Shi, Ziyou Jiang, Ye Yang, Xiao Chen, Yumin Zhang, Fangwen Mu, Hanzhi Jiang, and Qing Wang. 2021. ISPY: Automatic Issue-Solution Pair Extraction from Community Live Chats. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 142–154. https://doi.org/10.1109/ASE51524.2021.9678894 ISSN: 2643-1572.

[57] Emad Shihab, Zhen Ming Jiang, and Ahmed E. Hassan. 2009. Studying the use of developer IRC meetings in open source projects. In *2009 IEEE International Conference on Software Maintenance*. 147–156. https://doi.org/10.1109/ICSM.2009.5306333 ISSN: 1063-6773.

[58] Keerthana Muthu Subash, Lakshmi Prasanna Kumar, Sri Lakshmi Vadlamani, Preetha Chatterjee, and Olga Baysal. 2022. DISCO: A Dataset of Discord Chat Conversations for Software Engineering Research. *MSR 2022* (2022), 5.

[59] Emillie Thiselton and Christoph Treude. 2019. Enhancing Python Compiler Error Messages via Stack. In *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 1–12. https://doi.org/10.1109/ESEM.2019.8870155

[60] Yuan Tian, Ferdian Thung, Abhishek Sharma, and David Lo. 2017. APIBot: question answering bot for API documentation. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2017)*. IEEE Press, Urbana-Champaign, IL, USA, 153–158.

[61] Christoph Treude, Ohad Barzilay, and Margaret-Anne Storey. 2011. How Do Programmers Ask and Answer Questions on the Web?. In *Proceedings of the 33rd International Conference on Software Engineering* (Waikiki, Honolulu, HI, USA) *(ICSE '11)*. Association for Computing Machinery, New York, NY, USA, 804–807. https://doi.org/10.1145/1985793.1985907

[62] Liting Wang, Li Zhang, and Jing Jiang. 2020. Duplicate Question Detection With Deep Learning in Stack Overflow. *IEEE Access* 8 (2020), 25964–25975. https://doi.org/10.1109/ACCESS.2020.2968391 Conference Name: IEEE Access.

[63] Yuhao Wu, Shaowei Wang, Cor-Paul Bezemer, and Katsuro Inoue. 2019. How do developers utilize source code from stack overflow? *Empirical Software Engineering* 24 (2019), 637–673.

[64] Xin Xia, Lingfeng Bao, David Lo, Pavneet Singh Kochhar, Ahmed E. Hassan, and Zhenchang Xing. 2017. What do developers search for on the web? *Empirical Software Engineering* 22, 6 (Dec. 2017), 3149–3185. https://doi.org/10.1007/s10664-017-9514-4

[65] Bowen Xu, Zhenchang Xing, Xin Xia, and David Lo. 2017. AnswerBot: Automated generation of answer summary to developers' technical questions. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, Urbana, IL, 706–716. https://doi.org/10.1109/ASE.2017.8115681

[66] Zhou Yang, Chenyu Wang, Jieke Shi, Thong Hoang, Pavneet Kochhar, Qinghua Lu, Zhenchang Xing, and David Lo. 2023. What Do Users Ask in Open-Source AI Repositories? An Empirical Study of GitHub Issues. *arXiv preprint arXiv:2303.09795* (2023).

[67] Ming-Ho Yee and Arjun Guha. 2023. Do Machine Learning Models Produce TypeScript Types that Type Check? *arXiv preprint arXiv:2302.12163* (2023).

[68] Haoxiang Zhang, Shaowei Wang, Tse-Hsun Chen, Ying Zou, and Ahmed E Hassan. 2019. An empirical study of obsolete answers on stack overflow. *IEEE Transactions on Software Engineering* 47, 4 (2019), 850–862.

[69] Yun Zhang, David Lo, Xin Xia, and Jian-Ling Sun. 2015. Multi-factor duplicate question detection in stack overflow. *Journal of Computer Science and Technology* 30 (2015), 981–997.