# User-Centered Intelligent Information Support for Programmers

**Daye Nam**

CMU-S3D-24-101

May 2024

Software and Societal Systems Department
School of Computer Science
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213

**Thesis Committee:**
Bogdan Vasilescu, Co-Chair
Brad Myers, Co-Chair
Vincent Hellendoorn, Co-Chair
Baishakhi Ray, (Columbia University)
Andrew Macvean, (Google, Inc.)

*Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Software Engineering.*

Copyright © 2024 Daye Nam

This work was supported by the National Science Foundation, under NSF grant CCF-2007482, and by Google. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author and do not necessarily reflect those of the National Science Foundation or Google.

**Keywords:** Information Seeking, Program Understanding, API learning, Developer Tools, Human-Computer Interaction, Software Engineering

*To my loving family,*
*for unwavering support and belief in me.*

# bi` +i

aQ7ir `2 2M;BM22`BM; Bb M BM7Q`K iBQM@BMi2MbBp2 /B
M/ K BMi BMBM; bQ7ir `2 bvbi2Kb- T`Q;` KK2`b 2M+QmMi2`
Q7 [m2biBQMb ` M;BM; 7`QK BKTH2K2Mi iBQM bT2+B}+b iQ
>Qr2p2`- b iBb7vBM; i?2B` BM7Q`K iBQM M22/b Bb MQi 2 bv-

1tT2`BK2Mi H .2VBiMX We chose a within-subjects design, with participants using both GILT (treatment) and a search engine (control) for code understanding, but they did so on different tasks. This allowed us to ask participants to rate both conditions and provide comparative feedback about both treatments.

The control-condition participants were not allowed to use our prototype, but they were free to use any search engine to find the information they needed. The treatment-condition participants were encouraged to primarily use our prototype for information support. However, if they could not find a specific piece of information using our prototype, we allowed them to use search engines to find it. This was to prevent participants from being entirely blocked by the LLM. We expected that this was a realistic use case of any LLM-based tool, but it rarely happened during the study. Only 2 participants ended up using search engines during the treatment condition, but they could not complete the tasks even with the search engines.

We counterbalanced the tasks and the order they were presented to participants to prevent carryover effects, resulting in four groups (2 conditions x 2 orders). We used random block assignments when assigning participants to each group. Participants were assigned to each group to balance the self-reported programming and domain experience (data visualization and 3D rendering). For every new participant, we randomly assigned them to a group that no previous participant with the same experience level had been assigned. If all groups had previous participants with the same experience level, we randomly assigned the participant to one of them.

aim/v S`QiQWQHX We conducted the study via a video conferencing tool and in person, with each session taking about 90 minutes; in-person participants also used the video conferencing tool, for consistency. At the beginning of the study, we asked participants to complete a pre-study survey, collecting their demographic information, background knowledge, and experience with LLMs. We also estimated their general information processing and learning styles using a cognitive style survey [92] categorizing participants into two groups per dimension: comprehensive / selective information processing and process-oriented learning / tinkering. The participants were then asked to join our web-based VS Code IDE hosted on GitHub CodeSpaces [3], which provided a realistic IDE inside a web browser with edit, run, test, and debug capabilities, without requiring participants to install any software locally [58]. We then showed them a demo task description and explained what they would be working on during the real tasks. Before their first task in the treatment condition, we provided a tutorial for our plugin using the demo task, introducing each feature and giving three example prompts for the LLM. For the control condition, we did not provide any demo, as we expected every participant to be able to use search engines fluently. For each task, we gave participants 20 minutes to complete as many sub-tasks as they could. During the task, we did not use the think-aloud protocol because we wanted to collect timing data. Instead, we collected qualitative data in the post-survey with open-ended questions. We also collected extensive event and interaction logs during the task. After each task, we asked participants to complete a post-task survey to measure their understanding of the provided code and the API calls therein. At the very end, we asked them to complete a post-study survey where we asked them to evaluate the perceived usefulness and perceived ease of use of each code understanding approach and each feature in GILT. We based our questionnaire on the Technology Acceptance Model (TAM) [130], NASA Task Load Index (TLX) [95], and pre- and post-study questionnaires that were previously used in similar studies [219, 245].

We conducted 33 studies in total, with 33 participants. The initial 18 studies were conducted on a one-on-one basis, while some studies in the latter half (involving 15 participants) were carried out in group sessions, with two to five participants simultaneously and one researcher serving as

the moderator. We took great care to ensure that participants did not interrupt each other or share their progress. As mentioned before, we excluded one participant's data and used 32 participants' for the analysis. We discovered this issue after the study, as this participant was part of the largest group session (with five participants).

## 7.4 RQ 7.1: Effects of GILT

In this section, we report on the effectiveness of using GILT in understanding unfamiliar code.

### 7.4.1 Data Collection

*Q/2 mM/2`bi MTBMX To evaluate the effectiveness of each condition, we used three measurements: (1) Task completion time: to complete each sub-task; (2) Task progress: we rated the correctness of the participants' solution to each sub-task and measured how many sub-tasks they correctly implemented; and (3) Understanding level: we cross-checked participants' general understanding of the starter code by giving them sets of quiz questions about the APIs included in the starter code. Each set contained three questions, requiring an in-depth understanding of the functionalities of each API call and the application domains. To measure the effect of using GILT and search engines, we excluded the sub-tasks data if participants ;m2bh2/ the solution without using the tool (i.e., zero interaction with the tool) or search engines before completing it (i.e., no search queries).

S`BQ` FMQrHT2/2X To control for prior knowledge, we used self-reported measures of participants' programming and domain experience. We expected more programming experience, especially in the specific domain, to lead to a faster understanding of code.

1tT2`B2M+2 BM A/2p2HQT2`iQQHbX Crafting effective prompts for LLM-based tools requires trial and error, even for NLP experts [59, 280]. Therefore, we asked participants about their experience with LLM-based developer tools. We expected participants' familiarity with other AI tools to affect their usage of the LLM-based information support tool, especially the use of free-form queries, and lead to more effective use of the extension than participants without such experience.

### 7.4.2 Methodology

To answer RQ7.1, we compared the effectiveness of using a GILT with traditional search engines for completing programming tasks by estimating regression models for three outcome variables. For task progress and code understanding, we used quasi-Poisson models because we are modeling count variables, and for the task completion time, we used a linear regression model.

To account for potential confounding factors, we included task experience, programming experience, and LLM knowledge as control variables in our models. Finally, we used a dummy variable (mb2bn) AGILT to indicate the condition (using GILT vs. using search engines). We considered mixed-effects regression but used fixed effects only, since each participant and task appear only once in the two conditions (with and without GILT). For example, for the task completion time response, we estimate the model:

+QKTH2iBQMn/iQBK2BMn2tT2`B2M+2 Y T`Q;` KKBM;n2tT2`B2M+2
Y AniQQHn7 KBHB `Biv Y mb2bn:A

Table 7.2: Summaries of regressions estimating the effect of using the prototype. Each column summarizes the model for a different outcome variable. We report the coefficient estimates with the standard errors in parentheses.

| | Progress (1) | Time(s) (2) | Understd. (3) | Progress Pros | Progress Students |
|---|---|---|---|---|---|
| Constant | 0.41 | 312.65 | 1.81 | 0.38 | 1.82 |
| | (0.49) | (185.33) | (0.89) | (0.68) | (0.83) |
| Domain experience | 0.13 | 23.14 | 0.41 | 0.16 | 0.04 |
| | (0.07) | (25.40) | (0.12) | (0.09) | (0.11) |
| Programming experience | 0.10 | 23.67 | 0.20 | 0.01 | 0.37 |
| | (0.12) | (43.53) | (0.22) | (0.17) | (0.21) |
| AI tool familiarity | 0.01 | 7.70 | 0.09 | 0.07 | 0.10 |
| | (0.07) | (27.04) | (0.14) | (0.11) | (0.10) |
| Uses GILT | 0.47 | 9.10 | 0.29 | 0.57 | 0.29 |
| | (0.16) | (57.26) | (0.28) | (0.22) | (0.25) |
| $R^2$ | 0.173 | 0.022 | 0.202 | 0.341 | 0.137 |
| Adj. $R^2$ | 0.117 | 0.046 | 0.148 | 0.243 | 0.010 |

Note: *p <0.1; **p <0.05; ***p <0.01.

The estimated coefficient for the mb2bn:AuGh variable indicates the effect of using GILT while holding fixed the effects of programming experience, domain experience, and LLM knowledge.

### 7.4.3 Results

Table 7.2 columns (1)-(3) display the regression results for three response variables. The task progress model (Table 7.2-(1)) shows a significant difference between the two conditions, with participants in the GILT condition completing statistically significantly more sub-tasks (0.47 more, $p < 0.01$) than those who used search engines, controlling for experience levels and AI tool familiarity. This indicates that GILT may assist users in making more progress in their tasks compared to search engines.

On the other hand, models (2) and (3) fail to show any significant difference in completion time and code understanding quiz scores between conditions. This suggests that users in the GILT condition do not complete their tasks at a sufficiently different speed or have a sufficiently different level of understanding than those in the control group, given the statistical power of our experiment.

In summary, the results suggest that GILT may help users make more progress in their tasks without changing, for better or worse, their speed and code understanding abilities.

### 7.4.4 Additional Analysis

After observing the significant effect of GILT on task progress, we dove deeper to examine whether all participants benefited equally from the tool. To do this, we divided the participants into two

distinct groups based on their self-reported occupations (professionals and students) and estimated the effects of GILT usage in each group.[kR] We opted for these groups as we did not have any prior theoretical framework to guide our grouping choices, and it provided a simple yet effective approach to group participants with multiple dimensions, including programming experience, skills, and attitude toward programming.

Although both groups were more successful when using the tool, there were notable differences in their performance gains. To better understand these variations, we estimated coefficients for each group (Table 7.2-Pros and -Students) and observed that the impact of GILT was significant only in the Pros group model. Specifically, professionals completed 0.57 more sub-tasks with GILT support compared to when they used search engines, whereas students did not experience significant gains. These findings suggest that the degree of benefit provided by GILT may vary depending on participants' backgrounds or skills.

## 7.5 RQ 7.2: GILT Usage

In this section, we focus on how participants interacted with GILT, their perception of the importance of different features, and how different factors correlate with the feature usage.

### 7.5.1 Usage of Features

To analyze in more detail how participants actually used the tool, we instrumented GILT and recorded participants' event and interaction logs. The logs allowed us to count the number of times participants triggered each feature, and in what order. To supplement the usage data, participants were asked to rate the importance of each feature in a post-task survey. We used these ratings to triangulate our findings from the usage data.

Figure 7.3 summarizes the sequences of GILT features used by participants in the treatment condition. On average, to complete their tasks in this condition, participants interacted with the LLM via GILT 15.34 times. The number of interactions per participant ranged from a minimum of 5 to a maximum of 23. The P p 2 ` p B 2 r feature was the most frequently used method to interact with the LLM, with an average of 4.76 activations per participant. Many participants also used P p 2 ` p B 2 r as their first feature, possibly because it requires minimal effort, with just a single click, in contrast to other features that necessitated the formulation of queries by participants, and perhaps also because some of the buttons (e.g., * Q M + ) 2 T i required first using the P p 2 ` p B 2 r feature. Participants also frequently used S ` Q K T i @ + Q M 2 2 T i (4.12 times) and S ` Q K T i @ 7 Q H H Q r m T (2.88 times). General prompting without code context was used less frequently (1.27 times). While participants generally used buttons less frequently, some used them more frequently than queries (e.g., P29), indicating personal preferences in prompt-based and prompt-less interactions. Specifically, the S A button was used 1.24 times, the * Q M + 2 T i button 0.45 times, and the l b ; 2 button 0.24 times on average.

The reported importance of the features by participants (see Figure 7.4) generally corresponds to the observed usage data. Most of the participants (97%) responded that the ability to directly ask questions to the LLM was extremely/very important, whereas their reported usefulness of the buttons varied. The reported importance of the overview feature (53% extremely/very important)

[kR] We considered but decided against modeling interaction effects as they would have required more statistical power.

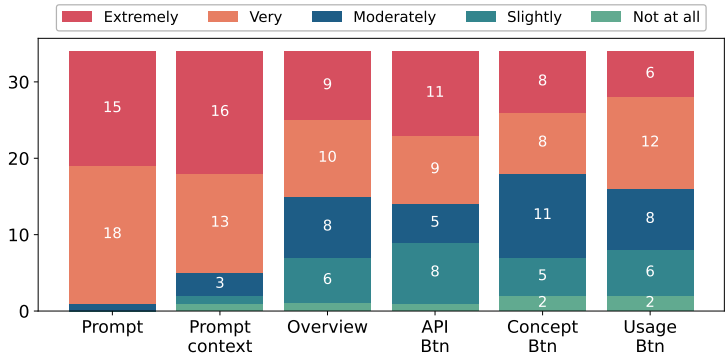
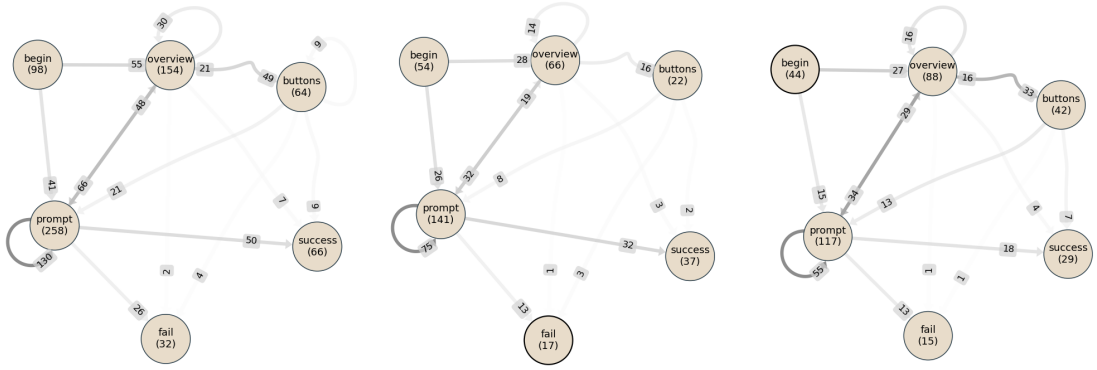



Figure 7.4: Participants' report on the importance of GILT features.




(a) All (b) Professionals (c) Students

Figure 7.5: Transition Graphs for User Interaction. Each node displays the number of times users interacted with respective features, and each edge indicates the counted number of transitions between the connected features. For space and readability reasons, S`QK,Ti S`QKTi@*Q and 2 S`QKTi@6Q are merged into T`QK, and S,A*QM+,2 and Ib ;2 are merged into #miiQM. Counts lower than 5 are omitted except for the edges connected to the 'Success' and 'Fail' nodes.

was relatively low compared to the actual use, suggesting that participants may not have used the summary description provided by the overview but instead used it as context for further prompting or to activate buttons.

To further investigate participants' interaction with the tool, we created transition graphs (Figure 7.5) using sequences of feature-use events for each sub-task, using both the sub-tasks successfully completed by participants and those that resulted in failure (due to incorrect answers or timeouts). Out of the potential total of 128 sub-tasks (32 participants $\times$ 4 sub-tasks), 98 sub-tasks were started before the time ran out. In understanding the transition graph, we focused on the last feature in each participant's sequence, with an assumption that when a participant completes a task, it is likely that the information from the last interaction satisfied their information needs. Among the sub-tasks they successfully completed, a substantial majority (75%) originated from prompt-based interactions. At the same time, 83% of the failed tasks were also preceded by prompt-based interactions, so prompt-based interactions were not particularly likely to result in successful information seeking.

### 7.5.2 Professionals v.s., Students

To better understand the experiences of professionals and students (see Section 7.4.4), we compared the transition graphs for both groups (Figure 7.5 (b) and (c)). Notable distinctions emerged in terms of the features more likely influencing the success and failure of sub-tasks. Specifically, for professionals, a majority (86%) of successful sub-tasks originated from T`QK, whereas for students, this percentage (62%) was statistically significantly lower ( $^2(1, 66), p < .05$). The success rate of prompt-based interaction was also higher among professionals (71%: 32 out of 45) compared to students (58%: 18 out of 31). Conversely, the success rate of the overview and buttons for professionals (56%: 5 out of 9) was lower than that of students (85%: 11 out of 13). These results may indicate that students, possibly with less experience in information seeking for programming, encounter challenges in formulating effective prompts compared to the professionals, and rely more on prompt-less interaction. However, we can also infer that prompt-less interaction is still not sufficient to compete with the benefits of prompt-based interaction with the current design, as they only accounted for less than 40% of the completed tasks.

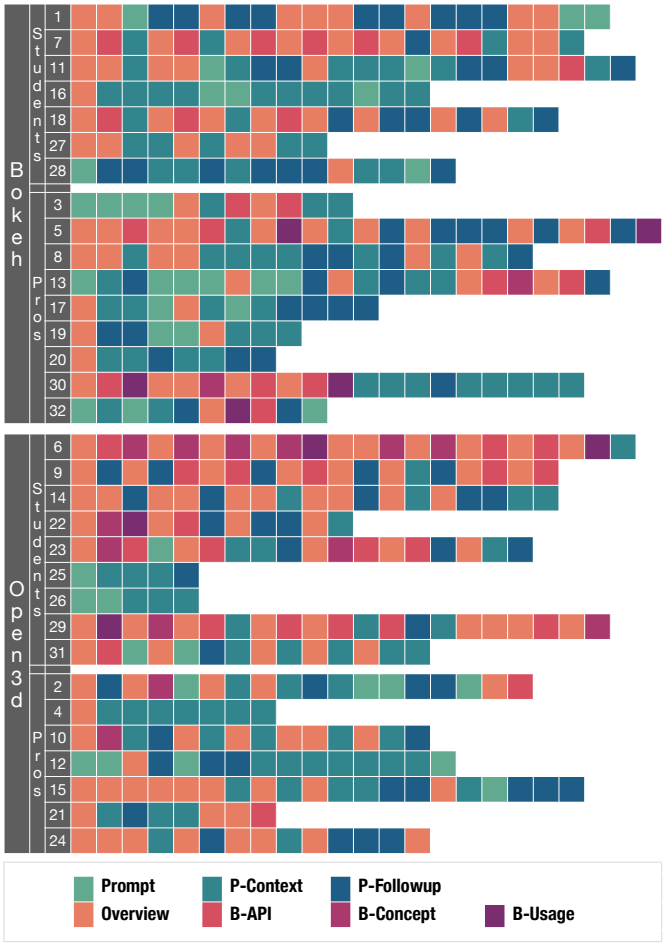



Figure 7.3: The sequences of feature usage in GILT. Each row corresponds to an individual participant, and the color cells are arranged chronologically.

To further investigate the differences in the two groups' prompt engineering, we analyzed the text of the prompts they wrote, by comparing the frequencies of bi-, tri-, and quadrigrams in the prompts. Table 7.3 presents the list of n-grams that showed divergent usage between the two groups. One notable observation is that the n-grams used by the professional group include more effective keywords, or they revise the prompts to incorporate such keywords. For instance, in the bokeh-3 sub-task, none of the participants in the student group used the critical keyword "annular wedge," which is essential for generating the information needed to solve the task, although it was used multiple times in the provided starting code. Instead, students tended to use more general keywords or keywords that had a different concept in the library (e.g., "legend") and faced difficulties in effectively revising the prompts. In addition, more participants in the professional group demonstrated proficiency in refining their prompts by providing further specifications. For example, one participant revised the prompt from "How to change the position of the bunny to 180 degrees" to "How to transform the bunny_mesh to 180 degrees." We infer

Table 7.3: Frequencies of n-grams used differently in prompts by professionals and students. For clarity, we only include n-grams used uniquely by one of the two groups, with a frequency difference of more than 2. If multiple n-grams share the same longer n-gram, we report only the superset.

| Sub-task | n-gram | Pro. | Stu. |
|---|---|---|---|
| bokeh-2 | (‘align’, ‘text’) | 3 | 0 |
| | (‘flip’, ‘label’) | 0 | 3 |
| bokeh-3 | (‘annular’, ‘wedge’) | 6 | 0 |
| | (‘grid’, ‘annular’, ‘wedge’) | 3 | 0 |
| | (‘first’, ‘pie’) | 3 | 0 |
| | (‘pie’, ‘chart’) | 3 | 0 |
| | (‘add’, ‘legend’) | 0 | 4 |
| | (‘tell’, ‘line’, ‘need’, ‘change’) | 0 | 3 |
| o3d-3 | (‘sit’, ‘upright’, ‘chair’) | 4 | 0 |
| | (‘make’, ‘bunny’, ‘sit’, ‘upright’, ‘chair’) | 3 | 0 |

that the difference in the benefit received from GILT by the two groups can be at least partially attributed to their proficiency in prompt engineering.

### 7.5.3 Other Factors Associated with Feature Use

During the pilot studies, we observed that participants approached the tasks differently depending on their familiarity with other LLM-based tools, and styles of information processing and learning as observed in many previous studies on software documentation and debugging [29, 158, 159]. Thus, we tested whether the GILT feature use correlates with factors other than their experience.

> v T Q i ? 2 b 2 b X

Out of two information processing styles [39, 56, 88], people who exhibit a “selective information processor” tendency focus on the first promising option, pursuing it deeply before seeking additional information. On the other hand, people who are “comprehensive information processors” tend to gather information broadly to form a complete understanding of the problem before attempting to solve it. Based on these processing styles, we hypothesized that selective processors would utilize GILT’s S ` Q K T i @ 7 Q H H Q r m J as they would prefer to use a depth-first strategy.

In terms of learning styles [39, 193], “process-oriented learners” prefer tutorials and how-to videos, while “tinkerers” like to experiment with features to develop their own understanding of the software’s inner workings. Consequently, we hypothesized that tinkerers would use GILT less often, as they would prefer to tinker with the source code rather than collect information from the tool.

We also expected that participants who were already familiar with LLM-based tools would use prompt-based interaction in general ( S ` Q K T i ), especially the chat feature, more frequently, as they would already be accustomed to using chat interfaces to interact with LLMs. Conversely,

Table 7.4: Summaries of regressions testing for associations between the user factors and the feature usage counts. Each column summarizes a regression modeling different outcome variables. We report the coefficient estimates with their standard errors in parentheses.

| | Prompt (1) | Followup (2) | All (3) |
|---|---|---|---|
| Constant | 1.39*** | 0.82 | 2.43*** |
| | (0.31) | (0.69) | (0.27) |
| AI tool familiarity | 0.19** | 0.38** | 0.11 |
| | (0.07) | (0.15) | (0.06) |
| Infomation Comprh. | 0.04 | 0.44 | 0.04 |
| | (1.15) | (0.30) | (0.13) |
| Learning Process | 0.19 | 0.60** | 0.12 |
| | (1.14) | (0.29) | (1.13) |
| $R^2$ | 0.262 | 0.283 | 0.165 |
| Adj. $R^2$ | 0.184 | 0.206 | 0.075 |

Note: *p <0.1; **p <0.05; ***p <0.01.

we posited that participants with less experience with such tools would use the buttons more, as prompt engineering might be less familiar to them and place greater cognitive demands on them.

J 2 i ? Q / Q H Q T v X To test for associations between GILT features used and the factors above we again used multiple regression analysis. We estimated three models, each focused on one particular feature. For each model, the dependent variable was the feature usage count, while participants' information processing style, learning style, and familiarity with AI developer tools were modeled as independent variables to explain the variation in usage counts.

_ 2 b m H T b X Table 7.4 presents the results of the regression analysis conducted for three response variables. The first model (Prompt (1)), which uses the total count of prompt-based interactions ( T ` Q K T i @ + Q M T i 2 Q i K T i @ 7 Q ) H H Q r m T , reveals that developers who are more familiar with other AI developer tools are more likely to prompt the LLMs using natural language queries. This result confirms our hypothesis that the AI tool familiarity level influences developers' use of queries. The familiarity level also has a statistically significant impact on T ` Q K T i @ 7 Q H H Q r m T , as shown in the Followup model (2). However, we did not find any significant impact of participants' information processing style on their use of GILT. This means that selective processors and comprehensive processors probed the LLMs similarly, as far as we can tell. The model, however, shows a statistically significant correlation between participants' learning styles and T ` Q K T i @ 7 Q H H Q r m T feature usage. Specifically, process-oriented learners tend to probe LLMs more frequently than tinkerers. This result might indicate that process-oriented learners are more likely to learn thoroughly before proceeding to the next step, while tinkerers tend to tinker with the code after getting the minimum amount of direction from GILT. Finally, the All model (3), which uses the total count of all GILT interactions, indicates that there is no statistically significant difference between the information styles, learning styles, and familiarity levels in terms of overall feature usage counts.

## 7.6 RQ 8.3: User Perceptions

In this section, we investigate how participants perceived their experience of using GILT. Specifically, we examine their perceived usefulness, usability, and cognitive load in comparison to search-based information seeking. Additionally, we explore the pros and cons participants reported, and suggestions for improving the tool.

### 7.6.1 Comparison with Web Search

We employed two wildly-used standard measures, TLX and TAM, in our post-task survey and compared them using two-tailed paired t-tests. TAM (Technology Acceptance Model) [130] is a widely used survey that assesses users' acceptance and adoption of new technologies, and TLX (Task Load Index) [95] is a subjective measure of mental workload that considers several dimensions, including mental, physical, and temporal demand, effort, frustration, and performance. The summaries of TAM and TLX comparisons can be found in Appendix D.

The average scores for the [perceived usefulness, perceived ease of use] in TAM scales were [27.3, 29.75] for the control condition, and [33.49, 34.2] for the treatment condition. The paired t-tests on the TAM scores indicated that there were significant differences in perceived usefulness and perceived usability scores between the two conditions ($p < 0.001$). Specifically, participants rated GILT higher on both dimensions than they did search engines.

For TLX items [mental demand, physical demand, temporal demand, performance, effort, frustration], the average scores were [3.8, -2.1, 4.0, 1.6, 3.4, -0.1] for the control condition and [3.3, -2.5, 2.6, 3.3, 3.3, 1.0] for the treatment condition. Paired t-tests on the TLX scores revealed statistically significant differences between the tool and search engines in temporal demand ($p < 0.05$) and performance ($p < 0.05$) but not in other items. These results indicate that the participants felt less rushed when using GILT than when using search engines, and they felt more successful in accomplishing the task with the tool than with search engines, but there were no significant differences in other dimensions.

### 7.6.2 User Feedback

In the post-task survey, we asked open-ended questions regarding (i) their general experience with using GILT, (ii) the tool's fit with the participants' workflow, (iii) comparison with other tools, and (iv) opportunities for improvement. Two researchers conducted a thematic analysis [50] to analyze the answers. Initially, two researchers separately performed open coding on the same set of 8 responses (25% of the entire data), and convened to discuss and merge the codes into a shared codebook. I coded the rest of the responses and discussed with the rest of the researchers whenever new codes needed to be added. The codebook is available in Appendix D, and we discuss some of them here.

The participants in this study reported several positive aspects of the tool, with the most notable being context incorporation. Participants valued the ability to prompt the LLM with their code as context, which allowed them to tailor the LLM's suggestions to their specific programming context, e.g., "the extension generated code that could easily be used in the context of the task I was performing, without much modification." (P5) Participants also found it extremely useful to prompt the LLM with just code context, as it allowed them to bypass the need to write proficient queries, a well-known challenge in search-based information seeking [116, 270]. P15 mentioned

"A'b MB+2 MQi iQ M22/ iQ FMQr Mvi?BM; #Qmi i?2 +QMi2ti #27Q`2 #2E bi` i2"; vX

Many participants reported that using the tool helped them speed up their information seeking, by reducing the need to forage for information, e.g., "Stack Overflow or a Google search would require more time and effort in order to find the exact issue and hence would be time-consuming." (P27)

Some participants, however, reported having a hard time finding a good prompt that could give them the desired response. Combined with the need for good prompts and the limitations of LLM, this led some participants to report that the responses provided by the tool were occasionally inaccurate, reducing their productivity. P28 summarized this issue well: "[T `QiQ]iv T2 MQi #H2 iQ ;Bp2 K2 i?2 +Q/2 i? i A r b HQQFBM; 7Q?B+b?QABiQ`Q2F`mT H MMQv2/). #AQim?BMF A DimbQ`//Bi?M2 [m2biBQM r2HHX

Participants had mixed opinions on the different features of the tool, especially the buttons. Some preferred to use "/Bz2`2Mi #miiQMb 7Q` /Bz2`2Mi ivT2b Qi7?BpM27iQ`K iBQM `2/ HQi Q7 i2ti iQ }M/ r? i A" (P7), while others thought that was overkill and mentioned " bBKTH2` pB2r rQ" (P8) #2 MB+2X

Compared to ChatGPT, 17 participants (out of 19 who answered) mentioned advantages of GILT, with the S`QKTi@ +QMi2ti feature being one of the main ones. Participants expressed positive feelings about CoPilot but acknowledged that the tool had a different role than CoPilot and that they would be complementary to each other, e.g.: "*QTBHQi Bb iQQH i? i A + M +QK K2+? MB+ H rQ`Fb [mABG+hFQ`v2-`b mBiMbB;?i BMiQ KQ`2" (P29) H H2M;BM; i bFbX

Many participants reported that the tool would be even more useful when combined with search engines, API documentation, or CoPilot,[kk] as they provide different types of information than the tool. Having the ability to choose sources based on their needs would enhance their productivity by giving them control over the trade-offs, such as speed, correctness, and adaptability of the information.

## 7.7 Threats to Validity

One potential concern with our study design is the task and library selection. We only used tasks that show visible outputs, which might have led participants to detect potential errors more easily, compared to other tasks, such as optimization or parallel programming. However, we believe that the tasks we chose are representative of common programming errors that would need to be identified in real-world programming situations. Indeed, when we asked the participants in the post-task survey, both data visualization and 3D rendering tasks were reported to very or extremely closely resemble real-world tasks by 82% and 73% of the participants.

Similarly, the selection of libraries might have biased the study results. However, in selecting libraries for our study, we avoided using popular libraries that could unintentionally give an advantage to LLMs. We believe that the libraries we chose are of medium size and quality, and therefore represent a fair test of the LLM tools. However, it is possible that different libraries or larger codebases could produce different results.

Despite our efforts to create a controlled experience, several factors differentiate our in-IDE extension from search engines, aside from the inclusion of LLMs. For example, although previous

[kk] Notably, GitHub independently announced these enhancements to Copilot already, after we conducted our study: ?iiTb,ffr:ii?2p2`;+2QKfkykjfdfkyfkj3yR9N3f;Bi?m#@+QTBHQi@t@+? i@+ TM#HB+@#2i

research investigating the incorporation of search into IDE did not find a statistically significant difference between the control and treatment groups [35, 134], the in-IDE design itself may have been more helpful than access to LLMs, as it potentially reduced context-switching. Thus, further studies are needed to gain a better understanding of the extent to which each benefit of our prototype can be attributed to these differences.

Additionally, the laboratory setting may not fully capture the complexity of real-world programming tasks, which could impact the generalizability of our findings. Also, the time pressure participants could have felt, and the novelty effect in a lab setting could have changed how users interact with LLMs. Our sample size, 32, was relatively small and skewed towards those in academia. This may also limit the generalizability of our findings to more professional programmers. Thus, future research with larger, more diverse samples is necessary to confirm and expand upon our results.

Our analysis also has the standard threats to statistical conclusion validity affecting regression models. Overall, we took several steps to increase the robustness of our estimated regression results. First, we removed outliers from the top 1% most extreme values. Second, we checked for multicollinearity using the Variation Influence Factor (VIF) and confirmed that all variables we used had VIF lower than 2.5 following Johnston et al. [111].

Another potential threat to the validity of our findings is the rapid pace of technological development in the field of LLM tools. Despite our efforts to use the most up-to-date LLM available at the time of the submission, it is possible that new breakthroughs could render our findings obsolete before long.

## 7.8 Discussion and Implications

*QKT`2?2MbBQM QmiOQm`+BMX* Our analysis revealed an intriguing finding regarding participants' behavior during the study, where some of them deferred their need for code comprehension to the LLM, which was well described by one participant as +QKT`2?2MbBQM. Qmi bQm`+BM; These participants prompted the model at a higher level directly and did not read and fully comprehend the code before making changes. As one participant commented, "Ar bbm`T`Bb2/ #v ?Qr HBiiH ?/iQ FMQr (QQ2i2M i22/bi `i2` +Q/2 #27Q`2 A+M DmKT'BM This behavior might be attributed to developers' inclination to focus on task completion rather than comprehending the software, as reported in the literature [151]. Or, participants may have also weighed the costs and risks of comprehending code themselves, and chosen to defer their comprehension efforts to the language model. While this behavior was observed in the controlled setting of a lab study and may not fully reflect how developers approach code comprehension in their daily work, it does raise concerns about the potential impact of such a trend (or over-reliance on LLMs [261]) on code quality. This highlights the importance of preventing developers who tend to defer their comprehension efforts to the LLM from being steered in directions that neither they nor the LLM are adequately equipped to handle. Studies showing developers' heavy reliance on Stack Overflow, despite its known limitations in accuracy and currency [269, 281], further emphasize the need for caution before widely adopting LLM-based tools in code development. Research on developers' motivations and reasons for code comprehension when LLMs are available will be valuable in informing future tool designs.

*lBHBx2 KQ`2 +QMi2tX* One of the main advantages of GILT reported by the participants is its ability to prompt the LLM with the code being edited as context. We believe that additional

types of context can be leveraged to improve the tool's utility, including project context (e.g., project scale and domain), system context (e.g., programming languages and target deployment environments), and personal context (e.g., programming expertise in libraries, and domains). By combining these contexts with proper back-end engineering, we believe that GILT, or other LLM-powered developer tools, will be able to provide relevant information to developers with even less prompt engineering efforts of the users.

L 2 2 / 7 m `i ? 2 ` b i m / B 2 b B M ` 2 H @ One possible explanation for some of the models with null results from RQ7.1 and RQ7.2 is the artificial setting of the lab study, where participants were encouraged to focus on small, specific task requirements instead of exploring the broader information dimension. For example, participants prioritized completing more tasks rather than fully understanding the code, as reported by participant P18 in their survey response: " [ : A G h ..., r ? B + ? + Q m H / / 2 } M B i 2 H v ? 2 H T Q M 2 i Q i + F H 2 m ? M 2 / 2 b i F ? 2 2 B i 2 2 B @ b 2 2 i ` B 2 M 2 b Thus, although our first study shed some light on the potential challenges and promises, to fully understand the implications of deploying this tool into general developer pipelines, it is necessary to observe how programmers use it in real-world settings with larger-scale software systems, less specific goals, and over a longer time frame. Given that GitHub recently launched CopilotX [83], a tool that offers a comparable set of features to our prototype to enhance developer experience, such research is urgently needed. We believe that our findings are a timely contribution and a good first step for researchers and tool builders in designing and developing developer assistants that effectively use LLMs.

## 7.9 Summary

The user study results have demonstrated that GILT significantly enhances developers' ability to complete tasks compared to traditional information-seeking methods. Participants also expressed positive experiences while using GILT, especially on the ability to incorporate their task context.

# Chapter 8

# Towards Building Personalized Information Support

In evaluating user-centered information support tools, we have demonstrated that user-centered design can meaningfully improve programmers' information seeking. Specifically, the qualitative analysis of user responses in Chapter 7 has shown that incorporating the code context improves their information-seeking. In this final chapter, we test the potential value of incorporating an even broader user context, and eventually T2`bQMHBx2/ generation-based information support by automatically augmenting the user context.

## 8.1 Introduction

Developing a software system requires developers to consider numerous dimensions, ranging from its architecture to implementation bugs. Each dimension demands information of varying scopes and abstractions, making software engineering highly information-intensive. Thus, developers spend 35% of their time on the foraging itself [117], and 50% when other aspects of foraging are taken into account [197]. This intensive search for information is driven by the need to evaluate multiple viable solutions, taking into account their constraints and the trade-offs involved. Inadequate or missing information can lead to suboptimal design decisions, adversely affecting the quality of the software product. Therefore, timely access to all necessary information is crucial for enhancing developers' productivity and, consequently, the quality of the final software products.

However, information seeking is still not easy in software engineering [14, 205, 270]. Much information about software and systems is not properly documented [255] or it is spread across varying mediums in different formats, often getting outdated as software evolves [55]. Developers have used search engines as a major way of information seeking (e.g., they issue more than 20 search queries every day [270]), which many developers find challenging because they may not know, or may forget to include, the right keywords [65, 116, 147]. Even when they retrieve a promising page, they still need to evaluate the relevance of the information [80, 116], which is itself challenging, especially to those who are new to the API, or programming in general [116].

The emergence of generative models and large language models (LLMs) has introduced new avenues for satisfying programmers' information needs, offering some advantages over traditional methods [138, 176, 219]. However, the effectiveness of these models is still contingent upon the ability to write effect prompts, a task that often requires trial and error, even for NLP

experts [59, 280]. While LLMs can process complex queries, allowing users to provide rich clues for search, as users often receive a response from LLMs instead of having the opportunity to browse through multiple potential solutions as in conventional information retrieval approaches, it poses a challenge for discovering suitable solutions when queries fail to direct the LLM effectively.

In this chapter, we posit that considering developer + Q M i2t can alleviate the challenges associated with software information seeking, as it can explain some of the programmers' information needs, even when not explicitly expressed in their queries. Context is defined as "everything that affects a computation except the explicit input and output" [139], or, more specifically for the information retrieval system, as "everything that a perfect information retrieval system would need to know about the user, their situation, the domain, and anything else in order to return exactly the results relevant to the user' s stated query" [67]. Leveraging context has proven beneficial in other domains, such as search engines and assistance systems, which have successfully utilized spatiotemporal context and interaction history to provide personalized results [48, 67, 82]. Although mostly imagined to operate without considering the software development context, thus serving as a "one size fits all" solution, there is already a vision in our community that contextual search would be useful in supporting developers' information seeking [13, 215].

We argue that the time is ripe to begin developing T 2 ` b Q M information support. The personalization of information support for programming might have not seemed to be feasible, as most of the general information seeking was done at search engines, which can only retrieve information from existing documents, providing a limited level of personalization. However, since LLMs generate information, they can further personalize the information support, by suggesting or skipping information at a granularity finer than a document level, which retrieval mechanisms like search engines are limited to, and LLMs can even format the information as the programmer would want it. Furthermore, LLMs can accept a much richer request than a single search query that programmers often use for information retrieval with search engines, enabling the user to provide even broader contextual information.

As an initial step towards providing personalized information support, this chapter focuses on identifying the categories of context that may enhance information support and testing the feasibility of measuring the impact of including such contextual factors on developers' productivity, especially when working with unfamiliar code. Through the examination of ChatGPT prompts in software engineering artifacts and an experimental ablation study, this research aims to address the following questions:

- RQ8.1: What are the contextual factors programmers use in information seeking?
- RQ8.2: Does providing contextual factors enhance the quality of generated information?

## 8.2 RQ8.1: Understanding Context for Information Support

We conducted an empirical study to understand what type of contextual factors can be useful in satisfying the information needs of programmers in the context of LLM-based information support.

### 8.2.1 Study Design

We analyzed the DevGPT dataset [271]. DevGPT comprises mentions of ChatGPT conversations in software development artifacts, including GitHub commits, issues, pull requests, discussions, files, and Hacker News. The dataset [kj] contains 4,733 shared ChatGPT links, a total of 29,778 prompt/answer-pairs, sourced from 3,559 GitHub or Hacker News references.

First, as we have little prior understanding of how programmers express their information needs to LLM-powered generation-based information support tools, we performed open coding. For the initial analysis, we randomly selected 10 conversations per source type (e.g., GitHub issues), resulting in 60 conversations. We excluded the ChatGPT conversations that contained dead links (i.e., 404), or the ones containing prompts written in languages other than English. We also excluded the conversations that are not relevant to Software Engineering tasks, such as "what's the real Netflix idea origin story?" The open coding was conducted at a phrase level, where a phrase can be a sentence (e.g., "What is immutability?", or a part of a sentence (...“expert python programmers”...).

The open coding process yielded 39 distinct codes, encompassing various information needs, relevant contextual information provided, and some guidelines for directing LLMs. A significant observation from the initial coding revealed that the types of contextual information programmers supply are very similar to what they have provided as part of their Stack Overflow questions [146]. Specifically, we observed that the information needs we coded overlap with the seven-question types from Beyer et al. [31]: API usage, Conceptual, Discrepancy, Errors, Review, API change, and Learning. Beyer et al. [31] studied and merged the taxonomies of prior studies [16, 252] on information needs and created a new taxonomy of information needs shown in Stack Overflow questions. As the taxonomy is broad enough to cover the general information seeking in software engineering, although the focus was API-related questions, we found that it can explain more general information needs expressed in the DevGPT dataset.

The initial coding also revealed some overlap with the varied contextual factors influencing developers' information seeking behavior in conventional setting, studied by Freund [79]: Personal factors, Project factors, Work task factors, and Information task factors. The list was created before the advent of LLM-powered information seeking, from a semi-structured interview study conducted between 2004 and 2005, where programmers relied heavily on conventional information seeking, using search engines, documentation, and communication with colleagues. Although numerous studies have explored programmers' information foraging strategies and how these strategies vary with different programmer backgrounds [126, 151]), the differences were mostly discussed as implications from studies on documentation or program comprehension, providing more in-depth, but not extensive lists of contextual factors influencing programmers' information-seeking behaviors.

[kj] 20231012 snapshot

### 8.2.2 Results: Contextual Factors Used by Programmers

We conducted a second round of coding, using the two taxonomies discussed above (information needs, and contextual factors). The closed-coding round resulted in 6 codes for information needs (Table 8.1), and 5 codes for contextual factors ( Table 8.2). The codes from the existing two taxonomies could cover multiple codes, reducing the number of codes, from 39 to 11. For example, both phrases coded as "context: repo" and "context: environment" in the previous open coding fell under the "Project Factors" in this round of coding.

Table 8.1: Descriptions and examples of information needs types.

| *Q/2 | .2b+`BTiBQM _2T`2b2Mi iBp2 ZmQi2 |
|---|---|
| AKTH2K2@Mi iBQM | Needs information to implement specific functionality<br>">Qr + M A mb2 +27 iQ K F2 *?`QK2 /2piQQHb" QT2M QM b2H2 |
| *QM+2TimH | Needs information about conceptual knowledge, such as design patterns or architectural styles<br>"q? i Bb BKKmi" #HBiv\ |
| .Bb+`2T M+v | Needs information to understand and resolve unexpected behavior<br>"q?v Bb Kv `2/B`2+i M?Qi rQ`FBM;\ |
| 1``Q` | Needs information to resolve exceptions and errors<br>"i? i T`Q/m+2b i?Bb 2``Q`- ?Qr /Q r2 }t Bi\ |
| _2pB2r | Needs information to find for better solutions, best practice approaches, or verification of the program.<br>"Ab i?2`2 Mvi?BM; vQm bm;;2b" b M Hi2`M iBp2\ |
| G2 `MBM; | Needs resources to learn a tool, language, or concept.<br>"*QmH/ vQm ;2M2` i2 /2b+`BTiBQM Q7 ?Qr Bi rQ`Fb i? i F<br>mM/2`b" M/X |

The information needs articulated in the prompts were closely aligned with the taxonomy of Beyer et al. [31]. Our analysis of the sampled ChatGPT prompts revealed the presence of six out of the seven categories in the original taxonomy.

Many prompts contained phrases such as how to implement something or ways of using something, initially suggesting an alignment with the "API usage" category. However, to more accurately encapsulate the broader scope of information needs beyond just API knowledge expressed in these how-to inquiries, we revised this category to "Implementation". This adjustment better reflects the varied implementation-related questions found in the prompts. We observed an absence of prompts explicitly seeking information on "API changes", indicating a potential area of lesser concern or different information-seeking behavior among ChatGPT users.

Out of the four categories identified by Freund [79], "Personal factors", "Project factors", "Work task factors", and "Information task factors", "Information task factors" were excluded in our final codebook as they cover the same dimension as the information needs, which was covered in our previous codebook. "Work task factors" was further divided into "Task Requirement", "Status Description", and "Software Artifact" as the task factors were specified in detail in the majority of the prompts. In the end, contextual information in the prompts resulted in five codes:

Table 8.2: Descriptions and examples quotes of contextual factors

| *Q/2 | .2b+`BTiBQM 1t KTH2 |
|---|---|
| h bF _2[mB`2K2MA | Descriptions of the functional and non-functional requirements of the task<br>"A r Mi iQ /2p2HQT M 2ti2MbBQM i? i M22/b iQ F22T i` +F r?2M i?2 #`Qrb2" `2bi `ibX |
| ai imb .2b+`BTiBQM | Descriptions on the current status of the task, including the functionality that has been implemented or the actual undesirable results obtained<br>"Ai b22Kb i?2 b+`BTi Bb MQi }M/BM; * +Q" HmKM BM . b?#Q ` |
| aQ7ir `2 `iB7 +i | Products or by-products of software systems, including source code, error messages, or runtime outputs<br>"hvT21``Q`, * MMQi `2 / T`QT2`i(B2 b/ BQM;kmMQ/`2)E22/ |
| S`QD2+i 6 +iQ`b | Factors relevant to the software engineering projects, including a range of characteristics of the technical infrastructure or project scope, plan, or application domain<br>"S +F ;2b, KTH v2`- pH+- HH i?2 T +F" ;2b 7`QK # b2@/2p2 |
| S2`bQM H 6 +iQ`b | Programmers' personal backgrounds including expertise in programming, familiarity with application domains, and roles<br>- familiarity: "u2b- A K 7 KBHB ` rBi? bi iB+ HHv ivT2/ H M;m ? x `/b Q7 D p b+`BTi HQQb" 2Hv ivT2/ p `B #H2b<br>- role: "q2 `2 2tT2`i Tvi?QM T`Q;` KK2`b M/ / i M Hvbibf/ i iBbibf`2b2" `+?2`b |

h bF _2[mB`2KQ2Mi describes the goal and the constraints of the software engineering tasks programmers are working on. This code includes the desired functionality, type of functional improvement (e.g., "A M2p2` r Mi Mv /QrMiBK2 r?B+? rQmH/ + mb2"), i?2 BK ;2b or non-functional improvement to make (e.g., "S`QpB/2 b2p2` H bm;;2biBQMb 7Q` TQi2M i? i KB;?i bT22/ mT i?2[m2`vXX, and some constraints like " HH /Bpb TT2iM/22/ BQ Kmbi #2 Q`; MBx2/ bm+? i? i 2 +?2[mQr`2aQ7 /Bpb- MQ KQ`2 M MQ H" ai imb .2b+`BTiQM describes the less-desired current status, such as implemented functionality that requires further work or that does not meet certain functional or non-functional requirements, such as "LQr- Bi B;MQ`2b vQm` /27 mHi #`Qrb2` M/ QT"2Mb HBMFb BM 1/;2 aQ7ir `2 `iBi7+d includes source code, error messages, runtime outputs, and test cases that are generated during the software development life cycle. This is provided to incorporate a detailed context of the software systems and the status.

S`QD2+i 6 +iQ` includes meta-information of the software engineering projects programmers work on, including the technical infrastructure of the system they are working on, the programming languages, platforms, tools, packages, operating systems, and devices.

S2`bQM H 6+iQ` is related to the individuals, describing their professional expertise (e.g., "A K #2;BM7M2`), familiarity with application domains (e.g., "A /QiM? p2 Km+? 2tT2`B2M+2 rQ`FB rBi? ;QQ;H2"), libraries, frameworks, or programming languages, and programmers' roles (e.g., software architect). This is often provided to set the depth of the information generated by LLMs.

## 8.3 RQ8.2: Experimental approach to measuring the value of providing additional contextual factors

From our qualitative analysis, we identified various contextual factors that programmers consider when seeking information using large language models (LLMs). This observation aligns with empirical evidence from our previous study detailed in Chapter 7, which indicated a preference among programmers to integrate their context into such systems. Despite these findings, the direct impact of these contextual factors on the effectiveness of information generation remains uncertain, primarily due to challenges in quantitatively assessing the benefits of context integration. To address this gap, we have designed a novel experimental ablation study. This experimental study aims to quantitatively measure and compare the influence of different contextual factors, thereby elucidating their specific contributions to enhancing the quality of information support provided by LLMs.

### 8.3.1 Study Design

#### LLM-as-a-judge

The most straightforward way of measuring the usefulness of adding contextual factors is to compare the quality of information generated with and without specific contextual factors. Ideally, evaluating the quality of responses would involve consultations with human experts or conducting A/B tests with actual programmers. However, such evaluations can be very expensive, particularly because evaluating the quality of information for real-world software engineering tasks demands extensive technical expertise.

Table 8.3: Summary statistics of LLM coding

| # Prompt | Task Requirement | Status Description | Software Artifact | Project Factor | Personal Factor | Total |
|---|---|---|---|---|---|---|
| 458 | 364 | 66 | 255 | 179 | 47 | 911 |

To mitigate this challenge, we propose an experimental evaluation approach, using to the "LLM-as-a-judge" framework [284]. This method offers a cost-effective and scalable alternative for evaluating various LLMs. In this setup, a robust base LLM, serving as the "judge," is presented with corresponding outputs from two different LLMs, and a specific problem description to be used to ground its evaluation. The judge LLM is then tasked with selecting the preferred answer based on certain criteria, and with providing rationale behind the comparison.

For our study, we compare responses from the same LLM, GPT-4, generated with and without a particular contextual factor. We utilize a prompt template designed by Zheng et al. [284], which instructs the LLM to evaluate responses based on criteria such as helpfulness, relevance, accuracy, depth, creativity, and detail. In this experiment, we choose GPT-4 as the judge, following findings by Zheng et al. [284] that it exhibits the highest agreement level with human evaluations at 85%, surpassing even the between-human agreement rate of 81%.

### Data Collection

To compare the responses generated with and without specific contextual factors, it is essential to include user prompts that incorporate additional contextual factors identified previously. As manually coding the dataset is impractical, especially given the technical complexity, and thus the long conversations, included in DevGPT dataset, we decided to employ an LLM for automatic coding. We utilized a GPT model, following Chew et al. [47], which involves instructing an LLM to perform deductive coding using a predefined codebook. We adopted their prompt template, utilized the codebook from the previous section, and employed the GPT-4-turbo-preview model for this task.

To accurately assess the impact of contextual factors, we only included conversations that consisted of a single user prompt and a single response, with the prompt length limited to 4,096 characters. The LLM coder was allowed to apply the same code (e.g., "project factor") multiple times within a single prompt, but each segment of text was assigned only one code.

The results of LLM coding are detailed in Table 8.3. In our analysis, the LLM coder processed 458 ChatGPT prompts from the DevGPT dataset, assigning a total of 911 codes. A substantial majority of the prompts (79.47%) included task requirements, and over half (55.68%) contained software artifacts such as code snippets or error messages. The most common of the other contextual factors were project factors (e.g., programming languages in use), while personal factors (e.g., programmers' experience level) were the least common.

Although LLM coding facilitated large-scale analysis, the accuracy of the coding was insufficient for direct use in further steps. For instance, the model struggled with personal factors, often confusing context related to the prompt writer (e.g., "q2 `2 2tT2`i Tvi?QM T`Q;` KK2`b M/ / i M Hvbibf/ i b+B2MiB bibf`2b2`+l?2`b with hypothetical scenarios (e.g., "AK ;BM2 #2BM; /2p2HQT2` /pQ+ i2 rBi? 8 v2").`b Q7 2tT2`B2M+2

We also discovered that some contexts, particularly task requirements or status descriptions, were critical to the prompts, which posed challenges for our ablation experiment. In many cases, prompts were entirely coded as task requirements or status descriptions, such as "*`2 i2 Tvi?QM b+`BTiiQ b2M/ .LaT +F2i mbBM; b+ Tv" As excluding these contextual factors would leave insufficient information in the prompts to generate meaningful responses for the experiment, we refined our focus to software artifacts, project factors, and personal factors, which were less intertwined with the essential content. We also encountered prompts where software artifacts were pivotal, such as those containing solely an error message, but decided to keep software artifacts as one of the context categories for ablation study, because there were still many prompts adding software artifacts to enhance LLM responses even when they were not pivotal.

To address issues arising from coding accuracy and to filter cases where the context was critical in the prompts, we conducted a manual review of the LLM coding before proceeding with the ablation experiment. Starting with personal factors, due to their fewer instances, we identified 12 suitable for the ablation experiment. We then randomly sampled instances of software artifacts and project factors and continued filtering until we had 12 suitable instances from each of the three other categories to balance the category distribution in the experiment.

### Experiment Protocol

For an ablation study, for each prompt, we requested LLM to generate responses for the original prompt (e.g., "... Assume I am a beginner and have no git and node installed. ...") in the dataset, and the prompt after excluding the contextual factors (e.g., "... Assume I have no git and node installed." ). We then provided the two responses to the LLM evaluator to attain verdicts. The prompt template for the LLM evaluator consisted of a problem description, explaining the user's request, and two responses to compare. For both LLM generator and LLM evaluator, we used the same model, GPT-4. This consistency helps minimize any potential self-enhancement bias, where evaluators might prefer responses from the same model used for generating answers.

To quantify the impact of context on response quality, we introduced two metrics: score-original and score-removed. The score-original was calculated by providing the evaluator with the original, context-rich prompt as the problem description alongside the two different responses to compare. Conversely, the score-removed was derived by providing the modified, context-reduced (i.e., de-contextualized) prompt, with the two responses. These metrics are intended to reveal the importance of contextual information at different perspectives, with score-original highlighting the importance of MQi KBhbBiM; context, and score-removed assessing the effectiveness of augmenting //BiBQMH context in prompt.

To compensate for randomness and evaluator bias, each evaluation was repeated four times. We also counterbalanced the presentation order of the responses to remove positional bias, where the evaluator might favor the first-presented answer. We computed the scores by counting how many times the response generated with the original, context-rich prompt wins, making the possible range of both scores -4 to 4. A positive score means that the response generated with the prompt with contextual information was judged to be better, a negative score means the opposite, and 0 means the two responses are similar (i.e., tie).

The example original prompt, de-contextualized prompt, LLM generator's responses generated with the two different prompts, the LLM evaluator's comparisons with the rationales, and how we calculated the scores can be found in Table 8.4.

Table 8.4: An example the user prompt and the LLM-as-a-judge evaluation result including the comparison result and the rationale behind it. A/B in the parenthesis indicates which Assistant generated the response for the prompt *with* contextual information.

| **User Prompt** | |
|---|---|
| Mention Link | ?iiTb,ff;Bi?+mQ#Kfo M/BpB2`f B@bM +Fbf#HQ#f<br>+djk k#/8j/ Rj9 /7+82#R7+/y2Ndk2N/j82+8jfiBK2@iQ@H2 `M@+Q/<br>irBii2`@i?`2 /f_:1K/.J1 |
| Org. ChatGPT Conv. | ?iiTb,ff+?QiT2M+ QBKfb? `2f/7d882yR@8R+R@97de@3 /#@3#j2yje8e |
| Prompt (**context** removed for prompt-removed) | **ROLE: We are expert python programmers and data analysts/ data scientists/researchers**<br>CONTEXT: the attached raw-thread-text.txt file contains a raw and unstructured text dump based on a Twitter thread. In this thread, individuals discuss how long it took them to learn to code. ... |
| **LLM Responses** | |
| Response: original | To achieve the task of analyzing the time it took for individuals to learn coding as a skill and then land a professional programming job from a raw ... |
| Response: removed | Given the task at hand, we'll need to execute a few steps to efficiently create a structured dataset from the raw, unstructured text dump... |
| **LLM-as-a-judge Evaluation & Rationale - Score-original: 4** | |
| Problem Desc. | ROLE: We are expert python programmers and data analysts/data scientists/researchers CONTEXT: the attached raw-thread-text.txt file... |
| org-1 (A) | ... Assistant A's inclusion of a anticipatory guidance on potential challenges related to data extraction and the need for advanced NLP techniques, ...<br>Final Verdict: >A>@+@1 |
| org-2 (A) | ... Final Verdict: ** >A>@*@+1 |
| org-3 (B) | ... Final Verdict: >B>@+@1 |
| org-4 (B) | ... Final Verdict: ** >B>@*@+1 |
| **LLM-as-a-judge Evaluation & Rationale - score-removed: 2** | |
| Problem Desc. | CONTEXT: the attached raw-thread-text.txt file contains a raw and unstructured text dump based on a Twitter thread. ... |
| rm-1 (A) | ... Assistant A has a slight edge due to its detailed technical guidance and realistic acknowledgment of the complexity involved in dealing with unstructured text data. Therefore, >A>@+@1 |
| rm-2 (A) | ... >A>@+@1 |
| rm-3 (B) | ... >C>@0@ |
| rm-4 (B) | ... **Verdict** >C>@for a tie. **0** |

Table 8.5: Summary statistics of the comparisons between LLM response generated with prompts, with and without a specific contextual information category. *avg.* columns show the average score, and CI columns indicate the confidence intervals.

| Context | n | Score-Original | | Score-Removed | | Score Difference | |
|---|---|---|---|---|---|---|---|
| | | avg. | CI | avg. | CI | avg. | CI |
| Artifact | 12 | 3.17 | [2.27, 4.06] | 0.92 | [-0.37, 2.20] | 2.25 | [0.87, 3.63] |
| Personal | 12 | 0.92 | [-0.40, 2.23] | 0.08 | [-1.28, 1.45] | 0.83 | [-1.15, 2.82] |
| Project | 12 | 2.08 | [0.56, 3.60] | 0.00 | [-1.65, 1.65] | 2.08 | [0.74, 3.42] |
| All | 36 | 2.06 | [1.33, 2.78] | 0.33 | [-0.43, 1.09] | 1.71 | [0.87, 2.58] |

### 8.3.2 Results: Impact of Including Contextual Factors in Response Quality Enhancement

Table 8.5 summarizes the evaluation results. The average score-original, the score measured with the original, context-rich prompt as the problem description for the LLM evaluator, of 2.06 out of 4 indicates that including appropriate context information, or not missing necessary context in the prompts, can help generate better-quality LLM responses. The average score-removed, the score measured with de-contextualized prompt as the problem description, of 0.33 indicates that additional contextual information leads to answers that are perceived as slightly better, although the effect is not significant. The difference between the score-original and the score-removed (score-original - score-removed) was 1.71 on average, indicating that the benefit of including additional contextual information may be less significant, compared to not missing important, critical context.

#### Software artifact

Both scores suggest that including software artifacts in the prompt is beneficial: the score-original of 3.17 and the score-removed of 0.92 are the highest among the three context categories. The LLM evaluator preferred responses that included software artifacts, noting they "T`QpB/2 KQ`2 ++m`i2-`2H2pMi- M//2i BH2/ ?b2 ?2 QmBb2 ?KQM?2m b2 ?2 QmiTmi 7?QHKiX" and marked them better to "i?2 However, there are indications that including additional software artifacts can sometimes confuse the LLM evaluator, as evidenced by the disparity between the score-original and score-removed. For example, when LLM evaluator is asked to compare two responses with the de-contextualized prompt as the criteria, the LLM evaluator found the response generated with the original prompt to provide unnecessary information, mentioning ".2bTBi2bTQMb2 ;2M2`i2/rBi?i?2Q`B];BMQpHBT7BQMKTpi Hm #H2 //2ibTQMb2/pM-2 -i2/rBi? /2@+QMi2tim HBbx2/2bTQMKTbi2 Bb KQ`2 HB;Mb2/BKKi2/B?2mb2 /bX" suggests that while adding software artifacts may enhance responses in general, it is most effective when the artifacts are directly related to the developer's current task. It also indicates that indiscriminately including every code snippet and error message might not necessarily enhance the quality of an LLM's response.

### Project factor

Responses that are generated with prompts including project-specific factors achieved an average score-original of 2.08. This suggests that integrating these factors allows the LLM to generate more relevant responses tailored to the specific system a programmer is working on, thereby providing solutions that are better contextualized to the task at hand. For example, the LLM evaluator noted, "[ `2bTQMb2 ;2M2` i2/ rBi? +QMi]2Ti`Q pHB2M7Q+`QKKiB`Q?M2MbBp Mbr2`- //`2bbBM; p `BQmb `+?Bi2+im`2b" M/ T`QD2+i `2[mB`2K2MibX

However, when evaluated in a more general setting, with the de-contextualized prompt as the problem description, the inclusion of additional project factors did not improve the score-removed, which remained at 0.00. In some instances, adding project context refined the responses, which proved beneficial. For instance, one response was described as "[T`QpB;/bMH;B;?iHv KQ`2 i2+?MB+ M/ BM@/2Ti?" Conversely, for certain project factors (e.g., "AK r`BiBM; M BS?QM2 arB7 K2/Bi iBQ"), this additional information led to responses being overly specific, such as in cases where "i?2 `2bTQMb2 r b iQQ /2i BH2/ 7Q` mb2` r?Q /B/ MQi bT2+B TH i7Q`K M/ bF2/ #`Q /[m2biBQMX

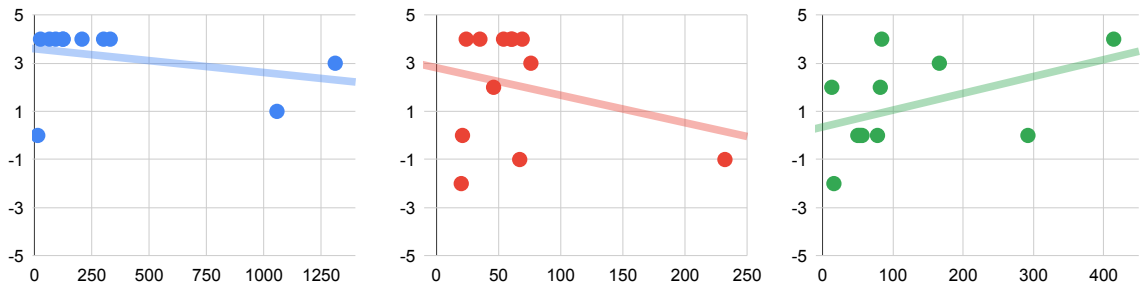
### Personal factor

The average score-original of 2.06 out of 4 suggests that including relevant personal characteristics in the prompts (e.g., user's experience level) can enhance the LLM's responses by allowing it to adjust the complexity and provide more detailed explanations. For instance, the LLM evaluator noted that "[`2bTQMb2 ;2M2` i2/ rBi? i?2 Q`B;BM H T`QKTi BM/bHm/BM; i?2 H v2` Q7 /2Ti? iQ i?2 /pB+2 i? i+QmH/ #2 2bT2+B HHv #2M2?}+B H 7Q` /
However, the differences between score-original and score-removed were the highest among the three context categories, as indicated by wide confidence intervals, 3.97, in score differences. This variability may be because personal factors influence the tone and depth of answers generally, so the evaluation heavily changes based on the problem description. This may indicate that although including the personal context might support LLMs generating better responses in general, but not consistently.

### Length analysis

Previous research has found that LLM evaluators tend to prefer longer responses due to a length bias [284]. Given that LLMs often produce longer responses when prompted with longer inputs, this could have influenced our results, as the original prompts were designed to be longer than the de-contextualized prompts. To investigate the presence of a length bias in our evaluation, we conducted a lightweight sanity check on whether a comparative quality benefit—specifically, higher scores—was attributable to the length (number of characters) difference between the original and de-contextualized prompts.

The results, shown in Figure 8.1, suggest that the influence of length bias might not be significant. Despite the challenges of conducting statistical tests due to the small sample size, there is no clear trend indicating that scores increase with the length difference; in fact, the opposite trend was observed for software artifacts and project factors. This finding suggests that augmenting user prompts with overly lengthy context may not always contribute to the response quality enhancement, as it could potentially confuse the LLM or lead to overly specific responses, diminishing the quality of the output.




(a) Software Artifact (b) Project Factor (c) Personal Factor

Figure 8.1: Scatter plots showing the trends between the character length difference between the original and de-contextualized prompts (x-axis) and the score-original (y-axis).

## 8.4 Discussion

By analyzing the information needs and contextual factors shown in LLM prompts, we have identified 6 types of programmer information needs and 5 types of contextual factors programmers provide as part of their prompts. In addition, through an experimental ablation study gauging the usefulness of the contextual factors, we saw promising findings indicating that adding these contextual factors can enhance the quality of LLM-generated information.

### 8.4.1 Study Design Limitation

However, although the ablation study to measure the importance of contextual factors is novel, and utilizing LLM-as-a-judge idea allowed us to automate the experiment, it also still holds many limitations.

#### Multiple layers of LLM usage

The study design involves multiple layers of LLM usage: we first generate responses using LLMs, with and without contextual factors in the prompt, and then evaluate these responses also using an LLM. While this methodology enables the automatic quality evaluation of different contexts—potentially facilitating larger-scale studies in the future—it also introduces significant noise into our experiment. Specifically, we observed inconsistencies in the criteria used by the LLMs during response generation and evaluation. Even though we employed the same model, GPT-4-turbo-preview, for all tasks, the necessity to initiate each chat anew made it challenging to maintain a consistent rationale and evaluation criteria between the generating and evaluating LLMs. For instance, the LLM generator provided a response to the prompt including a personal factor "I am a beginner" by omitting complex troubleshooting details present in the de-contextualized version, possibly because beginners might not be able to understand or need the troubleshooting details. However, when the LLM judge compared the two responses for score-original, it favored the de-contextualized response for including detailed troubleshooting information, noting: "h?Bb bK HH /2i BH +QmH/ bB;MB}+ MiHv B/ mb2`b 7 +BM; 2t2+miBQM" Inconsistencies, and unclear rationale behind LLM responses of each layer, necessitated the manual investigation to interpret the results, which we discuss more next.

### Need for human verification

In our study, we encountered significant limitations stemming from the need for extensive manual verification, contradicting our initial goal of using LLM evaluations to facilitate a large-scale study. This need arose due to inaccuracies in the coding step (see Section 8.3.1-Data collection), inconsistencies between the LLM generators and evaluators, and challenges in interpreting LLM evaluations. Specifically, manual qualitative coding limited our dataset to only 60 ChatGPT conversations for code identification. Furthermore, the need for preliminary manual verification of LLM-generated codes restricted our analysis to a sampled set of prompts for the ablation study. Expanding human resources could potentially aid in scaling the experiment; however, the expensive nature of human evaluation, particularly for technically complex content, inherently limits the practical expansion of the study. Therefore, we believe that developing mechanisms to reduce the reliance on human verification or to streamline the human evaluation process will be crucial for effectively scaling this experiment in future research.

### DevGPT dataset

The inherent limitations of conducting experiments with a pre-generated dataset, such as DevGPT, are also significant. For example, DevGPT, collected from public, open-source interactions with ChatGPT in October 2023, does not include data from proprietary software systems. Thus, the contextual factors identified in the DevGPT data may not be the extensive set of all possible factors users might use, as we might have missed them in coding. Moreover, the dataset's temporal scope means that many conversations may have been exploratory in nature, as the snapshot we used in the study was collected in October 2023, not too long after ChatGPT's public release in November 2022. Despite our efforts to manually filter out irrelevant interactions, such as philosophical inquiries or tests of malicious intent, the broader evaluation of contextual factors' value might still inadvertently include such data. Thus, we do not claim that we identified the extensive set of contextual factors or our finding can be generalized, and believe that similar studies with different populations, ideally with larger-scale, are needed to better reflect the evolving real-world use of LLMs by programmers.

## 8.4.2 Potential for automatic contextualization

Despite the limitations, the experimental study provides some promising preliminary evidence on the benefit of not missing important contextual factors, or even including additional ones. Thus, we argue that it might be worth investigating ways of automatically augmenting the user prompts by adding contextual factors, to enhance the programmers' information seeking in general. This may alleviate the programmers' challenges in forming effective prompts for LLM due to their lack of expertise in application domains or programming. In this section, we provide some ideas on how we can automate this context augmentation of the user prompts. Specifically, we examine each contextual factors we discovered focusing on where they can be found (source) and whether they change over time (variability).

a Q m ` + Z d X Not all contextual factors are observable. For example, the programming language might be easily identifiable from the file names, but it will be harder to identify the quality requirements of the software system. General contextual search mainly uses observable variables (e.g., search history) as it is difficult to make direct use of unobservable variables (e.g., demographic data). However, as most of the software engineering artifacts are archived, we envision

that some of the commonly unobservable factors that will be useful in developer information support can be identified from other sources. For example, the history of software systems, issue logs, continuous integration logs, or code reviews in version control systems (e.g., GitHub) will be useful in inferring both the history of the developer and what they work on. From the developer's IDE, the current project, code-base, and interaction traces can be collected to understand the developer's current context. For contextual variables that are not easily inferred from any of the sources, it might also be possible to ask users directly.

o `B #B HAB Bli X As software evolves, as do its developers, so most contextual factors change over time. Thus, the context might also be re-inferred or updated as the software and developer change. Knowing how often they get updated is necessary to properly update the context, as they can vary by context factors. For example, the job title of a developer rarely changes, but their familiarity with a certain API method will be updated fairly frequently.

### Software Artifact IDE logs, search queries, prompts

Software artifacts, including code snippets, test cases, or error messages can be extracted from the local file systems, online source repositories, continuous integration tools, and input/output of terminals. In addition to the above, the developers' IDE interaction history will also be useful in understanding the developer's interest (e.g., focal points in code) and clarify the relevancy of software artifacts to the current task of the developer. However, software artifacts can frequently change, as developers work on multiple tasks [85]. Sometimes, developers might work on multiple tasks concurrently [114], so inferring the relevant software artifacts to include will be more challenging than identifying appropriate context of other types.

### Project Factor design documents, backlogs

Project context, such as the length of the project or technical infrastructure, does not change often or changes slowly, especially when the project is well-planned. Technical infrastructure, like programming languages, libraries, their versions in use, operating systems, and devices, will be easier to extract. Many of the technical infrastructure can be inferred from the configuration and build files, such as T Q K X for K Maven or T + F ; 2 X D b QM Q M 7 B ; X D b QM for Typescript. Technical infrastructure does not often change after the software design phase, unless there are large refactorings or significant changes in requirements. Project scale, size, and domain can be easily inferred if there are written requirements or design documents, and the status can be inferred from the backlogs or task management tools (e.g., Kanban boards). However, when the project plan is not well documented (e.g., when it is initiated in a casual setting and grows organically), it is harder to infer the project scale or size, and explicit user inputs will be necessary.

### Personal Factor user input, IDE logs, personal repo.

The simplest way of extracting personal factors, including the developer's expertise in programming, application domains, libraries, tools, and roles, is to explicitly ask the developer to provide the information. As backgrounds or preferences will only rarely change and can be easily answered by users, it might be worth asking them when they start using the information support, as many recommendation systems (e.g., Apple Music) do when users sign up. The full personal context may not be obvious if not provided by the developers, but parts are still inferable from external sources. Personal repositories in a version control system or professional networking platforms

will be useful in inferring the expertise levels, skills, and backgrounds. They contain the history of the developer's programming works from which their use of programming languages, libraries, design patterns, and API methods can be easily inferred. In capturing developer backgrounds as well as their skills and expertise, LinkedIn or resumes will be great sources.

## 8.5 Summary

In this chapter, we identified the information needs of programmers and the contextual factors programmers use in the LLM-powered information support setting by analyzing the ChatGPT prompts used in the software engineering context. We further conducted an experimental ablation study to measure the importance of including such contextual factors in prompts, in enhancing the LLM response quality. The study provided promising preliminary evidence that adding contextual factors can be useful in providing better information support based on criteria such as helpfulness, relevance, accuracy, depth, creativity, and detail. We believe that this chapter motivates the appropriate augmentation of diverse contextual factors in providing LLM-powered information support for programmers, and provides a good starting point for the future generation of information support that is more contextualized and, thus, more effective.

# Chapter 9

# Conclusion & Future Work

## 9.1 Summary of Contributions

This thesis makes a number of major contributions to enhancing information support for programmers' learning, including:

- A thorough review of the background and related work on programmers' information seeking in general, and the various information support approaches (Chapter 2).
- Evidence that programmers' different use of documentation correlates with their user characteristics (e.g., experience level with the API), and with their future API adoption (Chapter 3).
- A novel approach for documentation design review using page-view log analysis (Chapter 3).
- MARBLE, an automatic approach to identify boilerplate code from API client code (Chapter 4).
- A dataset of boilerplate code for 13 popular Java libraries (Chapter 4).
- Evidence that push-based comparable API methods information support can enhance programmers' understanding of API design space (Chapter 5).
- A prototype information support tool that presents comparable API methods within Chrome browser (Chapter 5).
- SOREL, a learning-based model extracting comparable API methods and the support evidence from Stack Overflow posts (Chapter 5).
- A dataset of comparable API methods including 587 Stack Overflow answers, with 198 pairs of comparable API methods and 737 sentences summarizing the relations (Chapter 5).
- Showing the feasibility of providing programming by example support for real-world library users to mitigate the issue of the lexical gap (Chapter 6).
- A language model, which can predict a sequence of API methods given input and output value pairs (Chapter 6).

- Evidence that in-IDE, context-aware, prompt-less support using an LLM can help programmers working with new APIs complete more tasks (Chapter 7).

- A prototype in-IDE information support tool called GILT, that generates on-demand information using an LLM while considering the user' s local code context (Chapter 7).

- A novel experimental approach to measure the value of adding different contextual factors into LLM prompts in the response quality (Chapter 8).

- A preliminary evidence that including contextual factors are useful in enhancing the LLM response quality (Chapter 8).

## 9.2 Discussion & Future Work

Efficient and effective information seeking for programming is still a largely unresolved problem, requiring lots of research. > Q rdifferent programmers seek information is still an open problem, especially with the introduction of LLM-based information seeking, which totally changes the strategies and behaviors of programmers. There are many practical future work items, including extending the prototype tools to support other languages or libraries, but in this section, I focus on future directions that require collaborative efforts.

### More Personalized and Broader Information Support

In this thesis, I have concentrated on providing information support for professional programmers dealing with code that involves unfamiliar APIs and concepts. This work represents an initial exploration into the design of intelligent information support tools. Moving forward, it is imperative to consider more diverse scenarios (e.g., software design, refactoring) and users (e.g., end-user programmers and software architects). To support more diverse programmers in information seeking, it will be necessary to first understand the specific challenges they face with the intelligent information support tools. In-depth analysis with end-user programmers or programmers with specific roles can be conducted, like those reported in Chapter 7. To support broader tasks, information categories that have not covered in this thesis should also be prepared, beyond learning resources like documentation or usage examples. The various information needed for tasks like system design, debugging, refactoring, and testing, often involving multiple source files, may be prepared with program analysis techniques, in addition to the learning-based approaches trained with code data.

To provide support for broader tasks and users, further research is required to determine the extent and nature of the additional context needed, as its impact on response quality was highlighted in Chapters 7 and 8. Particularly, there is a need to develop a better understanding of the appropriate scope of context, to build sophisticated methods that can accurately predicting user intent, task scope, and relevant context for effective augmentation. For this, we advocate for a larger-scale version of our experimental ablation study, as detailed in Chapter 8, using more recent data that mirrors real-world usage patterns rather than exploratory usage of LLMs. Such a study would be invaluable in identifying common strategies among programmers and understanding the relationship between the necessary context and their information needs.

## Smooth Transitions of Programming Tools with New Intelligent Solutions

Research on integrating LLMs within the framework of existing programming tools, originally developed under more traditional paradigms, presents numerous opportunities to facilitate smoother transitions for programmers. Adopting interfaces with which programmers are already familiar is a good strategy, as it will leverage existing user comfort while introducing advanced capabilities. For instance, the introduction of prompt-less interactions in Chapter 7 in GILT proved effective, as programmers were already familiar with retrieving some information with a shortcut or a button, by using static analysis tools within IDE. Continuing this exploration by further integrating context-sensitive and user-friendly interfaces can significantly reduce the cognitive load on programmers as they adopt new programming tools.

Moreover, incorporating ideas used in more conventional programming tools into LLM-powered tools can enhance the experience for programmers familiar with chat-based interfaces. For example, although the SOREL system, described in Chapter 5, targets programmers who primarily seek information through search engines, the concept of push-based information delivery can be adapted for chat-based interfaces. This adaptation could involve suggesting follow-up queries or offering additional information proactively, even if the user has not explicitly requested it. Ideas like programming-by-example support in Chapter 6, and Chapter 4' s discussion on the utility of presenting boilerplate code, are also transferable across various interface designs. Reusing such strategies will provide opportunities to improve the usability and effectiveness of existing chat-based LLM tools.

User interfaces for intelligent information support systems also require research. Through our analysis, we observed that how and when information is presented can influence the benefit (e.g., Chapter 5), and can even benefit programmers differently (e.g., Chapter 3 and 7). Although some researchers have explored different design options in designing LLM-powered programming tools [21, 275], I believe more exploration is needed to support diverse populations. Specifically, the chat interface, which is commonly used for LLM-powered tools, is known to introduce challenges when the users are not already familiar with the application domains. Given that programming tools often have access to many contextual factors, more natural interfaces can be built by using such additional factors to infer users' information needs and intents. We believe that further research is needed, exploring various interaction options to support a diverse programmer population.

## Understanding evolving information-seeking strategies among programmers using generation-based information support

Recent developments in LLM-powered programming tools have introduced a new paradigm for programmer information seeking, allowing for the generation rather than retrieval of information. Despite this, as discussed in Chapters 7 and 8, programmers still employ and need traditional information-seeking strategies similar to those used in conventional settings, including skills like query revision to elicit useful responses from LLMs.

We hypothesize that the continued reliance on traditional strategies may be attributed to the novelty of LLM-powered tools, with programmers still in the early stages of adapting to these new technologies. As prompt engineering techniques improve and guidelines for effective interaction with LLMs become more widespread, we anticipate that programmers might adopt entirely new

strategies. Moreover, we predict that these strategies could vary significantly across different generations of programmers, ranging from senior software engineers accustomed to retrieval-based searches to junior engineers who are more familiar with generation-based approaches. Thus, we believe that understanding how these strategies evolve will be crucial for developing LLM tools that provide effective support for programmers. This will not only enhance the productivity of current programmers but also shape the training and integration of future professionals in the field.

### Next-generation SE with Programmer-AI partnership

In addition to the aforementioned future research directions, it is also important for us to understand what + Me automated by intelligent solutions, and what b ? Q mi H be done by programmers, because, eventually, the software systems will be maintained and used by people, and it is important for programmers to have an understanding of and control over the systems they build. At the same time, it will be important to design and build the next-generation software engineering processes, by answering questions like "What processes should we use to ensure the safety, robustness, and privacy of software systems built with AI tools?" Programmer-AI alignment should also be studied further, by answering "How can we design better evaluation and monitoring approaches to align AIs with SE experts?" and "How can we prevent misspecification and misgeneralization?" Finally, to guide the advancement of intelligent solutions to be aligned with what programmers need, further research is needed on how we evaluate the models. Specifically, building benchmarks that are more grounded towards real-world software engineering will be beneficial in extending the programming benchmarks, like HumanEval [44] or MBPP [22], that are widely used to evaluate foundation models and provide evaluations that are better aligned with real-world tasks.

### Inter-community Collaborations for Future Programming Paradigms

Beyond the goals above, it is necessary for NLP, HCI, and SE research to forge future programming tools that are not only accurate but also highly usable. Given that the forthcoming years will witness a surge in tool development, potentially reshaping the programming landscape, it is imperative that NLP researchers incorporate actual user needs into their model designs, and that tool creators actively address assumptions and challenges posed by machine learning models.

## 9.3 Concluding Remarks

Software engineering is an information-intensive discipline. While building and maintaining software systems, programmers face a broad spectrum of questions ranging from implementation specifics to architectural concerns. However, satisfying their information needs is not easy, because the relevant information is often scattered across varying mediums in different formats. It becomes even more challenging when a programmer needs to work with unfamiliar code or libraries, without the necessary knowledge and experience to search for information effectively.

To mitigate some of the challenges in information seeking for programming, my research has investigated designs, algorithms, and evaluation approaches for intelligent information support for programmers. By combining multiple research methods at the intersection of Software Engineering, Human-Computer Interaction, and Natural Language Processing, I gained a richer

understanding of the information seeking of programmers as users and also found many insights on how we can build better intelligent techniques to provide information support. For example, Chapter 5 not only demonstrated that the push-based interaction support can help programmers understand the API design space, but also built a learning-based model that can automatically extract comparable API methods information from Stack Overflow. Or, in Chapter 7, by building GILT and evaluating it with programmers, we could understand how programmers use LLM-powered information support tools for code understanding, and showed that utilizing context can be useful in information support, which led to the follow-up study of contextual factor investigation. The series of projects in this dissertation, involving iterations of user studies and tool building, demonstrates the importance of having user-centered tool support that helps information seeking for programming. It has been shown that user-centered intelligent support not only yields valuable insights for designing more useful and usable tools but also improves the performance of intelligent techniques used in these tasks.

As technology evolves at an unprecedented pace, there is a substantial effort being made to create intelligent programming tools. However, there is still a disconnect in understanding how programmers will use these tools and the implications of using them in real-world software engineering. Like what I have demonstrated with this dissertation, the synergy that could be achieved by studying both users and tools together holds immense potential. Thus, it is my hope that more intelligent technique development efforts will adopt a user-centered approach. By placing the users, here programmers, at the heart of tool creation, we can ensure that the technological advancements we make are not only innovative but also practical and impactful for the programmers who will use them, and eventually, the users of the software systems built by the programmers.

# Appendix A

# More Results for Logs Analysis

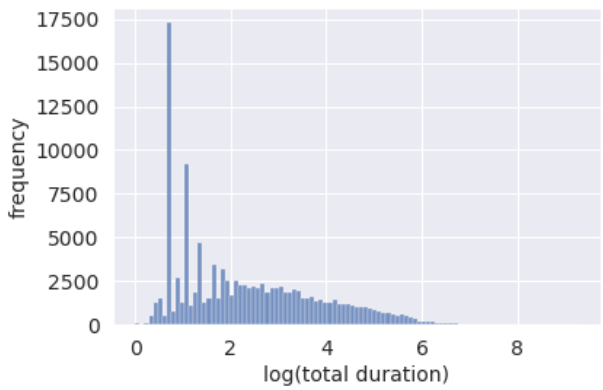



Figure A.1: Distribution of the log-transformed total dwell time (in minutes) on documentation.

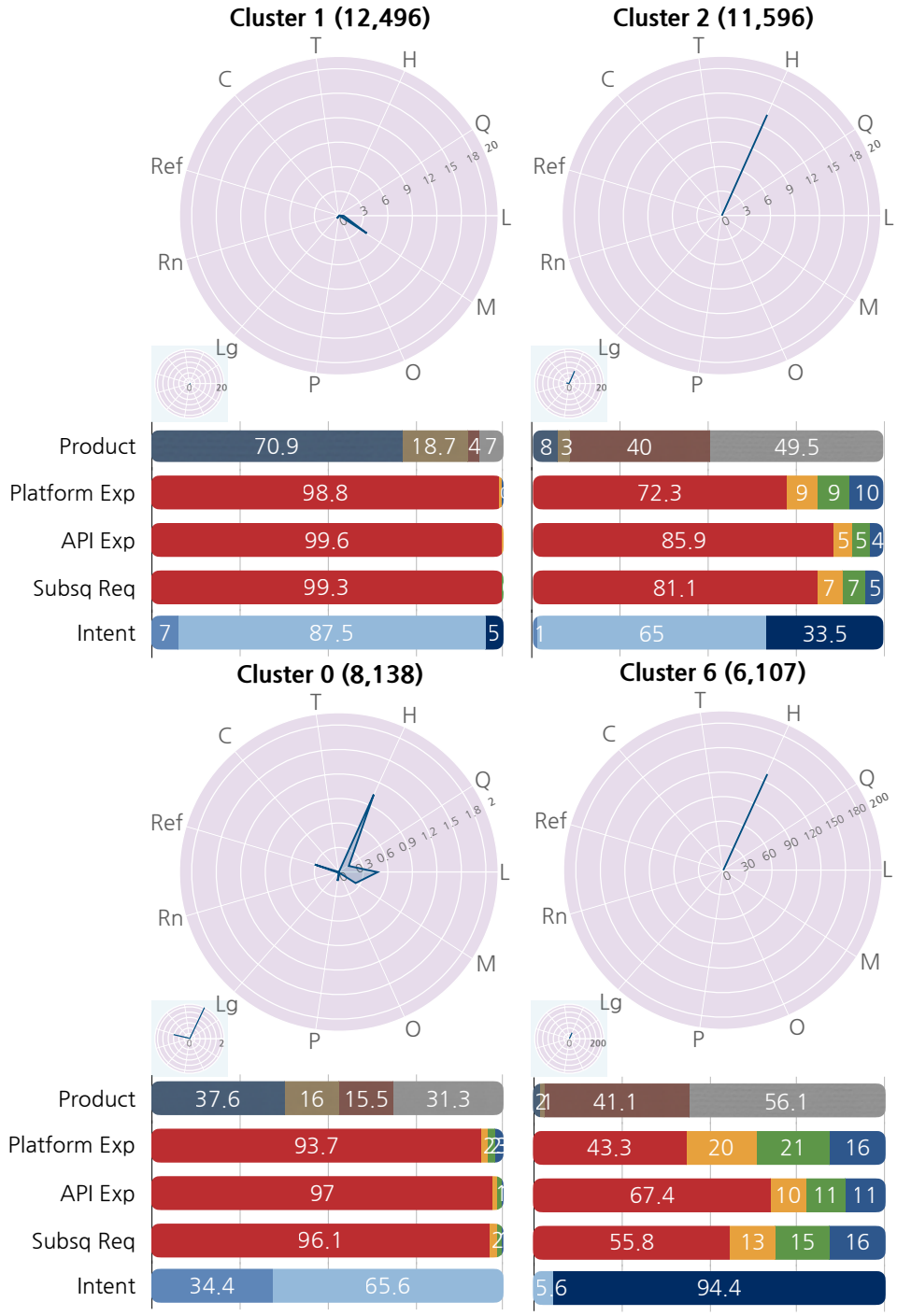



Figure A.2: Large clusters with more than 500 users sorted by the number of users (shown in parentheses). Each polar plot displays the average time spent on each type of documentation (T: Tutorial, H: How-to, Q: Quickstart, L: Landing, M: Marketing, O: Other, P: Pricing, Lg: Legal, Rn: Release note, Ref: Reference, C:Concept). The small polar plots show the average dwell time in the previous three months. Note that the ranges of the axes of the plots vary. Bar charts below the polar plots show the proportions (%) of each group in the cluster.

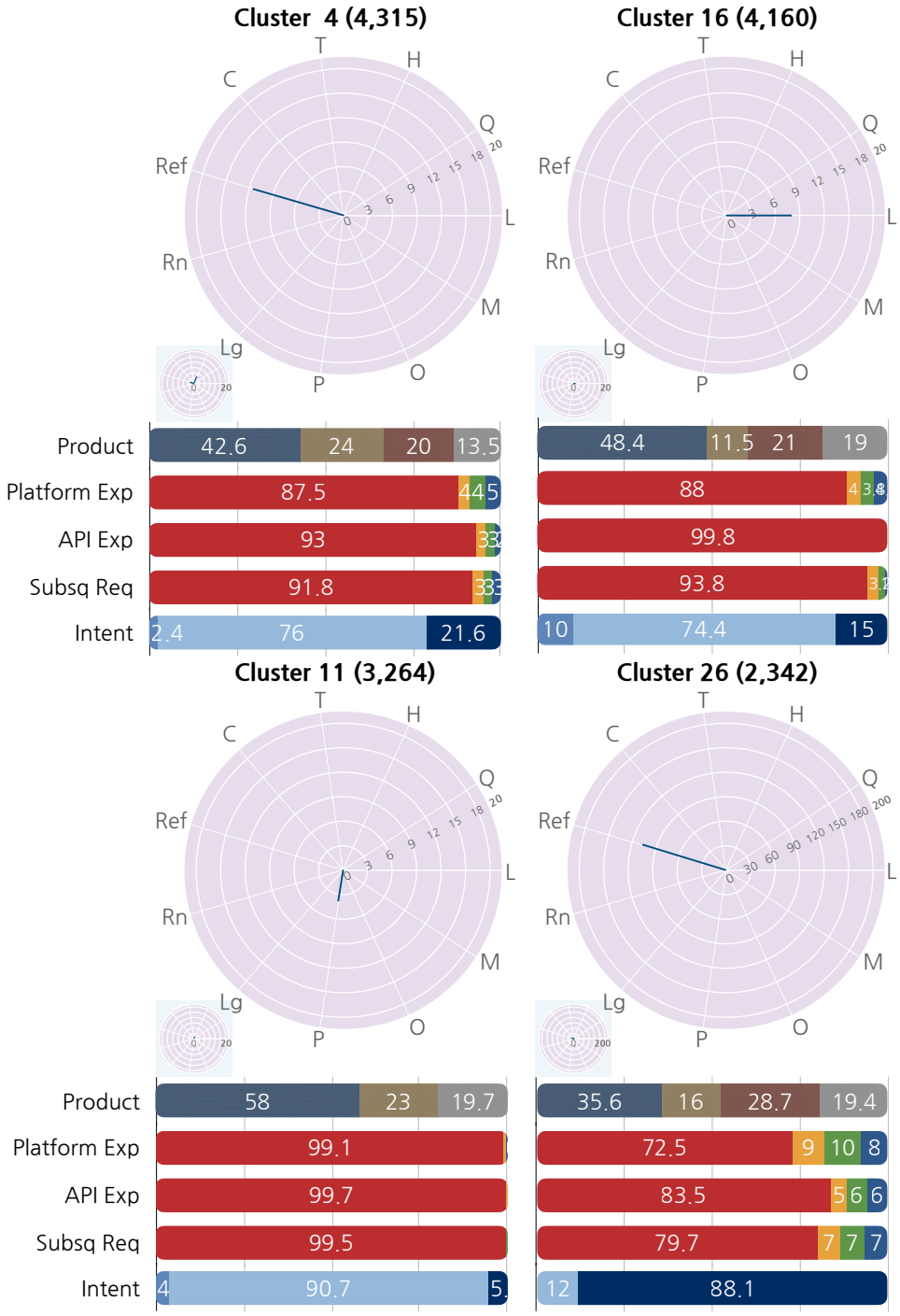



Figure A.3: Large clusters with more than 500 users sorted by the number of users (shown in parentheses). Each polar plot displays the average time spent on each type of documentation (T: Tutorial, H: How-to, Q: Quickstart, L: Landing, M: Marketing, O: Other, P: Pricing, Lg: Legal, Rn: Release note, Ref: Reference, C:Concept). The small polar plots show the average dwell time in the previous three months. Note that the ranges of the axes of the plots vary. Bar charts below the polar plots show the proportions (%) of each group in the cluster.

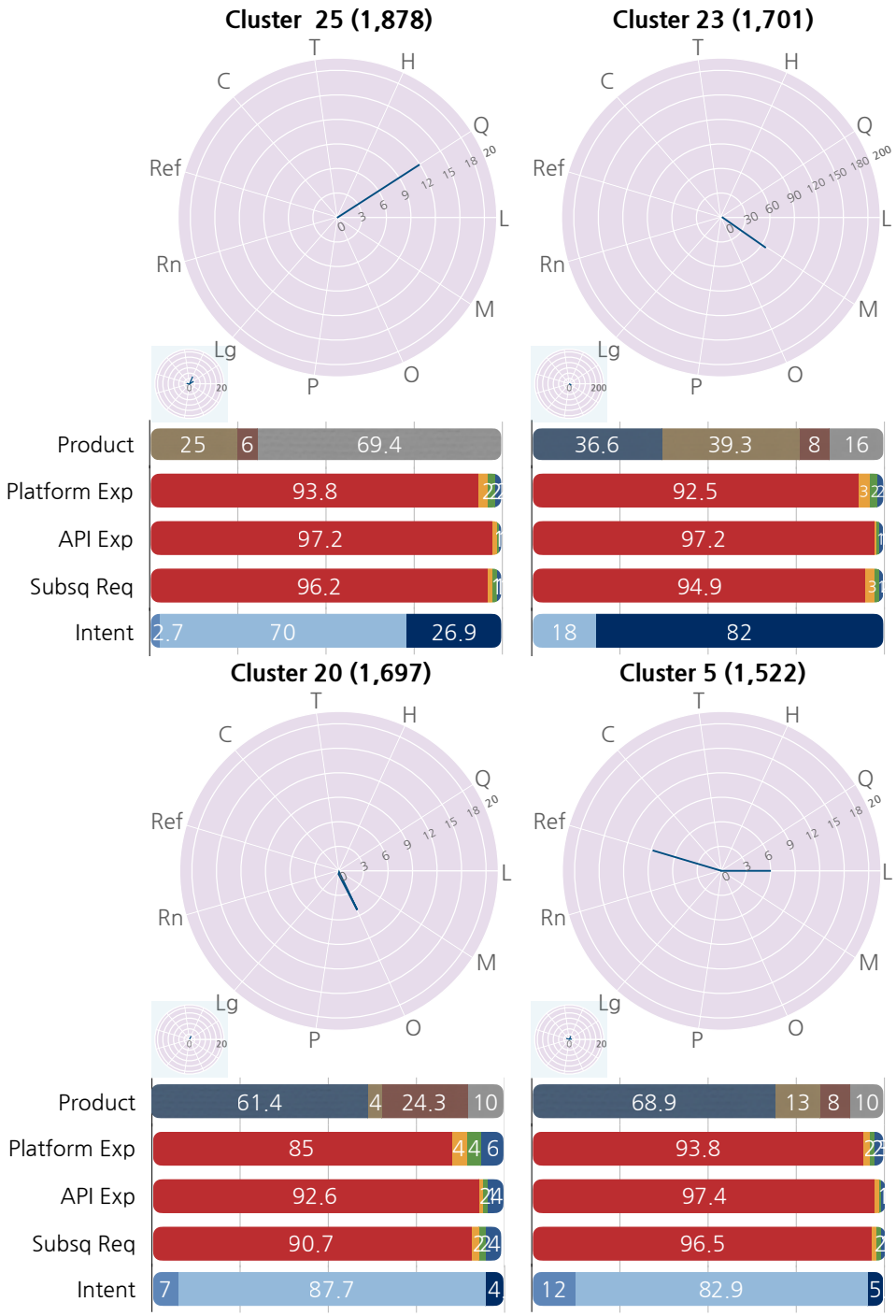



Figure A.4: Large clusters with more than 500 users sorted by the number of users (shown in parentheses). Each polar plot displays the average time spent on each type of documentation (T: Tutorial, H: How-to, Q: Quickstart, L: Landing, M: Marketing, O: Other, P: Pricing, Lg: Legal, Rn: Release note, Ref: Reference, C:Concept). The small polar plots show the average dwell time in the previous three months. Note that the ranges of the axes of the plots vary. Bar charts below the polar plots show the proportions (%) of each group in the cluster.

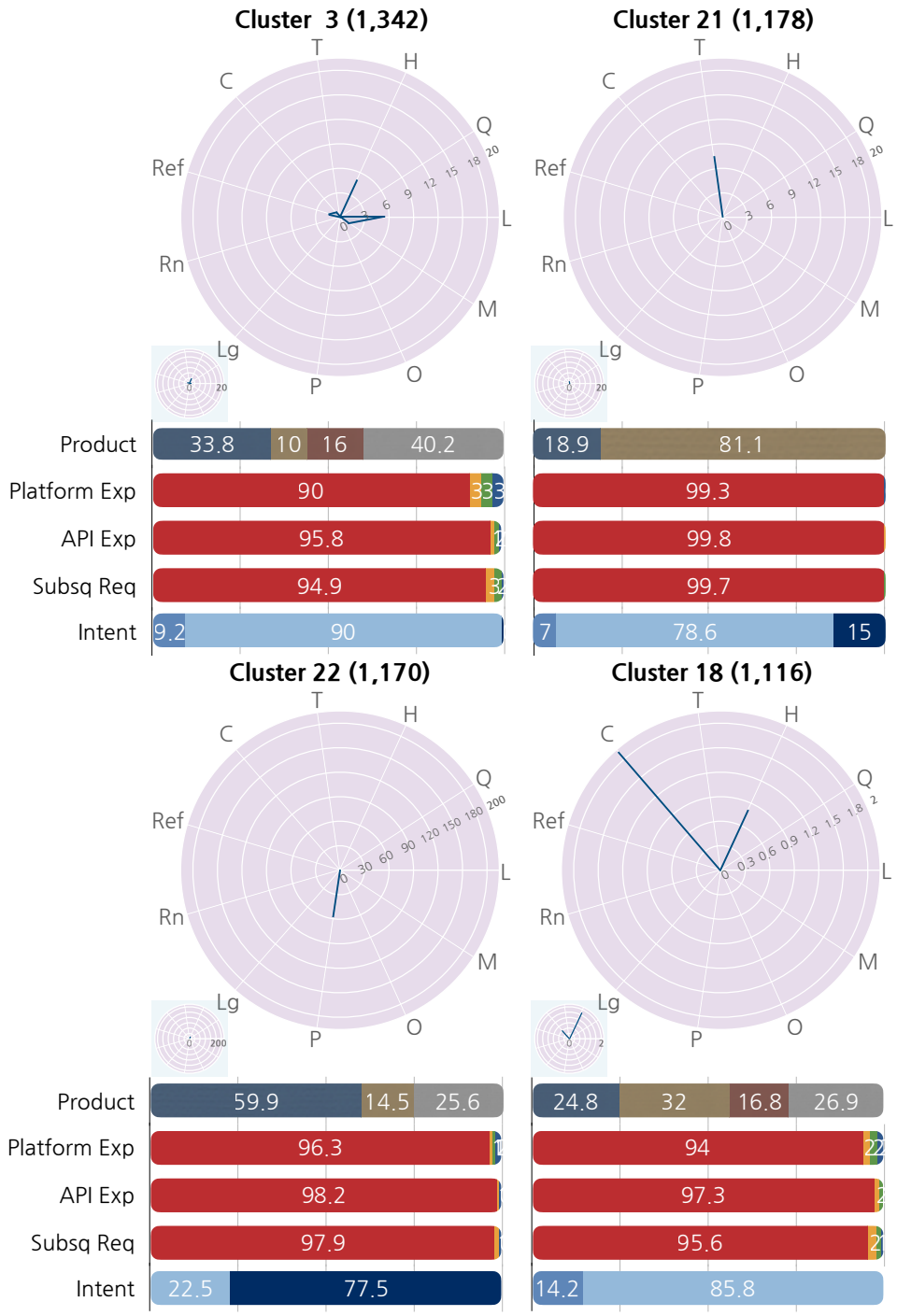



Figure A.5: Large clusters with more than 500 users sorted by the number of users (shown in parentheses). Each polar plot displays the average time spent on each type of documentation (T: Tutorial, H: How-to, Q: Quickstart, L: Landing, M: Marketing, O: Other, P: Pricing, Lg: Legal, Rn: Release note, Ref: Reference, C:Concept). The small polar plots show the average dwell time in the previous three months. Note that the ranges of the axes of the plots vary. Bar charts below the polar plots show the proportions (%) of each group in the cluster.

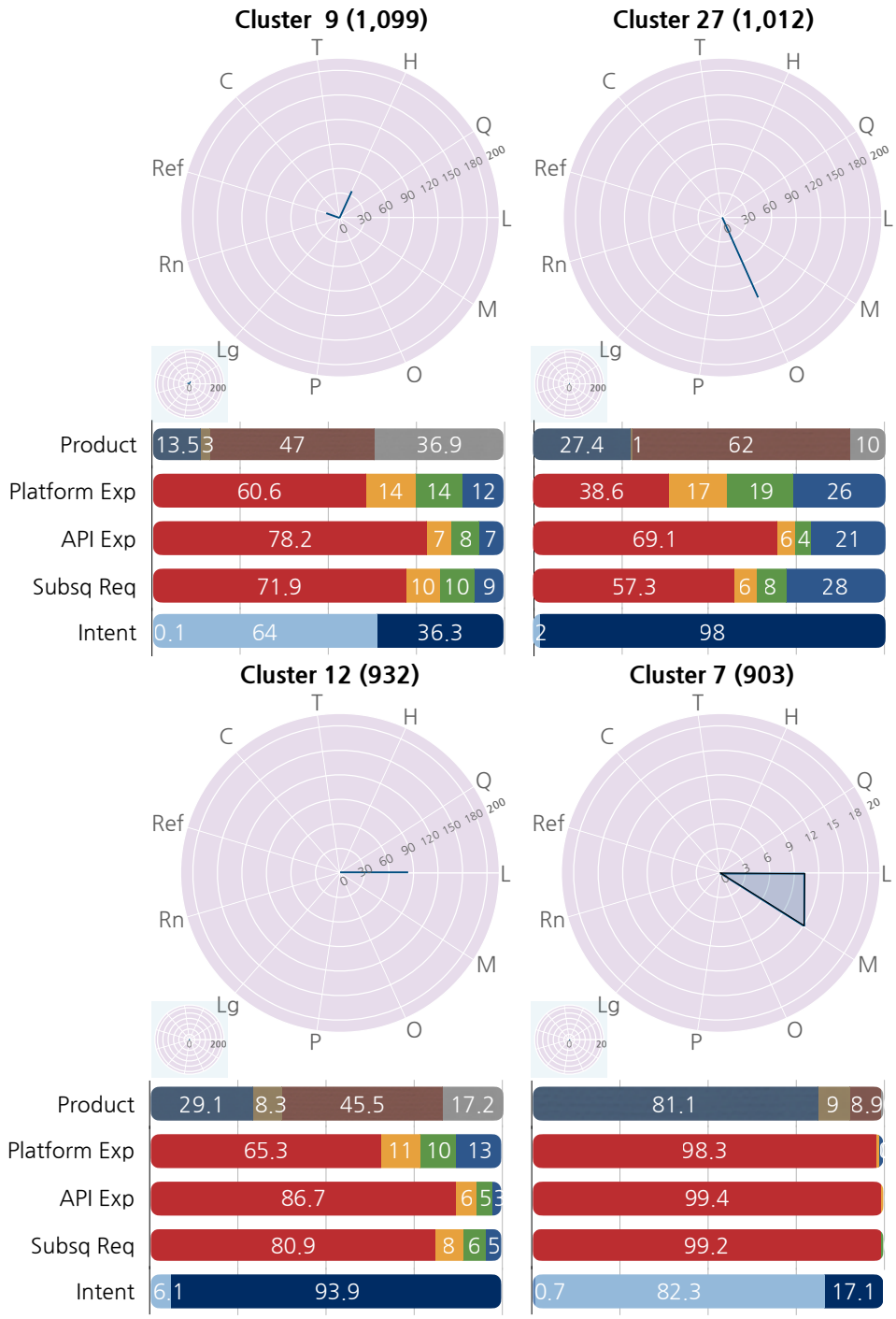



Figure A.6: Large clusters with more than 500 users sorted by the number of users (shown in parentheses). Each polar plot displays the average time spent on each type of documentation (T: Tutorial, H: How-to, Q: Quickstart, L: Landing, M: Marketing, O: Other, P: Pricing, Lg: Legal, Rn: Release note, Ref: Reference, C:Concept). The small polar plots show the average dwell time in the previous three months. Note that the ranges of the axes of the plots vary. Bar charts below the polar plots show the proportions (%) of each group in the cluster.

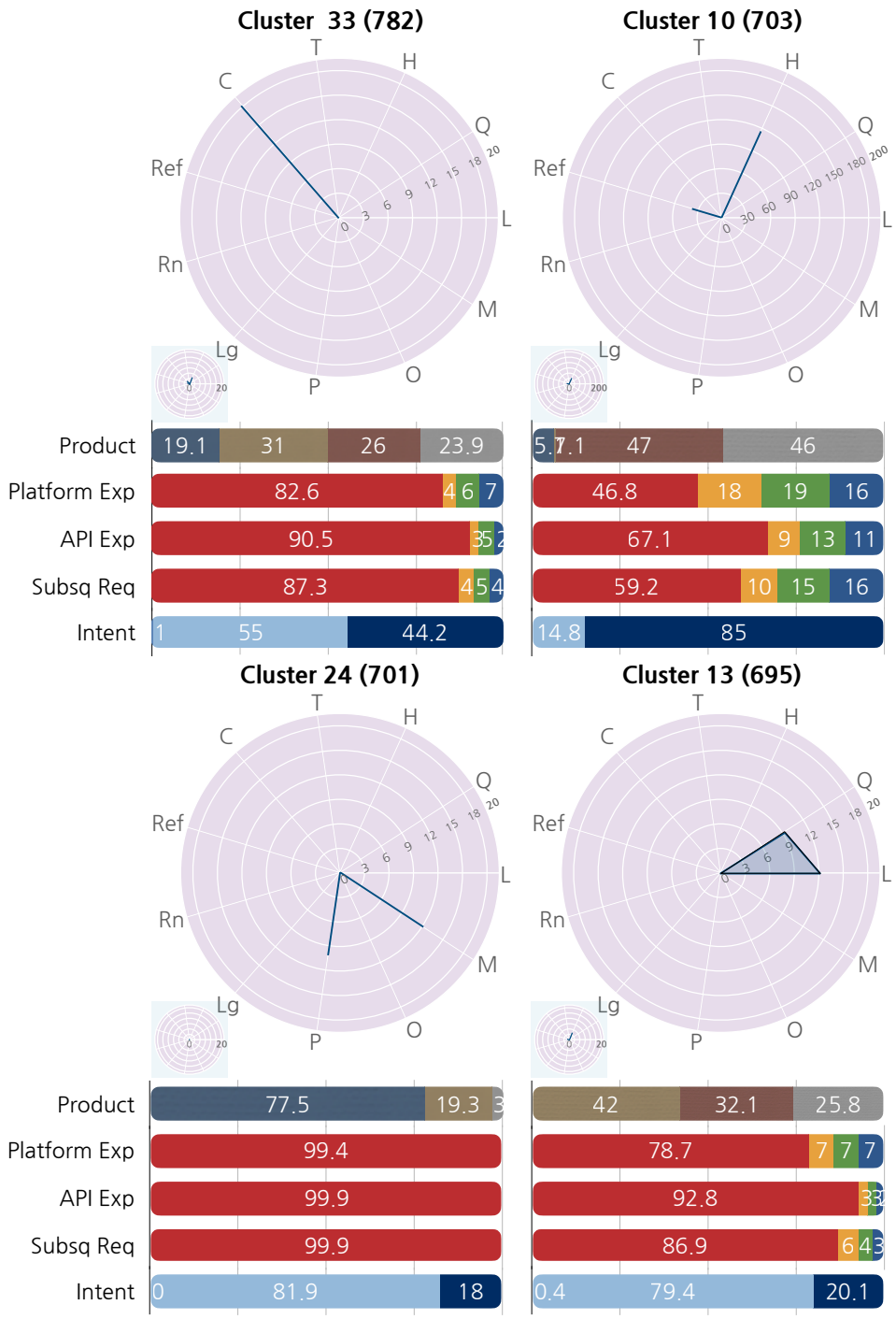



Figure A.7: Large clusters with more than 500 users sorted by the number of users (shown in parentheses). Each polar plot displays the average time spent on each type of documentation (T: Tutorial, H: How-to, Q: Quickstart, L: Landing, M: Marketing, O: Other, P: Pricing, Lg: Legal, Rn: Release note, Ref: Reference, C:Concept). The small polar plots show the average dwell time in the previous three months. Note that the ranges of the axes of the plots vary. Bar charts below the polar plots show the proportions (%) of each group in the cluster.

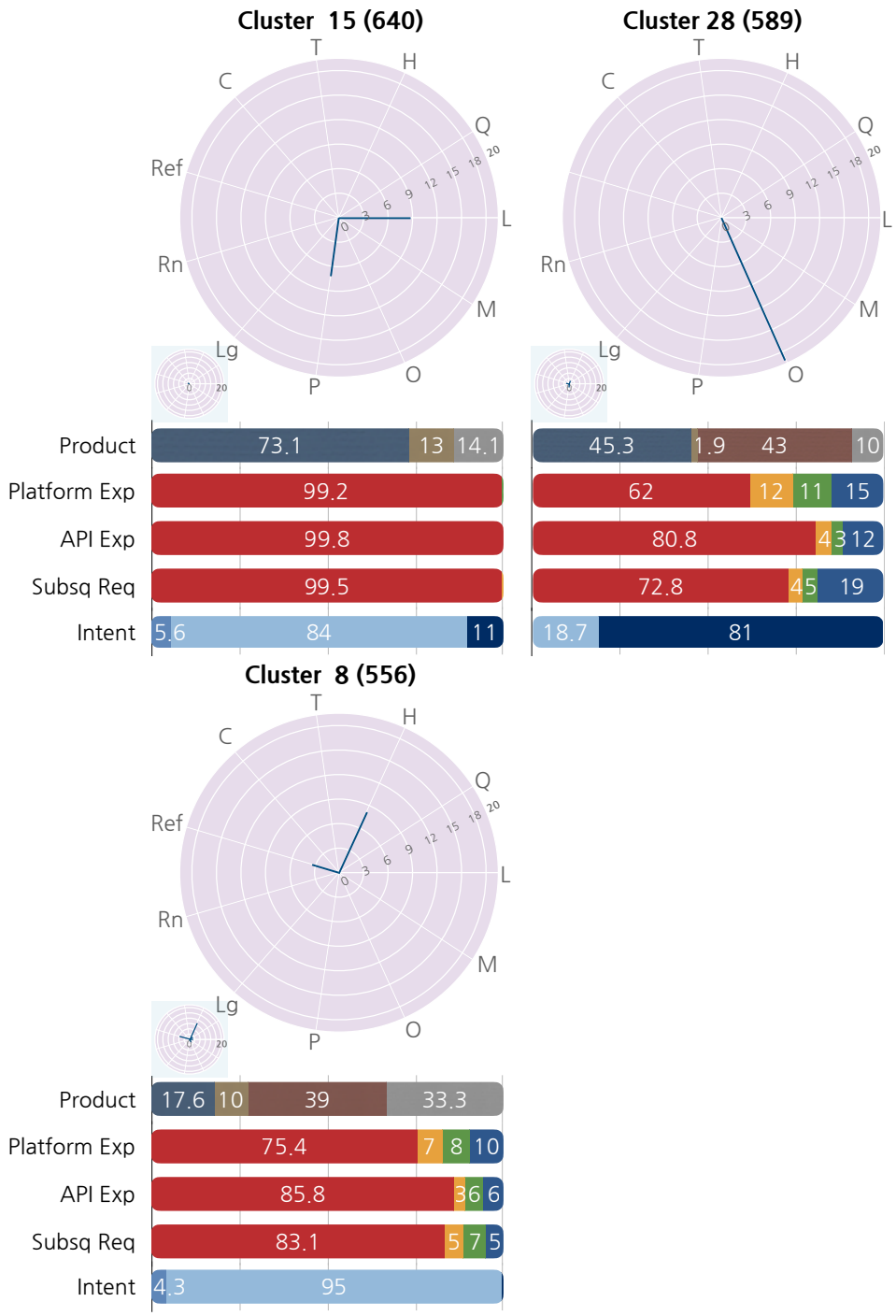



Figure A.8: Large clusters with more than 500 users sorted by the number of users (shown in parentheses). Each polar plot displays the average time spent on each type of documentation (T: Tutorial, H: How-to, Q: Quickstart, L: Landing, M: Marketing, O: Other, P: Pricing, Lg: Legal, Rn: Release note, Ref: Reference, C:Concept). The small polar plots show the average dwell time in the previous three months. Note that the ranges of the axes of the plots vary. Bar charts below the polar plots show the proportions (%) of each group in the cluster.

# Appendix B

# Experiment Details for Predictive Synthesis of API-Centric Code

## B.1 Supported operations of PyTorch

Below is the list of 33 PyTorch operations. 16 operations used in the original dataset (described in Section 6.6.1) are highlighted.

- / /
- M v
- arange
- argmax
- # B M + Q m M i
- cdist
- div
- 2 [
- 2 t T M /
- eye
- gather
- ; i
- H i
- K b F 2 / n b 2 H 2 + i
- K i K m H
- max

- minimum
- K m H
- M 2
- one_hot
- repeat_interleave
- reshape
- roll
- searchsorted
- square
- squeeze
- b i + F
- sum
- i 2 M b Q ` / Q i
- tile
- i ` M b T Q b 2
- m M b [ m 2 2 x 2
- r ? 2 ` 2

## B.2 Stack Overflow Benchmarks

As mentioned in Section 6.6.1, we adapted the Stack Overflow benchmarks created for TF-Coder [229]. The examples were collected from Stack Overflow posts and the benchmarks were inspired by those posts. However, the input/output values were replaced by the TF-Coder authors for licensing reasons. The input/output values created by TF-Coder authors, and we updated some values to fit into our scope.

### B.2.1 Input/output and Desired Code

| | Inputs | Output | Code & Function Sequence | Expected Sequence |
|---|---|---|---|---|
| SO01 | in1 = [[5, 2], [1, 3], [0, -1]] | [[[5, 5], [1, 1], [0, 0]],<br>[[2, 2], [3, 3], [-1, -1]]] | torch.transpose(in1, expand((2, 3, 2)), 0, 2) | expand, transpose |
| SO02 | in1 = [5, 1, 0, 3, 0, 0, 2, 0, 2]<br>in2 = 1 | [1, 1, 0, 1, 0, 0, 1, 0, 1] | torch.where(torch.lt(in1, 1), in1, 1) | lt, where |
| SO05 | in1 = [[4, 3, 1], [6, 5, 2]]<br>in2 = [[[5, 5]], [[1, 5]], [[6, 0]]] | [[[29, 35]], [[47, 55]]] | torch.tensordot(in1, in2, 1) | tensordot |
| SO06 | in1 = [3, 5, 0, 2, 3, 3, 0] | [[1, 0, 0, 0, 1, 1, 0],<br>[0, 1, 0, 0, 0, 0, 0],<br>[0, 0, 1, 0, 0, 0, 1],<br>[0, 0, 0, 1, 0, 0, 0],<br>[1, 0, 0, 0, 1, 1, 0],<br>[1, 0, 0, 0, 1, 1, 0],<br>[0, 0, 1, 0, 0, 0, 1]] | torch.eq(in1, torch.unsqueeze(in1, 1)) | unsqueeze, eq |
| SO07 | in1 = [[[8, 4, 6], [2, 12, 3]],<br>[[11, 12, 5], [9, 12, 12]],<br>[[9, 2, 13], [7, 0, 7]],<br>[[2, 10, 5], [7, 1, 2]]], | [[[8, 4, 6], [11, 12, 5], [9, 2, 13], [2, 10, 5]],<br>[[2, 12, 3], [9, 12, 12], [7, 0, 7], [7, 1, 2]]] | torch.transpose(in1, 0, 1) | transpose |
| SO08 | in1 = [1, 0, 0, 2, 1, 3, 5, 0, 1, 2, 10],<br>in2 = [12, 3, 45, 6, 7, 8, 9, 87, 65, 4, 32],<br>in3 = 1 | [6, 8, 9, 4, 32] | torch.masked_select(in2, torch.gt(in1, 1)) | gt, masked_select |
| SO11 | in1 = [4, 0, 1, 1, 0, 4, 0, 0, 3, 4, 1] | [4, 3, 0, 1, 3] | torch.bincount(in1) | bincount |
| SO13 | in1 = [[3, 5], [10, 2]],<br>in2 = [[[1, 0], [5, 4]], [[3, 10], [2, 0]]] | [[[28, 20], [19, 30]], [[20, 8], [34, 100]]] | torch.transpose(torch.matmul(in1, in2), 0, 1) | matmul, transpose |
| SO14 | in1 = [[[0, 0, 1],<br>[0, 0, 0],<br>[1, 0, 1],<br>[0, 1, 0],<br>[0, 0, 0],<br>[1, 1, 1],<br>[1, 1, 0]]] | [[1, 0, 1, 1, 0, 1, 1]] | torch.any(in1, -1) | any |
| SO15 | in1 = [3, 1, 2, 0, 1, 0, 10, 1, 0] | 3, 0, 2, 0, 0, 0, 10, 0, 0] | torch.mul(in1, torch.ne(in1, 1)) | ne, mul |
| SO16 | in1 = [[2, 5], [3, 0], [8, 7]],<br>in2 = [4, 10, 6] | [[8, 20], [30, 0], [48, 42]] | torch.mul(in1, torch.unsqueeze(in2, 1)) | unsqueez, mul |
| SO17 | in1 = [17, 32, 99] | [[17, 17], [32, 32], [99, 99]] | torch.stack((in1, in1), 1) | stack |
| SO18 | in1 = [[[1, 1, 1], [1, 0, 1]],<br>[[1, 2, 3], [4, 5, 6]]],<br>in2 = [[1, 1, 1, 1], [1, 2, 3, 4], [5, 6, 7, 8]],<br>in3 = [100, 200, 300, 400], | [[[107, 209, 311, 413], [106, 207, 308, 409]],<br>[[118, 223, 328, 433], [139, 250, 361, 472]]] | torch.add(in3, torch.matmul(in1, in2)) | matmul, add |
| SO20 | in1 = [[7, 2, 1],<br>[4, 5, 1],<br>[4, 4, 2],<br>[3, 4, 3],<br>[0, 0, 1]] | [[1, 0, 0],<br>[0, 1, 0],<br>[1, 0, 0],<br>[0, 1, 0],<br>[0, 0, 1]] | torch.nn.functional.one_hot(torch.argmax(in1, 1), 3) | argmax, one_hot |
| SO21 | in1 = [[2], [0], [1], [0]],<br>in2 = [[2, 5, 3], [1, 3, 6], [1, 6, 3], [7, 0, 3]] | [[3], [1], [6], [7]] | torch.gather(in2, 1, in1) | gather |
| SO22 | in1 = [3, 1, 10],<br>in2 = [[6, 4], [5, 1], [3, 4]] | [53, 53] | torch.tensordot(in1, in2, 1) | tensordot |
| SO23 | in1 = [[0, 5, 2], [3, 1, 4], [5, 1, 5]], | [[1, 0, 1, 0, 0, 1, 0, 0, 0],<br>[0, 1, 0, 1, 1, 0, 0, 0, 0],<br>[0, 1, 0, 0, 0, 1, 0, 0, 0]] | torch.sum(torch.nn.functional.one_hot(in1, 9), 1) | one_hot, sum |

| | | | | |
|---|---|---|---|---|
| SO24 | in1 = [3, 1, 4, 5, 2, 8, 6, 7],<br>in2 = [1, 0, 2, 0, 1, 1, 0, 2],<br>in3 = 0 | [3, 1, 2, 5, 2, 8, 6, 3.5] | torch.where(torch.ne(in2, in3), torch.div(in1, in2), in1) | div, where |
| SO25 | in1 = 3<br>in2 = 4 | [[1, 0, 0],<br>[0, 1, 0],<br>[0, 0, 1],<br>[1, 0, 0],<br>[0, 1, 0],<br>[0, 0, 1],<br>[1, 0, 0],<br>[0, 1, 0],<br>[0, 0, 1],<br>[1, 0, 0],<br>[0, 1, 0],<br>[0, 0, 1]] | torch.tile(torch.eye(in1), (in2, 1)) | eye, tile |
| SO26 | in1 = [[[[3, 4], [1, 2]], [[5, 2], [10, 3]], [[10, 20], [4, 7]]]] | [10, 20, 41] | torch.sum(torch.flatten(in1, 1), 1) | sum, sum |
| SO27 | in1 = [0, 3, 5, 6],<br>in2 = 8 | [1, 0, 0, 1, 0, 1, 1, 0] | torch.sum(torch.nn.functional.one_hot(in1, in2), 0) | one_hot, sum |
| SO29 | in1 = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21],<br>in2 = [12, 0, 10, 23, 16], | [6, 0, 5, 11, 8] | torch.searchsorted(in1, in2) | searchsorted |
| SO30 | in1 = [[1., 2.], [3., 4.], [5., 6.]],<br>in2 = [[9., 4.], [8., 5.], [7., 6.]], | [[math.sqrt(68), math.sqrt(58), math.sqrt(52)],<br>[math.sqrt(36), math.sqrt(26), math.sqrt(20)],<br>[math.sqrt(20), math.sqrt(10), math.sqrt(4)]] | torch.cdist(in1, in2) | cdist |
| SO32 | [[1, 6, 2, 1], [3, 1, 4, 2], [2, 1, 2, 5]] | [1.3, 1.5, 2.0] | torch.tensordot(in1, torch.arange(4), 1) | arange, tensordot |
| SO34 | in1 = [[[1, 2], [3, 4]],<br>[[5, 6], [7, 8]],<br>[[10, 20], [30, 40]]],<br>in2 = [3, 5, 10], | [[128, 236], [344, 452]] | torch.tensordot(in2, in1, 1) | tensordot |
| SO36 | in1 = [1, 0, 1, 1, 0, 1, 0, 1], | [1., 0., 0.333333, 0.25, 0., 0.166667, 0., 0.125] | torch.div(in1, torch.add(in1, torch.arange(8))) | arange, add, div |
| SO37 | in1 = [[[[10, 20, 30], [40, 50, 60]],<br>[[12, 34, 56], [78, 98, 76]]]],<br>in2 = [5, 10, 20], | [[[850, 1900], [1520, 2890]]] | torch.tensordot(in1, in2, 1) | tensordot |
| SO39 | in1 = [[15, 10, 9, 20], [11, 0, 1, 9], [10, 1, 11, 25]] | [[225, 100, 81, 400],<br>[121, 0, 1, 81],<br>[100, 1, 121, 625]] | torch.square(torch.mul(in1, in1)) | mul, square |
| SO41 | in1 = [5, 2, 8, 2, 4, 1, 1, 0, 2, 1],<br>in2 = 3 | [5, 2, 8, 4, 1, 1, 0, 2, 1] | torch.masked_select(in1, torch.ne(torch.arange(10), in2)) | arange, ne, masked_select |
| SO42 | in1 = [4, 6, 2, 6, 7, 3, 3]<br>in2 = 7 | [0, 0, 0, 0, 1, 0, 0] | torch.eq(in1, 7) | eq |
| SO44 | in1 = [[3, 5, 2],<br>[6, 2, 3],<br>[8, 7, 1],<br>[0, 3, 5],<br>[4, 7, 3],<br>[2, 1, 6],<br>[10, 20, 30],<br>[4, 5, 6]] | [[9, 7, 5], [8, 19, 6], [6, 8, 9], [14, 25, 36]] | torch.sum(torch.reshape(in1, (-1, 2, 8)), 1) | reshape, sum |
| SO45 | in1 = [[[12, 34], [56, 78], [23, 54], [76, 78], [42, 24]]],<br>in2 = [1, 0, 1, 0, 1], | [[[34, 12], [56, 78], [54, 23], [76, 78], [24, 42]]] | torch.where(torch.unsqueeze(in2, 1), torch.roll(in1, 1, -1), in1) | roll, unsqueeze, where |
| SO46 | in1 = [3, 4, 1], | [0, 0, 0, 1, 1, 1, 1, 2] | torch.repeat_interleave(torch.arange(3), in1, 0) | arange, repeat_interleave |
| SO48 | in1 = [32, 53, 45, 38, 29, 89, 64, 23],<br>in2 = [38, 53, 89, 38, 32, 64], | [3, 1, 5, 3, 0, 6] | torch.argmax(torch.eq(in1, torch.unsqueeze(in2, 1)).float(), 1) | unsqueez, eq, argmax |

| | | | | |
|---|---|---|---|---|
| SO49 | in1 = [[[[1, 2, 3], [4, 5, 6]]],<br>[[[8, 10, 0], [6, 4, 2]]],<br>[[[9, 8, 7], [1, 2, 3]]],],<br>in2 = [20, 5, 10], | [[[[20, 40, 60], [80, 100, 120]]],<br>[[[40, 50, 0], [30, 20, 10]]],<br>[[[90, 80, 70], [10, 20, 30]]]] | torch.transpose(torch.mul(torch.transpose(in1, 0, 3), in2), 0, 3) | transpose, mul, transpose |
| SO50 | in1 = [3] | [[0, 0, 0, 1, 0, 0],<br>[0, 0, 0, 1, 0, 0],<br>[0, 0, 0, 1, 0, 0],<br>[0, 0, 0, 1, 0, 0],<br>[0, 0, 0, 1, 0, 0]] | torch.nn.functional.one_hot(in1.expand(in1,)) in2) | expand, one_hot |

### B.2.2 Links to Original StackOverflow Posts

- ?iiTb,ffbi +FQp2+ QHKQ[m2biBQMbf9y99R8yjfi2MbQ`7HQr@i2MbQ`@
- ?iiTb,ffbi +FQp2+ QHKQ[m2biBQMbf9e9y33jNfi2MbQ`7HQr@i`BK@p H
- ?iiTb,ffbi +FQp2+ QHKQ[m2biBQMbf9jyedjj3fi2MbQ`@KmHiBTHB+ iBQ
- ?iiTb,ffbi +FQp2+ QHKQ[m2biBQMbf9d3RekjRf+`2 i2@#BM `v@i2MbQ BM@i2MbQ`7HQr
- ?iiTb,ffbi +FQp2+ QHKQ[m2biBQMbfj3kRkky8fbr T@i2MbQ`@ t2b@BM
- ?iiTb,ffbi +FQp2+ QHKQ[m2biBQMbfjjdeNy9Rfi2MbQ`7HQr@BM/2tBM; i2MbQ`
- ?iiTb,ffbi +FQp2+ QHKQ[m2biBQMbf98RN9edkf?Qr@iQ@+QmMi@2H2 i2MbQ`
- ?iiTb,ffbi +FQp2+ QHKQ[m2biBQMbf8ydddd y9fM@/@i2MbQ`@K i`Bt@ rBi?@i2MbQ`7HQr
- ?iiTb,ffbi +FQp2+ QHKQ[m2biBQMbfj8e8dyyjf ;;`2; i2@2 +?@2H2K2M BM@i2MbQ`7HQr
- ?iiTb,ffbi +FQp2+ QHKQ[m2biBQMbfjNy98dNdf+QM/BiBQM H@ bbB;M p Hm2b@BM@i2MbQ`7HQr
- ?iiTb,ffbi +FQp2+ QHKQ[m2biBQMbf9ek9ye9efi2MbQ`@KmHiBTHv@ H i2MbQ`7HQr
- ?iiTb,ffbi +FQp2+ QHKQ[m2biBQMbf8RdeRj8jf #Qmi@i2MbQ`@Q7@i2
- ?iiTb,ffbi +FQp2+ QHKQ[m2biBQMbfj3kkkRkefi2MbQ`7HQr@277B+B2M KmHiBTHB+ iBQM
- ?iiTb,ffbi +FQp2+ QHKQ[m2biBQMbf993j9djNf `;K t@QM@ @i2MbQ`@ BM@i2MbQ`7HQr
- ?iiTb,ffbi +FQp2+ QHKQ[m2biBQMbf8ReNyyN8f?Qr@iQ@; i?2`@2H2K BM@i2MbQ`7HQr
- ?iiTb,ffbi +FQp2+ QHKQ[m2biBQMbf9jk393Ndf?Qr@+ M@B@KmHiBTH @K i`Bt@BM@i2MbQ`7HQr@rBi?Qmi@`2b? TBM;
- ?iiTb,ffbi +FQp2+ QHKQ[m2biBQMbf8j9R99jjfi2MbQ`7HQr@i2MbQ`@#
- ?iiTb,ffbi +FQp2+ QHKQ[m2biBQMbf8je9jjjNfi2MbQ`7HQr@Qp2``B/BM iQ@`2im`M@i?2@MmK2` iQ`@r?2M@/BpB/BM;@#v@y
- ?iiTb,ffbi +FQp2+ QHKQ[m2biBQMbf8jeykeNRf/mTHB+ i2@ @i2MbQ`@
- ?iiTb,ffbi +FQp2+ QHKQ[m2biBQMbf89kN9d3yf?Qr@iQ@T2`7Q`K@`2/ /BK2MbBQMb@ i@QM+2

- ?iiTb,ffbi +FQp2+ÖQHKQ[m2biBQMbf89kk8dy9f?Qr@/Q@B@;2i@ @i2M
i?2@QM@TQbBiBQMb@BM@i?2@Q`B;BM H@i2MbQ`
- ?iiTb,ffbi +FQp2+ÖQHKQ[m2biBQMbf89R88y38f#m+F2iBM;@+QMiBMQ
BM@i2MbQ`7HQr
- ?iiTb,ffbi +FQp2+ÖQHKQ[m2biBQMbf89R9dd3yfi2MbQ`7HQr@?Qr@iQ@
2m+HB/2 M@/Bbi M+2@#2ir22M@irQ@i2MbQ`
- ?iiTb,ffbi +FQp2+ÖQHKQ[m2biBQMbf93e8N99Nf?Qr@iQ@+QKTmi2@i?
Q7@ @i2MbQ`@BM@i2MbQ`7HQr
- ?iiTb,ffbi +FQp2+ÖQHKQ[m2biBQMbf9N8jkjdRf+QKTmi2@ @HBM2 `@+
i2MbQ`b@BM@i2MbQ`7HQr
- ?iiTb,ffbi +FQp2+ÖQHKQ[m2biBQMbf9jjyed33f/BpB/2@2H2K2Mib@Q7@
#v@i?2@+Q``BbTQM/2Mi@BM/2t
- ?iiTb,ffbi +FQp2+ÖQHKQ[m2biBQMbf9Nkyey8RfKmHiBTHv@9@/@i2Mb
- ?iiTb,ffbi +FQp2+ÖQHKQ[m2biBQMbfjdNRkReRf?Qr@+ M@B@+QKTmi2
+QM/BiBQM Hb@QM@# i+?2b@BM@i2MbQ`7HQr
- ?iiTb,ffbi +FQp2+ÖQHKQ[m2biBQMbf899NNy8Rf2H2; Mi@r v@iQ@ ++2
M/@i2MbQ`@BM@i2MbQ`7HQr
- ?iiTb,ffbi +FQp2+ÖQHKQ[m2biBQMbf899Nj3R9f#BM `v@p2+iQ`@Q7@K
- ?iiTb,ffbi +FQp2+ÖQHKQ[m2biBQMbf899ykj3NfbmK@i?2@+QHmKMb@
`Qrb@Q7@ @i2MbQ`@Q7@j@/BK2MbBQMb
- ?iiTb,ffbi +FQp2+ÖQHKQ[m2biBQMbf89jjdNk8f`2p2`b2@Q`/2`@Q7@bQ
BM@i2MbQ`7HQr
- ?iiTb,ffbi +FQp2+ÖQHKQ[m2biBQMbf83e8kReRf?Qr@iQ@+QMp2`i@k@
R@R@k@k@k@k@iQ@miBHBx2@i7@K i?@b2;K2Mi@bmK
- ?iiTb,ffbi +FQp2+ÖQHKQ[m2biBQMbf8393Rjjkf;2iiBM;@i?2@BM/B+2b@
2H2K2Mib@BM@ @i2MbQ`7HQr@ i@QM+2
- ?iiTb,ffbi +FQp2+ÖQHKQ[m2biBQMbf839ee8ekf;Bp2M@ @# i+?@Q7@
iQ@b+ H `@KmHiBTHv@2 +?@BK ;2@#v@ @/B772`2Mi@b+ H
- ?iiTb,ffbi +FQp2+ÖQHKQ[m2biBQMbf838jd9N8fi2MbQ`7HQr@BMBiB H
i2MbQ`@rBi?@QMHv@QM2@HBM2@+QHmKM@MQi@x2`Q

## B.3 Implementation of ML Models

We implement the model in Python using the PyTorch. We describe some implementation details here, but the code will be shared after anonymous period is over.

1 M + Q / B I M ; X Encoding of a tensor requires three encodings, separated by a separator: value, size, and type. The max sizes of the encodings are 150, 5, 3, respectively. Our approach supports up to 3 input tensors and one output tensor, and each tensor is separated by a separator, which makes the the size of input encoding to be 640 (4 (150 + 1 + 5 + 1 + 2 + 1) ).

* Q K T Q b B i B Q M H J Q / 2 H X For the embedding, we use the feed forward network, that is identical to the classification model, which is trained jointly with a bi-RNN model. The embedded input-output pair is passed to bi-RNN, having 1 hidden layer. To evaluate the model, we use a beam size of 3. We use the compositional model in two modes: "full sequence" mode, which returns the predicted API function sequence, and a "first-of-sequence" mode that returns only the first API function from the predicted sequence.

J m H i B @ H # 2 H * H b b B } + i B Q M i Q / 2 H X For the weight-enumerative search with prioritization (Figure 6.1 (b)), we trained a simple multi-label classification model following DeepCoder, instead of TF-Coder which uses manually defined features (e.g., whether a value is a primitive) which we found less generalizable. We used the same model architecture with the classification model, but changed the last activation function into b B ; K Q B / for the multi-label classification. We trained this model with input-output of b 2 [ m 2 M + A 2 b f A 2 b functions.

1 p H m i B Q M J 2 B + b X We evaluate our models on the accuracy of their predictions. For the model accuracy with synthetic data, we check whether the model correctly predict the ground-truth APIs, and for the Stack Overflow benchmarks evaluation, we measure the rank of the correct prediction and extract top-1, top-3 and top-10 accuracy metrics.

## B.4 Algorithms

**Algorithm 1** Weighted Enumerative Synthesis

A M T m Λi, task specification input/output, (I; O)
P m i T m Λ, program P such that P(I) = O

```
B   f   I; 0;   1; 1; :::g                                   . Base values
E     B                                                      . Pool of values
Ops       b b B ; M P 7 (I; *O Q) b i
 7 Q ` v H 2 H C  / Q
    v:cost       b b B ; M o (H v)* Q b i
 7 Q C = 1!     max_cost  / Q                                . Cost Budget
     7 Q ` op H 2 H Ops  / Q
        c     op:cost
        n     op:arity
        . Partition cost budgets into n arguments
         7 Q ` [c1; H H :; cn] 2 partition (C     c; n)  / Q
            7 Qi = 1; :::; n  / Q
               . Collect values satisfying i-th arg cost budget
               Ai   f   e 2 E je:cost = ci g
            7 Q ` arg H 2   i Ai  / Q
               V     Execute(op; args)                        . Run op w/ args
               B V 7 = O  i ? 2 M  ` 2 i m o: 2 M t T `
               B V 7 2 E  i ? 2 M
                  V:cost    C
                  E     E [ f  V g
 ` 2 i m "Fail: reached maximum cost"
```

**Algorithm 2** AssignOpCost

A M T m Λi, task specification input/output, (I; O)
P m i T m L, list of operations with costs Ops

```
 7 Q ` op H 2 H Ops  / Q
    op    preset_cost
B 7 ToModelPrioritization  i ? 2 M
    candidate_ops     J m H i B * H   b b B 7 B(H; Oi) B Q M J Q / 2 H
     7 Q ` op H 2 H +   M / B / Q 2 T / b Q
        op:cost    op:cost  reweight_multiplier
 ` 2 i m ` Ops
```

**Algorithm 3** Compositional Model - Full-Sequence

A M T m A i, task specification input/output, (I; O)
P m i T m A, program P such that P(I) = O

1: B f I; 0; 1; 1; :::g
2: op_seq P * Q K T Q b B i B Q(M, O) H J Q / 2 H
3: n op_i :arity
4: args_list n B
5: 7 Q ` arg H 2 args_list / Q
6: V Execute(op_seq; args)
7: B V = O i ? 2 M ` 2 i m o: 2 M t T `
8: ` 2 i m ”Fail” or b i ` i ”Enumerative Search”

**Algorithm 4** Compositional Model - First-Of-Sequence

A M T m A i, task specification input/output, (I; O)
P m i T m A, program P such that P(I) = O

1: B f I; 0; 1; 1; :::g
2: 7 Q i = 0! k / Q . Sequence of k operations
3: op_i CompositionalModel (V_{i 1}; I_i; O)
4: n op_i :arity
5: args_list n B
6: 7 Q ` arg H 2 args_list / Q
7: V_i Execute(op_i; args)
8: B V_i = O i ? 2 M ` 2 i m o: 2 M t T `
9: ` 2 i m ”Fail” or b i ` i ”Enumerative Search”

# Appendix C

# Additional Study Results for SOREL

## C.1 Outcome Variables for Quantitative Analysis.

Table C.1: Outcome variables for quantitative analysis.

| Variable | Details |
|---|---|
| **Task completion time** i B K 2 | We measured the time from when a participant first enters a search query to when they finish submitting the solution. |
| **Number of search queries** [ m 2 ` B 2 b | We counted the number of search queries participants wrote until they submitted the final answer, as an approximation of how easy it was to retrieve necessary information. |
| **Number of pages visited** T ; 2 b | We measured the number of web pages visited by a participant, to approximate the efforts needed in discovery. |
| **Prior knowledge of the task** T ` B Q ` | After each task, we asked participants whether they had implemented similar code in the past, and if so, whether that helped the search or not. We asked the level of prior knowledge in four levels: no experience (0), vaguely remembering that they have worked on similar tasks, but not enough to be helpful (1), experience with similar tasks helped them complete the task (2), and prior knowledge helped them recollect the exact function name (3). |
| **The correctness of the solutions given the task definition** Ù **correctness** | We tested whether a participant submits an API method that matches with our ideal solution, which we determined based on our expert knowledge of the tasks. The tasks were purposefully designed to have multiple potential solutions, with varying implication and trade-offs, to test whether participants are aware of the alternative solutions. Thus, "incorrect" solutions may not necessarily be "bad" solutions, and they can still meet the task requirements with different quality implications, such as performance, readability, or scalability. |

Table C.1: Outcome variables for quantitative analysis (cont.).

| Variable | Details |
|---|---|
| **Awareness of the comparable API methods** r `2M2bb | In each post-task interview, we tested whether a participant was aware of comparable API methods given the task context. When participants answered that they did not know other relevant API methods or admitted that they *guessed* the difference, we considered them to not be aware of the comparable methods. If they could elaborate on the differences, we considered that they were aware of the comparable API methods. |
| **Understanding of difference** mM/2`bi M/BM | In the post-task interviews, we also asked the participants to explain the differences between the comparable API methods. This is independent of the *awareness*, as one might already know about comparable API methods but not consider using them during the task. We measured the level of understanding with two levels: did not know the difference or guessed the difference (0), and describe at least one difference between the methods(1). |

## C.2 User Study Results

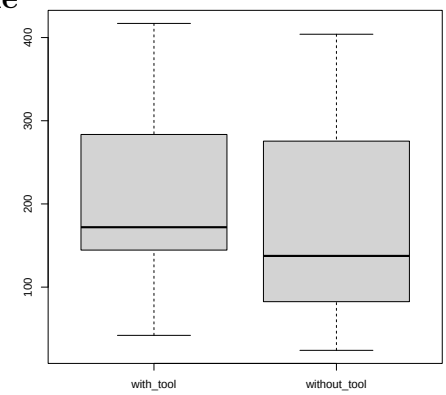
### C.2.1 Quantitative Data Summary

| Task ID | Our Ideal Solution | Comparable Methods | iBK2 | | T ;2b | | [m2`B2b | | +Q``2+iM2bb | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | Tool | No | Tool | No | Tool | No | Tool | No |
| 1 | b[m22x2 | `2b? T2 | 153.50 | 69.75 | 2.00 | 1.00 | 1.25 | 1.00 | 0.75 | 1.00 |
| 2 | 2BMbmK | (K iKmH- K iKmH) | 171.75 | 223.00 | 3.00 | 2.75 | 1.75 | 2.25 | 0.00 | 0.00 |
| 3 | +QM+ i | bi +F | 139.25 | 123.75 | 1.25 | 2.25 | 1.25 | 1.00 | 1.00 | 1.00 |
| 4 | #QQH2 MnK bF | (r?2`2- ; i?2`) | 282.00 | 120.75 | 2.50 | 2.00 | 1.25 | 2.00 | 1.00 | 0.75 |
| 5 | 7HQQ`/Bp | (/BpB/2- 7HQQ) | 224.50 | 106.00 | 2.00 | 2.50 | 1.50 | 1.25 | 1.00 | 1.00 |
| 6 | `2bBx2nrBi?nT / | `2bBx2 | 222.75 | 158.25 | 2.50 | 1.75 | 1.00 | 1.25 | 1.00 | 0.75 |
| 7 | +QMpR/ | +QMpk/f+QMpQ+mi BQM | 240.25 | 298.75 | 3.50 | 2.50 | 1.50 | 1.50 | 0.50 | 0.25 |
| 8 | bT `b2nbQ7iK tn+`Qbb@ n2Mi`QTvnrBi?nHQ;Bib | bQ7iK tn+`Qbb@ n2Mi`QTvnrBi?nHQ;Bib | 217.00 | 255.50 | 2.00 | 2.25 | 1.00 | 1.00 | 0.75 | 0.50 |
| Avg | | | 206.25 | 169.47 | 2.34 | 2.13 | 1.31 | 1.41 | 0.75 | 0.66 |
| SD | | | 95.53 | 105.40 | 1.47 | 1.31 | 0.69 | 0.87 | 0.44 | 0.48 |

| Task ID | Our Ideal Solution | Comparable Methods | r `2M2bb | | mM/2`bi M/BM; | |
|---|---|---|---|---|---|---|
| | | | Tool | No | Tool | No |
| 1 | b[m22x2 | `2b? T2 | 0.75 | 0.25 | 2.00 | 2.00 |
| 2 | 2BMbmK | (K iKmH- K iKmH) | 1.00 | 0.00 | 1.75 | 0.00 |
| 3 | +QM+ i | bi +F | 1.00 | 0.75 | 2.00 | 1.00 |
| 4 | #QQH2 MnK bF | (r?2`2- ; i?2`) | 0.75 | 0.00 | 0.75 | 0.50 |
| 5 | 7HQQ`/Bp | (/BpB/2- 7HQQ) | 1.00 | 0.50 | 1.75 | 1.50 |
| 6 | `2bBx2nrBi?nT / | `2bBx2 | 1.00 | 0.75 | 2.00 | 1.75 |
| 7 | +QMpR/ | +QMpk/f+QMpQ+mi BQM | 0.75 | 0.00 | 1.75 | 0.50 |
| 8 | bT `b2nbQ7iK tn+`Qbb@ n2Mi`QTvnrBi?nHQ;Bib | bQ7iK tn+`Qbb@ n2Mi`QTvnrBi?nHQ;Bib | 0.75 | 0.75 | 1.50 | 1.50 |
| Avg | | | 0.88 | 0.38 | 1.69 | 1.09 |
| SD | | | 0.34 | 0.49 | 0.64 | 0.96 |

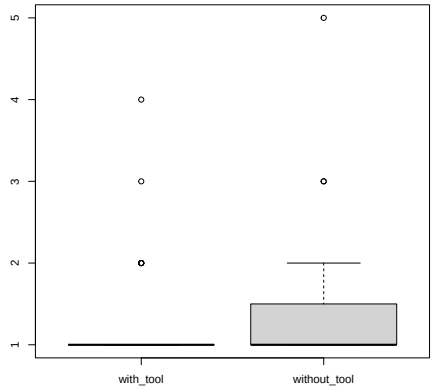
### C.2.2 Statistical Test Results

**Time**



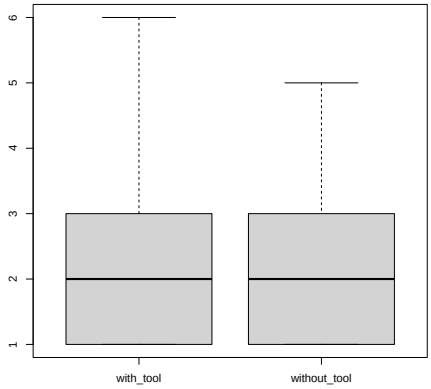

| | Coeffs (Errors) |
|---|---|
| (Intercept) | 198.69 (30.01)*** |
| prior | -19.08 (14.35) |
| tool | 37.97 (20.93) |
| AIC = 775.1; BIC = 788.1; LogLik = -381.5 | |
| $R2^m$ = 0.06. $R2^c$ = 0.30 | |

**Queries**



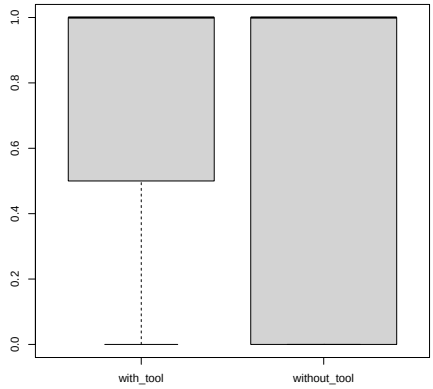

| | Coeffs (Errors) |
|---|---|
| (Intercept) | 1.41 (0.23)*** |
| prior | -0.01 (0.12) |
| tool | -0.09 (0.18) |
| AIC = 160.1; BIC = 173.0; LogLik = -74.0 | |
| $R2^m$ = 0.00. $R2^c$ = 0.14 | |

**Pages**



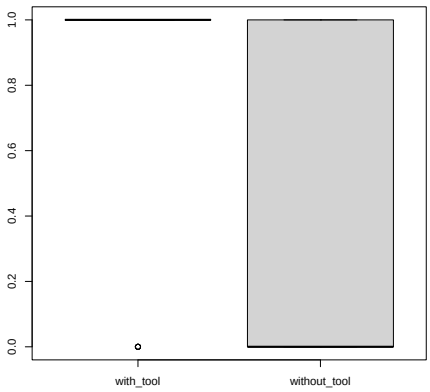

| | Coeffs (Errors) |
|---|---|
| (Intercept) | 2.36 (0.40)*** |
| prior | -0.15 (0.20) |
| tool | 0.23 (0.32) |
| AIC = 231.8; BIC = 244.8; LogLik = -109.9 | |
| $R2^m$ = 0.02. $R2^c$ = 0.14 | |

**Correctness**



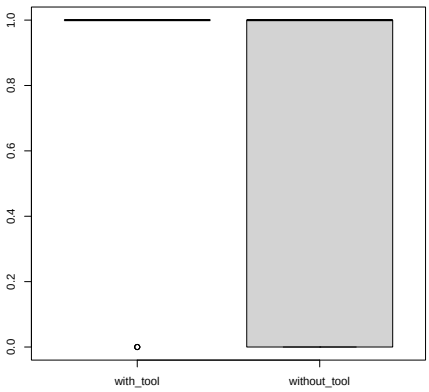

| | Coeffs (Errors) |
|---|---|
| (Intercept) | -0.22 (1.50) |
| prior | 0.96 (0.66) |
| tool | 0.88 (0.87) |
| AIC = 66.2; BIC = 77.0; LogLik = -28.1 | |
| $R2^m$ = 0.07. $R2^c$ = 0.74 | |

**Awareness**




| | Coeffs (Errors) |
|---|---|
| (Intercept) | -1.12 (0.92) |
| prior | 0.30 (0.44) |
| tool | 3.03 (0.95)** |
| AIC = 73.7; BIC = 84.5; LogLik = -31.9 | |
| $R2^m$ = 0.35. $R2^c$ = 0.53 | |

**Understanding**




| | Coeffs (Errors) |
|---|---|
| (Intercept) | -1.08 (1.10) |
| prior | 1.11 (0.58). |
| tool | 2.63 (0.95)** |
| AIC = 64.8; BIC = 75.6; LogLik = -27.4 | |
| $R2^m$ = 0.31. $R2^c$ = 0.63 | |

# Appendix D

# Experiment Details for GILT Study

Table D.1: Completion time (s) of each subtask.

| | **All** | | **Professionals** | | **Students** | |
|---|---|---|---|---|---|---|
| | GILT | Search | GILT | Search | GILT | Search |
| Bokeh-1 | 193.75 | 202.8 | 244.86 | 239.25 | 122.20 | 178.50 |
| Bokeh-2 | 352.47 | 321.63 | 263.00 | 429.50 | 486.67 | 213.75 |
| Bokeh-3 | 610.86 | 764.20 | 634.75 | 723.83 | 579.00 | 791.11 |
| Bokeh-4 | 295.63 | 156.11 | 351.75 | 127.80 | 239.50 | 191.50 |
| Open3d-1 | 483.47 | 503.80 | 458.38 | 663.00 | 505.78 | 265.00 |
| Open3d-2 | 371.53 | 603.80 | 307.29 | 474.38 | 427.75 | 751.71 |
| Open3d-3 | 408.83 | 326.67 | 454.67 | 267.67 | 363.00 | 444.67 |
| Open3d-4 | 329.00 | 260.00 | 351.33 | 520.00 | 262.00 | - |

Table D.2: Completion rates of each subtask.

| | **All** | | **Professionals** | | **Students** | |
|---|---|---|---|---|---|---|
| | GILT | Search | GILT | Search | GILT | Search |
| Bokeh-1 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| Bokeh-2 | 0.88 | 0.81 | 0.89 | 0.71 | 0.86 | 0.89 |
| Bokeh-3 | 0.38 | 0.38 | 0.44 | 0.43 | 0.29 | 0.33 |
| Bokeh-4 | 0.25 | 0.56 | 0.22 | 0.57 | 0.29 | 0.56 |
| Open3d-1 | 0.81 | 0.69 | 1.00 | 0.67 | 0.67 | 0.71 |
| Open3d-2 | 0.81 | 0.50 | 1.00 | 0.44 | 0.67 | 0.57 |
| Open3d-3 | 0.50 | 0.13 | 0.57 | 0.11 | 0.44 | 0.14 |
| Open3d-4 | 0.06 | 0.06 | 0.00 | 0.11 | 0.11 | - |

Table D.3: Understanding scores.

| | All | | Professionals | | Students | |
|---|---|---|---|---|---|---|
| | GILT | Search | GILT | Search | GILT | Search |
| Bokeh | 1.19 | 1.31 | 1.33 | 1.43 | 1.00 | 1.22 |
| Open3d | 0.75 | 0.125 | 1.43 | 0.11 | 0.22 | 1.14 |

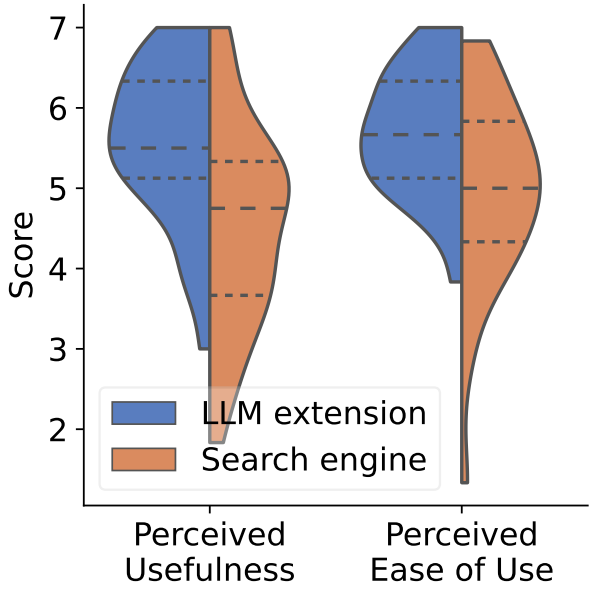



Figure D.1: Response to TAM items

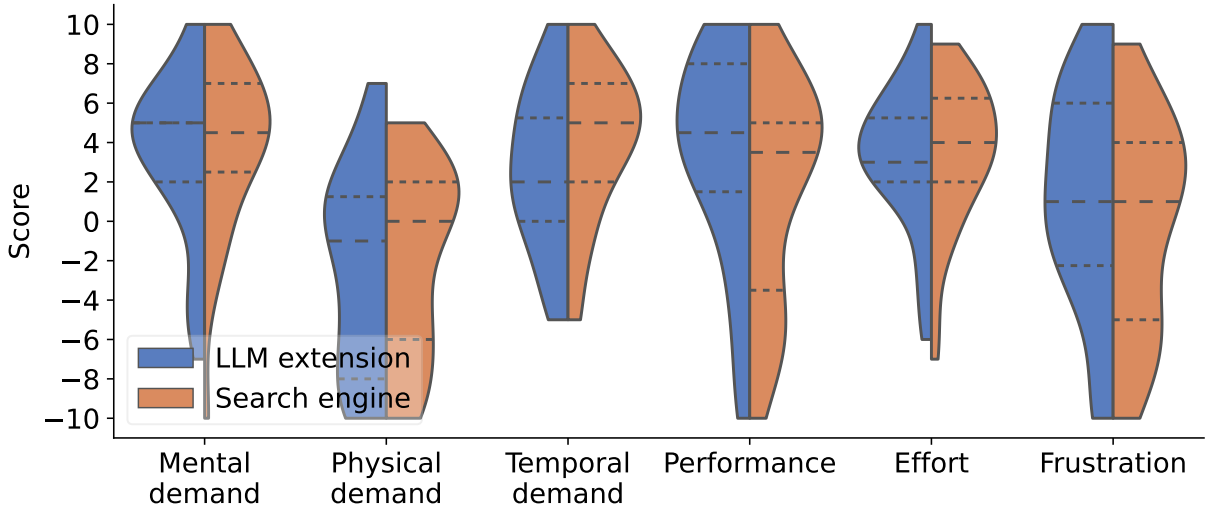



Figure D.2: Response to NASA TLX items

Table D.4: Participants' feedback codebook. The codes discussed in the paper are underlined. Initially, two authors separately performed open coding on the same set of 8 responses (25% of the entire data), and convened to discuss and merge the codes into a shared codebook. The first author coded the rest of the responses and discussed with the rest of the authors whenever new codes needed to be added.

| | *Q/2 .2b+`BTiBQM | _2T`2b2Mi iBp2 ZmQi2 |
|---|---|---|
| S`Qb | >_2/m+2/ BMBiB H TT`2?2MbBQM 2zQ@ The tool helps identify what to look for. | It was about to identify which sections of the code I need to modify much faster with the extension, even though I have no prior knowledge of 3D rendering. (P25) |
| | >*QMi2ti@ /Dmbi2/ `@bTQMb2 The tool provides code or explanation that is tailored to the user code. | Biggest difference using AI-based VS code extension is that the given solutions are specific to the code you are actually writing, whereas searching the web are usually more general solutions. (P32) |
| | >K x2K2M@ General positive reaction for a new tool. | I was surprised how good the explanations were, how tailored it was to the code, how easily it could make the changes that were required, (P12) |
| | >AMi2` +iB@pBiv Interactivity of the tool helps probing LLM. | The AI-based extension can be more interactive than search engines by maintaining conversation history. (P5) |
| | >G2bb +QMi2ti brB@i+?BM; The tool does not break the workflow and saves clicks as it is integrated in IDE. | ··· I can accomplish everything within the IDE without having to context switch to other resources (P3) |
| | >am{+B2Mi HB#` `v@/2i BHb The tool provides sufficient library information. | AI-based code extension was useful in a sense that it could also provide some details on concept, which could definitely help one to better tackle the task. (P28) |
| | >*QMi2ti BM+Q`T@` iBQM The tool provides the ability to easily incorporate the context | (LLM-based AI programming tool is) better (than search engines). Since I don't need to come up with the good searching words (P20) |
| | >o `vBM;;` MmH@`Biv The tool provides the ability to prompt about code at different granularity | Being able to change your level of insight was helpful (highlighting certain portions of code vs overall knowledge about the entire program (P15) |
| | >G2bb BM7Q`K iBQM@7Q` ;BM; The tool reduces the need for information foraging. | The main advantage for me is in circumventing the reading of the documentation. Official documentation is often lacking examples or short and concise explanations. For this reason, I often resort to stack overflow, in search of usage examples. (P24) |

Table D.4: Participants' feedback codebook. The codes discussed in the paper are underlined. Initially, two authors separately performed open coding on the same set of 8 responses (25% of the entire data), and convened to discuss and merge the codes into a shared codebook. The first author coded the rest of the responses and discussed with the rest of the authors whenever new codes needed to be added (cont.).

| | *Q/2 .2b+`BTiBQM | _2T`2b2Mi iBp2 ZmQi2 |
|---|---|---|
| JBt2/ | >_2[mB`2/ FMQrH2/;2 H2p2H 7Q` iQQH mi@HBiv<br>The tool is more useful when a user has / does not have existing knowledge. | It would be more useful when I learn a new library/PL. (P13)<br>AI based are more helpful when you already know what the outcome should be (P11) |
| | >Ib2` BMi2`@ +2<br>The user interface of the tool is usable / can be improved. | I liked the different buttons for different types of information so I didn't have to read a lot of text to find what I was looking for (P7)<br>A simpler view would be nice, I felt like there were too many features which could get some time to get used to. (P8) |
| | >SmHH@# b2/ BM@2` +iBQM<br>Pull-based interaction is useful / not useful. | GitHub Copilot's autocomplete is faster as it automatically suggests what I'm supposed to type next. But I can't actually ask co-pilot follow up questions, which I can do in the ai-based vs code extension, and that makes it really helpful. (P1) |
| *QMb | >LQ +Q``2+iM2bb ;m@` Mi2.2<br>The lack of trust and correctness guarantee of the outputs. | but there's always an underlying doubt in our mind regarding whether the information provided is 100% factual or useful. (P24) |
| | >S`QKTi /2T2M/2@M+v<br>The quality of the LLM-generated information depends on well-crafted prompts. | AI-based VS code extension was not able to give me the code that I was looking for, so it took up all my time (which I got very annoyed about). I think I just didn't word the question well. (P28) |
| | >AM7Q`K iBQM /B@p2`bBiv<br>The tool provides less diverse information compared to search engine results. | Stack overflow provides user historical comments and may be important for different approaches to the same problem. (P30) |

Table D.4: Participants' feedback codebook. The codes discussed in the paper are underlined. Initially, two authors separately performed open coding on the same set of 8 responses (25% of the entire data), and convened to discuss and merge the codes into a shared codebook. The first author coded the rest of the responses and discussed with the rest of the authors whenever new codes needed to be added (cont.).

| | *Q/2 .2b+`BTiBQM | _2T`2b2Mi iBp2 ZmQi2 |
|---|---|---|
| am;;2@ biBQMb | >Pi?2` `2bQm@+2b<br>Add (links to) other relevant resources like documentation and Stack Overflow posts. | I hope it can be combined with a documentation of the library, which makes it more accurate when providing descriptions of a method. (P21) |
| | >GGJ /2#m;;BM; + T @BHBiv<br>Ability to debug or auto-repair LLM outputs. | The ability to interactively debug incorrect AI-proposed solutions would be helpful. (P5) |
| | [GGJ T2`7Q`K] M+2<br>Use more powerful and faster underlying LLM. | They were both quite verbose even when a single line would've sufficed. (P24) |
| | >1tTH BM #@HBiv<br>Add ability to provide the rationale behind the outputs. | I think it may be better if we could understand the why aspect of the code. (P10) |

# Bibliography


[1] Bokeh. # Q m i , # H R M F. Retrieved: 2013-05-05. 7.3

[2] Chatgpt|openai. ? i i T b , f f + ? Q i T 2 M + Q K f. Retrieved: 2013-05-05. (document), 5.3, 5.4.4, 5.4.4, 7.1

[3] Github codespaces. ? i i T b , f f ; B i ? + m Q # K f 7 2 i m ` 2 b f + Q . / 2 b T + 2 b. Retrieved: 2013-05-05. 7.3

[4] Github copilot. ? i i T b , f f ; B i ? + m Q # K f 7 2 i m ` 2 b f + Q T B H Q ?. Retrieved: 2013-05-05. 2.4, 7.1

[5] Open3d – a modern library for 3d data processing. ? i i T , f f r r Q T 2 M Q / ; f. Retrieved: 2013-05-05. 7.3

[6] Tabnine: Ai assistant for software developers. ? i i T b , f f r : r # M B + M Q K f. Retrieved: 2013-05-05. 7.1

[7] Google privacy principles, 2023. URL ? i i T b , f f T Q H B ; + Q 2 ; b + 2 Q K f T ` B . p 3.2.3

[8] Google safety principles, 2023. URL ? i i T b , f f b 7 : 2 Q v Q ; H 2 f T ` B M + B 2 T H 2 b f. 3.2.3

[9] Google documentation style guide, 2023. URL ? i i T b , f f / 2 p 2 H : Q T 2 Q ` ; b + 2 Q K f b i . v H 2. 3.2.2

[10] Rabe Abdalkareem, Emad Shihab, and Juergen Rilling. What do developers use the crowd for? a study using stack overflow. A 1 1 1 a Q 7 , i 3 4 ( 2 ) : 2 5 3 – 6 0 , 2017. 2.1

[11] Mohamed Hussein Abdi, George Onyango Okeyo, and Ronald Waweru Mwangi. Matrix factorization techniques for context-aware collaborative filtering recommender systems: A survey. * Q K T m i X A M 7 1 X ( 2 ) : 1 – 1 0 , 2018. doi: 10:5539/cis:v11n2p1. URL ? i i T b , f f / Q : 2 Q ` ; f R 8 y 8 j N f + p B R M k 1 1 R, 3.5.3

[12] Deepak Agarwal, Bee-Chung Chen, Pradheep Elango, and Raghu Ramakrishnan. Content recommendation on web portals. * Q K K m M X , 5 6 ( 6 ) : 9 2 – 1 0 1 , 2013. doi: 10:1145/2461256:2461277. URL ? i i T b , f f / : Q B ; f R y R 9 8 f k 9 e k R 9 k 8 R k 1 1, 3.5.3

[13] Emad Aghajani. Context-Aware Software Documentation. k y R 3 A 1 1 1 A M i 2 ` M i B Q M H * Q 7 2 ` 2 M + 2 Q M a Q 7 i r ` 2 J B M i 2 M ( M + 2 a J ) M / 1 p 7 7 H 7 3 1 B Q M. pages 1–7, 2018. doi: 10:1109/icsme:2018:00090. 8.1

[14] Emad Aghajani, Csaba Nagy, Olga Lucero Vega-Márquez, Mario Linares-Vásquez, Laura Moreno, Gabriele Bavota, and Michele Lanza. Software documentation issues unveiled. In Joanne M. Atlee, Tevfik Bultan, and Jon Whittle, editors, S ` Q + 2 2 / B M ; b Q 7 i ? 2 9 R b A M i 2 ` M i B Q M H * Q M 7 2 ` 2 M + 2 Q M a Q 7 i r ` 2 1 M ; B M 2 2 ` B M ; - A * a 1 k y R J v k 8 @ j R -, pages 1199–1210. IEEE / ACM, 2019. doi: 10:1109/ICSE:2019:00122. URL ? i i T b , f f / : Q B ; f R y R y N f A k * y R y N y R . k 1 1, 8.1

[15] Wasi Uddin Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. Unified pre-training for program understanding and generation. pages 2655–2668, 2021. doi: 10:18653/v1/2021:naacl-main:211. URL ?iiTb,ff/:QB;fRy3e8jfpRfkyk RH@:kRBM2

[16] Miltiadis Allamanis and Charles Sutton. Why, when, and what: analyzing stack overflow questions by topic, type, and code. In S`Q+22/BM;bQ7i?2Ryi?qQ`FBM;*QM72`2 BM;aQ7ir`2_2 TQBMi;QBMi`B2 53–56. IEEE Press, 2013. ISBN 9781467329361. 2.1, 8.2.1

[17] Christoph Alt, Marc Hübner, and Leonhard Hennig. Fine-tuning pre-trained transformer language models to distantly supervised relation extraction. `sBp T`2T`BMi `sBp,RNye,2019. 2.3

[18] Matin Amoozadeh, David Daniels, Daye Nam, Stella Chen, Michael Hilton, Sruti Srinivasa Ragavan, and Mohammad Amin Alipour. Trust in generative ai among students: An exploratory study. In h?2h2+?MB+ HavKTQbBmKQM*QKT,2024. 1a+B2M+2 1/ 2.4

[19] AmuletxHeart. How to avoid writing boilerplate code in java swing mvc? ?iiTb, ffbi+FQp2`7HQ6f[m2biBQMbfkeR89kk8f?Qr@iQ@ pQB/@r`BiBM;@# +Q/2@BM@D p @2018 M2.2@ Kp+

[20] Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M. Dai, Anja Hauth, Katie Millican, David Silver, Slav Petrov, Melvin Johnson, Ioannis Antonoglou, Julian Schrittwieser, Amelia Glaese, Jilin Chen, Emily Pitler, Timothy P. Lillicrap, Angeliki Lazaridou, Orhan Firat, James Molloy, Michael Isard, Paul Ronald Barham, Tom Hennigan, Benjamin Lee, Fabio Viola, Malcolm Reynolds, Yuanzhong Xu, Ryan Doherty, Eli Collins, Clemens Meyer, Eliza Rutherford, Erica Moreira, Kareem Ayoub, Megha Goel, George Tucker, Enrique Piqueras, Maxim Krikun, Iain Barr, Nikolay Savinov, Ivo Danihelka, Becca Roelofs, Anaïs White, Anders Andreassen, Tamara von Glehn, Lakshman Yagati, Mehran Kazemi, Lucas Gonzalez, Misha Khalman, Jakub Sygnowski, and et al. Gemini: A family of highly capable multimodal models. *Q_,_abs/2312.11805, 2023. doi: 10:48550/ARXIV:2312:11805. URL ?iiTb, ff/Q:kQ`;fR9y388yf`:skBpkR R3y23

[21] Emily Judith Arteaga Garcia, João Felipe Nicolaci Pimentel, Zixuan Feng, Marco Gerosa, Igor Steinmacher, and Anita Sarma. How to support ml end-user programmers through a conversational agent. In S`Q+22/BM;bQ7i?29ei?A111f*JAMi2`M iBQM H* aQ7ir`2 1M;BM22`2 M, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400702174. doi: 10:1145/3597503:3608130. URL ?iiTb,ff/:QB;f R:yRR98fj8Nje8yy3Rj9y2

[22] Jacob Austin, Augustus Odena, Maxwell I. Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. Program synthesis with large language models. *Q_,_abs/2108.07732, 2021. URL ?iiTb, ff`tB:pQ`;f #bfk:Ryd3ij9.2

[23] Shams Azad, Peter C Rigby, and Latifa Guerrouj. Generating api call rules from version history and stack overflow posts. *Jh`Mb+iBQMbQM aQ7ir`2 1M;BM22`B J2i?Q/Q(HQP;a/1), 25(4):1–22, 2017. 2.4

[24] Matej Balog, Alexander L. Gaunt, Marc Brockschmidt, Sebastian Nowozin, and Daniel Tarlow. Deepcoder: Learning to write programs. 2017. URL ?iiTb,ffQT2M`2MpB2r

7 Q `?BK+ "v H / G `.[ (document), 6.1, 6.1, 6.1, 6.1, 6.2, 6.5, 6.6.3

[25] Matej Balog, Rishabh Singh, Petros Maniatis, and Charles Sutton. Neural program synthesis with a differentiable fixer. *Q _, _abs/2006.10924, 2020. URL ?iiTb,ff `tKQp`;f #bfky.yRey N.k92

[26] Shraddha Barke, Michael B. James, and Nadia Polikarpova. Grounded copilot: How programmers interact with code-generating models. S`Q+X *J S`Q;` K,X(OOPSLA1), apr 2023. doi: 10:1145/3586030. URL ?iiTb,ff/:QB;fRRyR98fj83.e24jy

[27] Anton Barua, Stephen W Thomas, and Ahmed E Hassan. What are developers talking about? an analysis of topics and trends in stack overflow. 1KTB`B+ H aQ7ir `2,1M;BM22` 19(3):619–654, 2014. 2.1

[28] Rohan Bavishi, Caroline Lemieux, Roy Fox, Koushik Sen, and Ion Stoica. Autopandas: neural-backed generators for program synthesis. S`Q+X *J S`Q;` K,X(OOPSLA): 168:1–168:27, 2019. doi: 10:1145/3360594. URL ?iiTb,ff/:QB;fRRyR98fjjeyk8N 6.2, 6.8

[29] Laura Beckwith, Cory Kissinger, Margaret Burnett, Susan Wiedenbeck, Joseph Lawrance, Alan Blackwell, and Curtis Cook. Tinkering and gender in end-user programmers' debugging. In S`Q+22/BM;b Q7 i?2 aA:*>A +QM72`2M+2 QM >mK M 6 +iQ`b B pages 231–240, 2006. 7.5.3

[30] Stefanie Beyer and Martin Pinzger. A manual categorization of android app development issues on stack overflow. In kyR9 A111 AMi2`M iBQM H *QM72`2M+2 QM aQ7ir M/ 1pQH, pages 531–535, 2014. doi: 10:1109/ICSME:2014:88. 2.1

[31] Stefanie Beyer, Christian Macho, Martin Pinzger, and Massimiliano Di Penta. Automatically classifying posts into question categories on stack overflow. In S`Q+22/BM;b Q7 i?2 kei? *QM72`2M+2 QM S`Q;`, KCPC'18, pages 211–221, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450357142. doi: 10:1145/3196321:3196333. URL ?iiTb,ff/:QB;fRRyR98fjRNjeRNkjj 2.1, 8.2.1, 8.2.2

[32] Joshua Bloch. How to design a good API and why it matters. In *QKT MBQM iQ *QM72`2M+ QM P#D2+i@Q`B2Mi2/ S`Q;` KKBM; avbi2Kb- G M, pages 506–507. M/ TTHE ACM, 2006. 4.1, 18, 4.1, 4.2, 4.2.1, 4.2.2

[33] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. CQm`M H Q7 ai iBbiB+ H J2+? MB+b, h 1tT2`BK, 2008(10):P10008, 2008. 4.3.3

[34] Joel Brandt, Philip J. Guo, Joel Lewenstein, Mira Dontcheva, and Scott R. Klemmer. Two studies of opportunistic programming: interleaving web foraging, learning, and writing code. In Dan R. Olsen Jr., Richard B. Arthur, Ken Hinckley, Meredith Ringel Morris, Scott E. Hudson, and Saul Greenberg, editors, S`Q+22/BM;b Q7 i?2 kdi? AMi2`M *QM72`2M+2 QM >mK M 6 +iQ`b BM *QKTmiBM; avbi2Kb- *>A kyyN T`BH 9@N, pages 1589–1598. ACM, 2009. doi: 10:1145/1518701:1518944. URL ?iiTb,ff/:QB;fRRyR98fR8RR38fyR3RN39, 3.1, 3, 3.4.2, 3, 3.4.3

[35] Joel Brandt, Mira Dontcheva, Marcos Weskamp, and Scott R. Klemmer. Example-centric programming: integrating web search into the development environment. In S`Q+22/BM;b Q7 i?2 k3i? AMi2`M iBQM H *QM72`2M+2 QM >mK M 6 (*>iQ`kyyB) M *QKTm

iH Mi - :2Q`;B - la - T`BH, pages 513–522, New York, NY, 2010. ACM. doi: 10:1145/1753326:1753402. URL ?iiTb,ff/:QB;fRRyR98fRd8Rjk8ej9y.k 7.7

[36] Broadcom. Spring framework. ?iiTb,ffbT`BQ. 4.2

[37] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. `sBp, 2020. doi: 10:48550/arxiv:2005:14165. 2.3

[38] Rudy Bunel, Matthew J. Hausknecht, Jacob Devlin, Rishabh Singh, and Pushmeet Kohli. Leveraging grammar and reinforcement learning for neural program synthesis. 2018. URL ?iiTb,ffQT2M`2Mp2Bf27rQ?`Bm#K>Rsrekf. 6.2

[39] Margaret M. Burnett, Simone Stumpf, Jamie Macbeth, Stephann Makri, Laura Beckwith, Irwin Kwan, Anicia Peters, and Will Jernigan. Gendermag: A method for evaluating software's gender inclusiveness. AMi2`+iX*,28(7):760–787, 2016. doi: 10:1093/IWC/IWV046. URL ?iiTb,ff/:QB;fRRyNjfBr+fBr7.y.9 7.5.9

[40] Open Source by greenrobot Logo Open Source by greenrobot Logo. Greendao. ?iiT,ff;`22M`Q#Q;if;`22M/ Qf/Q+mK2Mi iBQM,fBMi`Qt/m+i2BQM, greenrobot/m+i2BQM. 4.2

[41] Marc Carpentier, Christophe Combescure, Laura Merlini, and Thomas V Perneger. Kappa statistic to measure agreement beyond chance in free-response assessments. "J* K2/B+ H `2b2 `+? K2i?,27(Q:H–Q, 2017. 5.2

[42] Preetha Chatterjee, Kostadin Damevski, and Lori Pollock. Automatic extraction of opinion-based q&a from online developer chats. In AMi2`M iBQM H*QM72`2M+2 QM aQ7ir `BM, A*a), pages 1260–1272. IEEE, 2021. 2.4

[43] Jie-Cherng Chen and Sun-Jen Huang. An empirical analysis of the impact of software development problem factors on software maintainability. CX avbiX, 82(7):981–992, 2009. doi: 10:1016/J:JSS:2008:12:036. URL ?iiTb,ff/:QB;fRRyRedDkyy.8:kyje 3.1

[44] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harrison Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *Q_, abs/2107.03374, 2021. URL ?iiTb,ff `tKQp`;f #bfk:Ryjd. 1.1.2, 9.2

[45] Xinyun Chen, Chang Liu, and Dawn Song. Execution-guided neural program synthesis. 2019. URL ?iiTb,ffQT2M`2MpBf7rQ?Bm#K>R;7PB.5i2K

[46] Xinyun Chen, Dawn Song, and Yuandong Tian. Latent execution for neural program synthesis beyond domain-specific languages. pages 22196–22208, 2021. URL ?iiTb, ffT`Q+22/BM2;mb`:B+bT T2`fkykRf? b?f# j+N8+kNek/j #k7e2eedNjk/ #bi` +?iKH6.2

[47] Robert F. Chew, John Bollenbacher, Michael Wenger, Jessica Speer, and Annice Kim. Llm-assisted content analysis: Using large language models to support deductive coding. *Q_, abs/2306.14924, 2023. doi: 10:48550/ARXIV:2306:14924. URL ?iiTb,ff/:QB;f R:y9388yf `:kBypeR9Nk83.1

[48] Parmit K. Chilana, Amy J. Ko, and Jacob O. Wobbrock. Lemonaid: selection-based crowdsourced contextual help for web applications. pages 1549–1558, 2012. doi: 10:1145/ 2207676:2208620. URL ?iiTb,ff/:QB;fRRyR98fkkyykkyd3ek8.1

[49] Steven Clarke. What is an end user software engineer? In Margaret M. Burnett, Gregor Engels, Brad A. Myers, and Gregg Rothermel, editors, 1M/@lb2`aQ7ir `21M;BM22`BM;-F @kjXykXkkyd07081 of . ;bim?Ha2KBM `S`Q+22/BM Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2007. URL ?iiT,ff/`Q/Tbbim:?H2fQTmbfpQHHi2ti2fkyy, 3.4.1, 3.5.5

[50] Victoria Clarke and Virginia Braun. Teaching thematic analysis: Overcoming challenges and developing strategies for effective learning. h?2 Tbv+?,26(2):113–123, 2013. 7.6.2

[51] Wikipedia contributors. Boilerplate code definition of wikipedia. ?iiTb,ff 2MBFBT2QB;frBFBf"QBH2`THi2+Q4.1, 4.2.2, 4.2.2

[52] Carlos J Costa, Manuela Aparicio, and Robert Pierce. Evaluating information sources for computer programming learning and problem solving. In S`Q+22/BM;bQ7i?2Ni?qa1a AMi2`M iBQM H*QM72`2M+2QM SSGA1, pages 218–223, 2009. 3.1, 3.4.1

[53] Bill Curtis and Jakob Nielsen. Applying discount usability engineering. A111aQ7 (1):98–100, 1995. doi: 10:1109/52:363161. URL ?iiTb,ff/:QB;fRRyRyN:fe8kRe3.1

[54] d370urn3ur. Reduce boilerplate for subclasses issue #172 parse-community/Parse-SDK-Android. ?iiTb,ff;Bi:?+mQ#KfT `b2@+QKKmMBivfS `b2@a.E@4.1, 4.2

[55] Barthélémy Dagenais and Martin P Robillard. Creating and evolving developer documentation: understanding the decisions of open source contributors. /HX +KXQ`, page 127, 2010. doi: 10:1145/1882291:1882312. 8.1

[56] William K Darley and Robert E Smith. Gender differences in information processing strategies: An empirical test of the selectivity model in advertising response. CQm`M HQ7 /p2`iB,24(1):41–56, 1995. 7.5.3

[57] DatabaseGroup. Ap-ted implementation. ?iiTb,ff;Bi:?+mQ#Kf.i#b2:`Qm.Tf Ti2/ 4.4.1

[58] Matthew C Davis, Emad Aghayi, Thomas D LaToza, Xiaoyin Wang, Brad A Myers, and Joshua Sunshine. What's (not) working in programmer user studies? *Jh`Mb+iBQMb QMaQ7ir `21M;BM22`BM;, 2022. J7i?Q/QHQ;v

[59] Paul Denny, Viraj Kumar, and Nasser Giacaman. Conversing with copilot: Exploring prompt engineering for solving CS1 problems using natural language. In S`Q+22/@ BM;b Q7 i?2 89i? *J h2+?MB+ H avKTQbBmK QM *QKTmi2` a+B2M+ (aA:*a1 ky,kjQ`QMiQ- PL- * M / - J `+?,R8@ R36- 1k2k, pages 1–7. New York, NY, USA, 2023. ACM. doi: 10:1145/3545945:3569823. URL ?iiTb,ff/:QB;fRyR98f j898Nj98e N3k7j. 2.2, 7.4.1, 8.1

[60] Jacob Devlin, Jonathan Uesato, Surya Bhupatiraju, Rishabh Singh, Abdel-rahman Mohamed, and Pushmeet Kohli. Robustfill: Neural program learning under noisy I/O. 70: 990–998, 2017. URL ?iiT,ffT`Q+22:/KBHT;b2bbfpdyf/2pH-3KMR2d

[61] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. `sBp T`2T`BMi `sBp,R3Ry, 2019. 5, 8

[62] Ekwa Duala-Ekoko and Martin P Robillard. The information gathering strategies of api learners. Technical report, Technical report, TR-2010.6, School of Computer Science, McGill University, 2010. 7.2.2

[63] Ekwa Duala-Ekoko and Martin P Robillard. Using structure-based recommendations to facilitate discoverability in apis. In 1m`QT2 M *QM72`2M+2 QM P#D2+i,@Q`B2Mi2 pages 79–104. Springer, 2011. 5, 5.1

[64] Ekwa Duala-Ekoko and Martin P Robillard. Asking and answering questions about unfamiliar apis: An exploratory study. In kyRk j9i? AMi2`M iBQM H *QM72`2M+2 Q 1M;BM2 2A BM1, pages 266–276. IEEE, 2012. 3.1, 3.1, 7.2.2

[65] Ashish Dwivedi, Steve Clarke, Brad A. Myers, Sae Young Jeong, Yingyu Xie, Jack Beaton, Jeff Stylos, Ralf Ehret, Jan Karstens, Arkin Efeoglu, and Daniela K. Busse. End-User Computing, Development, and Software Engineering. pages 81–102, 2012. doi: 10:4018/978-1-4666-0140-6:ch004. 8.1

[66] Ralph H. Earle, Mark A. Rosso, and Kathryn E. Alexander. User preferences of software documentation genres. In Kathie Gossett, Angie Mallory, and Dawn M. Armfield, editors, S`Q+22/BM;b Q7 i?2 jj`/ MMm H AMi2`M iBQM H *QM72`2M+2 QM i?2 aA:.P* kyR8- GBK2`B+F- A`2H M/,-Gm 4k1–R6 @ ACM, 2015. doi: 10:1145/2775441:2775457. URL ?iiTb,ff/:QB;fRyR98fkddk8998 2.2, 3.1, 3.2.2, 3.4, 3.4.1, 1, 3.5.1

[67] Michael Ekstrand, Wei Li, Tovi Grossman, Justin Matejka, and George Fitzmaurice. Searching for software learning resources using application context. S`Q+22/BM;b Q7 i?2 k9i? MMm H *J bvKTQbBmK QM lb2` BMi2`7 +2 bQ7 R`2 M/ i2+?MQ 195, 2011. doi: 10:1145/2047196:2047220. 8.1

[68] Kevin Ellis, Maxwell Nye, Yewen Pu, Felix Sosa, Joshua B. Tenenbaum, and Armando Solar-Lezama. Write, execute, assess: Program synthesis with a repl. 2019. 6.2

[69] Jean-Rémy Falleri, Floréal Morandat, Xavier Blanc, Matias Martinez, and Martin Monperrus. Fine-grained and accurate source code differencing. In AMi2`M iBQM H *QM72`2M QM miQK i2/ aQ7ir `2 1M;BM22 BM1, pages 313–324. ACM, 2014. 4.3.3

[70] Umer Farooq and Dieter Zirkler. API peer reviews: A method for evaluating usability of application programming interfaces. In *QM72`2M+2 QM *QKTmi2` amTTQ`i2/ *Q qQ`F, pages 207–210. ACM, 2010. 4.1

[71] Yu Feng, Ruben Martins, Jacob Van Geffen, Isil Dillig, and Swarat Chaudhuri. Component-based synthesis of table consolidation and transformation tasks from examples. pages 422–436, 2017. doi: 10:1145/3062341:3062351. URL ?iiTb,ff/:QB;fRRyR98f jyekj9jRyekj8kR2

[72] Yu Feng, Ruben Martins, Osbert Bastani, and Isil Dillig. Program synthesis using conflict-driven learning. *J aA:SG L LQi, 53(4):420–435, 2018. 6.2

[73] James Finnie-Ansley, Paul Denny, Brett A. Becker, Andrew Luxton-Reilly, and James Prather. The robots are coming: Exploring the implications of openai codex on introductory programming. In mbi` H bB M *QKTmiBM; 1/m+( *BQ/kQBMi?2`BM1p2Mi- mbi` HB - 62#`m `vR9@ R3 19. ACM, pages 10–19. ACM, 2022. doi: 10:1145/3511861:3511863. URL ?iiTb,ff/:QB;fRRyR98fj8RJ83ReRR32.4

[74] Felix Fischer, Konstantin Böttinger, Huang Xiao, Christian Stransky, Yasemin Acar, Michael Backes, and Sascha Fahl. Stack overflow considered harmful? the impact of copy&paste on android application security. In kyRd A111 avKTQbBmK QM a2+m`Bi S`Bp(+aS, pages 121–136. IEEE, 2017. 5.5

[75] Scott D. Fleming, Chris Scaffidi, David Piorkowski, Margaret Burnett, Rachel Bellamy, Joseph Lawrance, and Irwin Kwan. An Information Foraging Theory Perspective on Tools for Debugging, Refactoring, and Reuse Tasks. *J h` Mb +iBQMb QM aQ7i@ r`2 1M;BM22`BM; M/ J2 P?aQ1/Q 22(2):14, 2013. ISSN 1049-331X. doi: 10:1145/2430545:2430551. 2.2

[76] Andrew Forward and Timothy Lethbridge. The relevance of software documentation, tools and technologies: a survey. In S`Q+22/BM;b Q7 i?2 kyyk *J avKTQbBmK QM 1M;BM22`BM;- J+G2 M- oB`;BMB - la, pages 26–33. ACM, 2002. doi: 10:1145/585058:585065. URL ?iiTb,ff/:QB;fRRyR98f83:8y8Ry.8

[77] Jaroslav Fowkes and Charles Sutton. Parameter-free probabilistic API mining across GitHub. In AMi2`M iBQM H avKTQbBmK QM 6QmM/ iB, pages 245–265. ACM, 2016. 4.1, 4.2.2, 4.3.1, 4.3.1, 4.4

[78] Steve Fox, Kuldeep Karnawat, Mark Mydland, Susan T. Dumais, and Thomas White. Evaluating implicit measures to improve web search. *J h` MbX AM7X(2):147–168, 2005. doi: 10:1145/1059981:1059982. URL ?iiTb,ff/:QB;fRRyR98fRy8RNy83RN3k 3.2.2

[79] Luanne Freund. Contextualizing the information-seeking behavior of software engineers: Contextualizing the Information-Seeking Behavior of Software Engineers. CQm`M H Q7 i?2 bbQ+B iBQM 7Q` AM7Q`K iBQM a, 63(2):1594–1605, 2012. ISSN 2330-1635. doi: 10:1002/asi:23278. 1.1.1, 3.1, 3.4.1, 1, 8.2.1, 8.2.2

[80] Daniel Fried, Armen Aghajanyan, Jessy Lin, Sida Wang, Eric Wallace, Freda Shi, Ruiqi Zhong, Wen-tau Yih, Luke Zettlemoyer, and Mike Lewis. InCoder: A Generative Model for Code Infilling and Synthesis. `sB, 2022. 8.1

[81] Golara Garousi, Vahid Garousi-Yusifoglu, Günther Ruhe, Junji Zhi, Mahmood Moussavi, and Brian Smith. Usage and usefulness of technical software documentation: An industrial case study. AM7X aQ7irX, 57:64–82, 2015. doi: 10:1016/J:INFSOF:2014:08:003. URL ?iiTb,ff/:QB;fRRyReBDM7bkQ7Ry3yy.j 3.1

[82] M. Rami Ghorab, Dong Zhou, Alexander O'Connor, and Vincent Wade. Personalised Information Retrieval: survey and classification. lb2` JQ/2HBM; M/ lb2`@ / Tj2/ AMi2` +, 23(4):381–443, 2013. ISSN 0924-1868. doi: 10:1007/s11257-012-9124-1. 8.1

[83] GitHub. Github copilot x: The ai-powered developer experience. ?iiTb,ff;Bi?+mQ#Kf 72 im`2bfT`2pB2rf+, 2023. Retrieved: 2013-05-05. 7.8

[84] Elena L Glassman, Tianyi Zhang, Björn Hartmann, and Miryung Kim. Visualizing API usage examples at scale. In >mK M 6 +iQ`b BM *QKTmiBM, pages 580:1–580:12. ACM, 2018. 4.1

[85] Victor M González and Gloria Mark. "Constant, constant, multi-tasking craziness": managing multiple working spheres. S`Q+22/BM;b Q7 i?2 kyy9 +QM72`2M+2 QM >mK +QKTmiBM; bvbi2yk9b@ 113–120, 2004. doi: 10:1145/985692:985707. 8.4.2

[86] Google. Android API 26 release note. ?iiTb,ff/2p2HQM/2QBQKf #QmifP2`bBQMbf Q`2Qf M/`QyB@@ 3 M;2bO7p#B@4b42;M im`2

[87] Georgios Gousios and Diomidis Spinellis. GHTorrent: GitHub's data from a firehose. In AMi2`M iBQ MH *QM72`2M+2 QM JBMBM, pages 12–21. IEEE, 2012. 4.1

[88] Valentina Grigoreanu, Margaret M. Burnett, and George G. Robertson. A strategy-centric approach to the design of end-user debugging tools. In S`Q+22/BM;b Q7 i?2 k3i? AMi2`M *QM72`2M+2 QM >mK M 6 +iQ`b BM(*QKTmiBM, pages 713–722. ACM, 2010. doi: 10:1145/1753326:1753431. URL ?iiTb,ff/QB`;fRyR98fRd8Rjk8ej9j7R. 5.3

[89] Tovi Grossman, Justin Matejka, and George W. Fitzmaurice. Chronicle: capture, exploration, and playback of document workflow histories. In Ken Perlin, Mary Czerwinski, and Rob Miller, editors, S`Q+22/BM;b Q7 i?2 kj`/ MMm H *JavKTQbBmK QM l aQ7ir `2 M/ h2+?MQHQ;v- L2r uQ`F- Lu- la, pages 143–152. ACM, 2010. doi: 10:1145/1866029:1866054. URL ?iiTb,ff/QB;fRyR98fR3eRyk9 2.1

[90] Sumit Gulwani. Automating string processing in spreadsheets using input-output examples. pages 317–330, 2011. doi: 10:1145/1926385:1926423. URL ?iiTb,ff/QB;fRyR98f RNke;jR8Nke.9k2j

[91] h5bp. html5-boilerplate. ?iiTb,ff;Bi?+mQ#Kf?8#Tf?iKH8@#.QB.H2`TH i2

[92] Md Montaser Hamid, Amreeta Chatterjee, Mariam Guizani, Andrew Anderson, Fatima Moussaoui, Sarah Yang, I Escobar, Anita Sarma, and Margaret Burnett. How to measure diversity actionably in technology. 1[mBiv-.Bp2`bBiv- M/ AM+HmbBQM BM aQ7i BM;, "2bi S` +iB+2b, 2023. AMbB;?ib

[93] Xiaoyu Han and Lei Wang. A novel document-level relation extraction method based on bert and entity information. A111 ++2bb, 2020. 5.4.7

[94] Steven Hardy. Automatic induction of lisp functions. In S`Q+22/BM;b Q7 i?2 Rbi amKK *QM72`2M+2 QM `iB}+B H AMi2HHB;2M+2, AMB'74, pages 50–62, NLD, 1974. IOS Press. 6.2

[95] Sandra G Hart and Lowell E Staveland. Development of nasa-tlx (task load index): Results of empirical and theoretical research. In /p M+2b BM Tbv-?QM Qay, volume 52, pages 139–183. Elsevier, 1988. 7.3, 7.6.1

[96] Arto Hellas, Juho Leinonen, Sami Sarsa, Charles Koutcheme, Lilja Kujanpää, and Juha Sorva. Exploring the responses of large language models to beginner programmers' help requests. In S`Q+22/BM;b Q7 i?2 kykj *J *QM72`2M+2 QM AMi2`M iBQM iBQM _2b2 `+A?1b_XRyk. ACM, August 2023. doi: 10:1145/3568813:3600139. URL ?iiT,ff/1Q:BQ`;fRRyR98fj8e:3e3RyjRj2N. 4

[97] Abram Hindle, Earl T. Barr, Mark Gabel, Zhendong Su, and Premkumar T. Devanbu. On the naturalness of software. *QKKmMX, 59(5):122–131, 2016. doi: 10:1145/2902362. URL ?iiTb,ff/:QB;fRRyR98fkNykk3ek

[98] Klaus Hinkelmann and Oscar Kempthorne. .2bB;M M/ M HvbBb Q7 2tT2`BK2Mib- p AMi`Q/m+iBQM iQ 2tT2`BK2M. John Wiley & Sons, 2007. 5.3.1

[99] Sture Holm. A simple sequentially rejective multiple test procedure. a+ M/BM pB M CQm`M H Q7 ai iB, 6(2):65–70, 1979. ISSN 03036898, 14679469. URL ?iiT,ffrrDbiQQ`;f bi #H2f9eR8dju. (Document), 3.4.4, 3.6, 3.7

[100] Amber Horvath, Sachin Grover, Sihan Dong, Emily Zhou, Finn Voichick, Mary Beth Kery, Shwetha Shinju, Daye Nam, Mariann Nagy, and Brad A. Myers. The long tail: Understanding the discoverability of API functionality. In Justin Smith, Christopher Bogart, Judith Good, and Scott D. Fleming, editors, kyRN A111 avKTQbBmK QM oBbm H G M;m >mK M@*2Mi`B+ *QKTmiBM;- oGf>** kyRN- J2KT?Bb- h2MM2bb22- kyRN, pages 157–161. IEEE Computer Society, 2019. doi: 10:1109/VLHCC:2019:8818681. URL ?iiTb,ff/:QB;fRRyRyNfo:kyRN3R3e13R. 2.1, 3.1, 3.5.2, 5.1, 5.5

[101] Amber Horvath, Michael Xieyang Liu, River Hendriksen, Connor Shannon, Emma Paterson, Kazi Jawad, Andrew Macvean, and Brad A Myers. Understanding How Programmers Can Use Annotations on Documentation. `sBp 2021. doi: 10:1145/3491102:3502095. 2.1

[102] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In AMi2`M iBQM H *QM72`2M+2 QM pages 2790–2799. G2 `MBM; PMLR, 2019. 5.4.1

[103] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. `sBp T`2T`BMi `sBp, 2018. 5.4.1

[104] Qiao Huang, Xin Xia, Zhenchang Xing, David Lo, and Xinyu Wang. Api method recommendation without worrying about the task-api knowledge gap. In AMi2`M iBQM H *QM72`2M QM miQK i2/ aQ7ir `2 1Ma;BM2 2BM. pages 283–294. IEEE, 2018. 2.4, 5.1

[105] Yintong Huo, Yuxin Su, Hongming Zhang, and Michael R. Lyu. ARCLIN: automated API mention resolution for unformatted texts. In AMi2`M iBQM H *QM72`2M+2 QM a 1M;BM2 2BM. pages 138–149. IEEE, 2022. 5.4

[106] Saki Imai. Is github copilot a substitute for human pair-programming? an empirical study. In S`Q+22/BM;b Q7 i?2 *JfA111 99i? AMi2`M iBQM H *QM72`2M+2 Q *QKT MBQM S`Q+22/BM;b. pages 319–321, 2022. 7.1

[107] Pankaj Jalote, Lionel Briand, André van der Hoek, Siddharth Subramanian, Laura Inozemtseva, and Reid Holmes. Live API documentation. S`Q+22/BM;b Q7 i?2 jei? AM i2`M iBQM H *QM72`2M+2 QM aQ7ir `2 1Ma;BM2 2BM. pages 642–651, 2014. doi: 10:1145/2568225:2568313. 2.4, 5.1

[108] Sae Young Jeong, Yingyu Xie, Jack Beaton, Brad A. Myers, Jeffrey Stylos, Ralf Ehret, Jan Karstens, Arkin Efeoglu, and Daniela K. Busse. Improving documentation for esoa apis through user studies. In Volkmar Pipek, Mary Beth Rosson, Boris E. R. de Ruyter, and Volker Wulf, editors, 1M/@lb2` .2p2HQTK2Mi- kM/ AMi2`M iBQM H avKT kyyN- aB2;2M- :2`K Mv- J `+? k@9, volume 5435 of G2/BM2 bLQi2b BM *QKTmi2` a, pages 86–105. Springer, 2009. doi: 10:1007/978-3-642-00427-8n_6. URL ?iiTb,ff/:QB;fRRyyydfNd3@j@e9k@3y9k8@23 3.3

[109] Susmit Jha, Sumit Gulwani, Sanjit A. Seshia, and Ashish Tiwari. Oracle-guided component-based program synthesis. pages 215–224, 2010. doi: 10:1145/1806799:1806833. URL ?iiTb,ff/:QB;fRRyR98fR3yR3yNe3Nj2

[110] Brittany Johnson, Yoonki Song, Emerson Murphy-Hill, and Robert Bowdidge. Why Don't Software Developers Use Static Analysis Tools to Find Bugs? kyRj j8i? AM@ i2`M iBQM H *QM72`2M+2 QM aQ7ia1 2, pages 672–682, 2013. 3.3, doi: 10:1109/icse:2013:6606613. 4.4

[111] Ron Johnston, Kelvyn Jones, and David Manley. Confounding and collinearity in regression analysis: a cautionary tale and an alternative procedure, illustrated by studies of british voting behaviour. Zm HBiv Zm, 52(1):80, 07 2018. doi: 10:1007/s11135-017-0584-6. 3.4.4, 7.7

[112] Cory J. Kapser and Michael W. Godfrey. "cloning considered harmful" considered harmful: patterns of cloning in software. 1KTB`B+ HaQ7ir `2, 13(6):645–692, 2008. 4.1, 4.2.1, 4.2.2, 4.2.2

[113] Majeed Kazemitabaar, Justin Chow, Carl Ka To Ma, Barbara J Ericson, David Weintrop, and Tovi Grossman. Studying the effect of AI Code Generators on Supporting Novice Learners in Introductory Programming. `sB2p, 2023. doi: 10:1145/3544548:3580919. 2.4

[114] Mik Kersten and Gail C. Murphy. Using task context to improve programmer productivity. S`Q+22/BM;b Q7i?2 R9i? *JaA:aP6h BMi2`M iBQM H bvKTQbBmK r `2 2M;BM22`BM; @yafA:aP@R9, pages 1–11, 2006. doi: 10:1145/1181775:1181777. 8.4.2

[115] Jinhan Kim, Sanghoon Lee, Seung-Won Hwang, and Sunghun Kim. Enriching Documents with Examples: A Corpus Mining Approach. *Jh` Mb +iBQMb QM AM7Q`K iBQM av (hPA, 31(1):1, 2013. ISSN 1046-8188. doi: 10:1145/2414782:2414783. 1.1.2

[116] Amy J. Ko and Yann Riche. The role of conceptual knowledge in API usability. In Gennaro Costagliola, Amy J. Ko, Allen Cypher, Jeffrey Nichols, Christopher Scaffidi, Caitlin Kelleher, and Brad A. Myers, editors, kyRR A111 avKTQbBmK QM oBbm H ;m ;2b M/ >mK M@*2Mi`B+ *QKTmiBM;- oGf>** kyRR- SBiib#m`;?- #2` R3@kk, pages 173–176. IEEE, 2011. doi: 10:1109/VLHCC:2011:6070395. URL ?iiTb,ff/:QB;fRRyRyNfo:kyRRydyj.N3. 1.1, 2.2, 3.4.1, 7.2.2, 7.6.2, 8.1

[117] Amy J. Ko, Brad A. Myers, Michael J. Coblenz, and Htet Htet Aung. An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks. A111 h` MbX aQ7i, 32(12):971–987, 2006. doi: 10:1109/TSE:2006:116. URL ?iiTb,ff/:QB;fRRyRyNfkyyRe. 1.1, 3.1, 3.1, 3.4, 3, 3.5.1, 7.1, 8.1

[118] Amy J. Ko, Robert DeLine, and Gina Venolia. Information needs in collocated software development teams. In kNi? AMi2`M iBQM H *QM72`2M+2 QM aQ7ir `2 1M

[illegible], pages 344–354. IEEE Computer Society, 2007. doi: 10:1109/ICSE:2007:45. URL [illegible]. 1, 2.1, 3.1, 3.5.2, 7.1

[119] A Gunes Koru and Hongfang Liu. Building effective defect-prediction models in practice. [illegible], 22(6):23–29, 2005. 4.4

[120] Sandeep Kaur Kuttal, Se Yeon Kim, Carlos Martos, and Alexandra Bejarano. How end-user programmers forage in online repositories? An information foraging perspective. [illegible], 62:101010, 2021. ISSN 2590-1184. doi: 10:1016/j:cola:2020:101010. 2.2

[121] Dmitry Lagun and Mounia Lalmas. Understanding user attention and engagement in online news reading. In Paul N. Bennett, Vanja Josifovski, Jennifer Neville, and Filip Radlinski, editors, [illegible], pages 113–122. ACM, 2016. doi: 10:1145/2835776:2835833. URL [illegible]. 3.1

[122] Ralf Lämmel and Simon Peyton Jones. [illegible], ACM, 2003. 4.2

[123] J Richard Landis and Gary G Koch. The measurement of observer agreement for categorical data. [illegible], pages 159–174, 1977. 5.2

[124] Thomas D. LaToza and Brad A. Myers. Hard-to-answer questions about code. [illegible], pages 8, 2010. doi: 10:1145/1937117:1937125. 1, 2.1

[125] Thomas D. LaToza, Gina Venolia, and Robert DeLine. Maintaining mental models: a study of developer work habits. In Leon J. Osterweil, H. Dieter Rombach, and Mary Lou Soffa, editors, [illegible], pages 492–501. ACM, 2006. doi: 10:1145/1134285:1134355. URL [illegible]. 2.2, 3.4.1, 7.1

[126] Thomas D LaToza, David Garlan, James D Herbsleb, and Brad A Myers. Program comprehension as fact finding. In [illegible], pages 361–370, 2007. 7.2.2, 8.2.1

[127] Joseph Lawrance, Rachel Bellamy, Margaret Burnett, and Kyle Rector. Using information scent to model the dynamic foraging behavior of programmers in maintenance tasks. In [illegible] '08, pages 1323–1332, New York, NY, USA, 2008. Association for Computing Machinery. ISBN 9781605580111. doi: 10:1145/1357054:1357261. URL [illegible]. 2.2

[128] Joseph Lawrance, Christopher Bogart, Margaret M. Burnett, Rachel K. E. Bellamy, Kyle Rector, and Scott D. Fleming. How programmers debug, revisited: An information foraging theory perspective. [illegible], 39(2):197–215, 2013. doi: 10:1109/TSE:2010:111. URL [illegible]. 2.2, 3.1

[129] Vu Le and Sumit Gulwani. Flashextract: A framework for data extraction by examples. In [illegible], pages 542–553, 2014. 6.2

[130] Younghwa Lee, Kenneth A Kozar, and Kai RT Larsen. The technology acceptance model: Past, present, and future. *QKKmMB+ iBQMb Q7 i?2 bbQ+B iBQM 7Q` BM7Q` 12, (1):50, 2003. 7.3, 7.6.1

[131] Juho Leinonen, Paul Denny, Stephen MacNeil, Sami Sarsa, Seth Bernstein, Joanne Kim, Andrew Tran, and Arto Hellas. Comparing Code Explanations Created by Students and Large Language Models. `sBp 2023. 2.4

[132] Timothy C Lethbridge, Janice Singer, and Andrew Forward. How software engineers use documentation: The state of the practice. A111 bQ7, 20(6):35–39, 2003. 5.1

[133] Hongwei Li, Zhenchang Xing, Xin Peng, and Wenyun Zhao. What Help Do Developers Seek, When and How? kyRj kyi? qQ`FBM; *QM72`2M+2 QM (q*p 21), b2 1M;BM2 pages 142–151, 2013. doi: 10:1109/wcre:2013:6671289. 2.2, 3.4.1

[134] Hongwei Li, Xuejiao Zhao, Zhenchang Xing, Lingfeng Bao, Xin Peng, Dongjing Gao, and Wenyun Zhao. amassist: In-ide ambient search of online programming resources. In kyR8 A111 kkM/ AMi2`M iBQM H *QM72`2M+2 QM aQ7ir `2 M HvbBb- 1pQ (a L1), pages 390–398. IEEE, 2015. 7.7

[135] Hongwei Li, Sirui Li, Jiamou Sun, Zhenchang Xing, Xin Peng, Mingwei Liu, and Xuejiao Zhao. Improving api caveats accessibility by mining api caveats knowledge graph. In AMi2`M iBQM H *QM72`2M+2 QM aQ7ir `2 J (AB*a 1 2M M+2 pages 183–193. IEEE, 2018. 1.1.2, 2.4

[136] Jing Li, Aixin Sun, and Zhenchang Xing. To Do or Not To Do: Distill crowdsourced negative caveats to augment api documentation. CQm`M HQ7i?2 bbQ+B iBQM 7Q` AM a+B2M+2 M/ h2??B:Q+Q, 69(12):2460–2475, 2018. ISSN 2330-1635. doi: 10:1002/asi:24067. 2.4

[137] Paul Luo Li, Amy J Ko, and Jiamin Zhu. What makes a great software engineer? In AMi2`M iBQM H *QM72`2M+2 QM (a *Qa7ir `2 1M;BM2 volume 1, pages 700–710. IEEE, 2015. 5.1

[138] Jenny T Liang, Chenyang Yang, and Brad A Myers. Understanding the usability of ai programming assistants. `sBp T`2T`BMi `sBp, 2023. j2X7dR8k8

[139] H Lieberman and T Selker. Out of context: Computer systems that adapt to, and learn from, context. A"Javbi2Kb C, 39(3.4):617–632, 2000. ISSN 0018-8670. doi: 10:1147/sj:393:0617. 8.1

[140] Bin Lin, Fiorella Zampetti, Gabriele Bavota, Massimiliano Di Penta, and Michele Lanza. Pattern-based mining of opinions in q&a websites. In AMi2`M iBQM H *QM72`2M+2 QM a 1M;BM22(ABa1), pages 548–559. IEEE, 2019. 2.4

[141] Bin Lin, Nathan Cassee, Alexander Serebrenik, Gabriele Bavota, Nicole Novielli, and Michele Lanza. Opinion mining for software development: A systematic literature review. *Jh` Mb +iBQMb QM aQ7ir `2 1M;BM22(hBMa;1), 31(3):1–1, 2022. 2.4

[142] Mario Linares-Vásquez, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, and Denys Poshyvanyk. How do api changes trigger stack overflow discussions? a study on the android sdk. In AMi2`M iBQM H *QM72`2M+2 QM S(A*S)*K*QK3T-94, 2014. bBQM 2.1, 5.1

[143] Bing Liu. q2# . i JBMBM;, 1tTHQ`BM; >vT2`HBMFb- *QMi2Mib- M/ lb 1/BiBQM. Data-Centric Systems and Applications. Springer, 2011. ISBN 978-3-642-19459-7. doi: 10:1007/978-3-642-19460-3. URL ?iiTb,ff/:QB;fRRyydfNd3@j@e9k@RN9ey@ 3.2.2

[144] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation, 2023. Retrieved from ?iiTb,ff `tRp`;f #bfk:jyR8k.R2y 4

[145] Michael Xieyang Liu, Jane Hsieh, Nathan Hahn, Angelina Zhou, Emily Deng, Shaun Burley, Cynthia Taylor, Aniket Kittur, and Brad A. Myers. Unakite: Scaffolding developers' decision-making using the web. In S`Q+22/BM;bQ7i?2jkM/ MMm H *JavKTQbBm AMi2`7 +2 aQ7ir `2 M/ UHT+?2 +?2 aQ7ir `2 M/ UHT+?2 +?2 aQ7ir `2 M/ UHT+?2 +?

[157] Sahar Mehrpour, Thomas D. LaToza, and Rahul K. Kindi. Active Documentation: Helping Developers Follow Design Decisions. kyRN A111 avKTQbBmK QM oBbm H G M;m >mK M@*2Mi`B+ *(QKT>mr;BM, pages 87–96, 2019. doi: 10:1109/vlhcc:2019:8818816. 3.1

[158] Michael Meng, Stephanie Steinhardt, and Andreas Schubert. Application programming interface documentation: What do software developers want? CQm`M H Q7 h2+?MB+ H q`B M/ *QKKmMB+ iBQM-48(3):295–330, 2018. doi: 10:1177/0047281617721853. URL ?iiTb, ff/Q:BQ`;fRRyRddfyy9dk3Re.R21,k3R,3:1j, 3.4, 3.5.1, 3.5.2, 3.5.3, 5.1, 7.2.2, 7.5.3

[159] Michael Meng, Stephanie Steinhardt, and Andreas Schubert. How developers use API documentation: an observation study. /HX +K,X2018.`; doi: 10:1145/3274995:3274999. 2.2, 3.1, 3.1, 3.4, 3.4.1, 2, 3.5.1, 5.5, 7.5.3

[160] Michael Meng, Stephanie M. Steinhardt, and Andreas Schubert. Optimizing API documentation: Some guidelines and effects. In Josephine N. Walwema, Daniel L. Hocutt, and Stacey Pigg, editors, aA:.P*ky, h?2j3i? *JAMi2`M iBQM H *QM72`2M+2 QM *QKKmMB+ iBQM-.2MiQM-hs-la -, pages 241–248. ACM, 2020. doi: 10:1145/3380851:3416759. URL ?iiTb,ff/:QB;fRRyR98fjj3yj39RRe d8.N, 3.1

[161] Andre N. Meyer, Laura E. Barton, Gail C. Murphy, Thomas Zimmermann, and Thomas Fritz. The Work Life of Developers: Activities, Switches and Perceived Productivity. A111 h`Mb +iBQMb QM aQ7ir,`43(12):1178–1193, 2017. ISSN 0098-5589. doi: 10:1109/tse:2017:2656886. 1, 7.1

[162] Tomás Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger, editors, /pM+2b BM L2m`H AM7Q`K iBQM S`Q+2bbBM; avb MMm H *QM72`2M+2 QM L2m`H AM7Q`K iBQM S`Q+2bbBM; avb Q7 K22iBM; ?2H/ .2+2K#2` 8@3- kyRj- G F2 h?Q2- L2p/- pages 3111–3119, 2013. URL ?iiTb,ffT`Q+22MB2m;bB+bT T2`fkyRjf? b?f N 9k#jR33k2+yjNNe87j+9Nkj+2?NykR#1@ #bi`+i

[163] Michela Montesi and Trilce Navarrete. Classifying web genres in context: A case study documenting the web genres used by a software engineer. AM7X S`Q+2b,b4 J M;X (4):1410–1430, 2008. doi: 10:1016/J:IPM:2008:02:001. URL ?iiTb,ff/:QB;fRRyRef DBT:Kyy.9kyy.R1

[164] Todd K Moon. The expectation-maximization algorithm. aB;M H S`Q+2bbB,M13; J ; xBM2 (6):47–60, 1996. 4.3.1

[165] Eduardo Mosqueira-Rey, David Alonso-Ríos, Vicente Moret-Bonillo, Isaac Fernández-Varela, and Diego Álvarez-Estévez. A systematic approach to API usability: Taxonomy-derived criteria and a case study. AM7Q`K iBQM M/ aQ7ir, 97:246–263, 2018. QHQ;v 4.1, 4.1, 4.2

[166] Hussein Mozannar, Gagan Bansal, Adam Fourney, and Eric Horvitz. Reading Between the Lines: Modeling User Behavior and Costs in AI-Assisted Programming. `sBp, 2022. doi: 10:48550/arxiv:2210:14306. 2.4

[167] Lauren Murphy, Mary Beth Kery, Oluwatosin Alliyu, Andrew Macvean, and Brad A Myers. Api designers in the field: Design practices and challenges for creating usable apis. In kyR3 B222 bvKTQbBmK QM pBbm H H M;m ;2b M/ H?m?2K M@ 2018 i`B+ +QKTm (pp 249–258), pages 249–258. IEEE, 2018. 4.1

[168] Brad A Myers and Jeffrey Stylos. Improving API usability. *QKKmMB+ iBQMb,Q7 i?2 *J 59(6):62–69, 2016. 4.1, 4.1

[169] Varvana Myllärniemi, Sari Kujala, Mikko Raatikainen, and Piia Sevón. Development as a journey: factors supporting the adooption and use of software frameworks. CX aQ7irX 1M;X _2bX „2bX 2018. doi: 10:1186/S40411-018-0050-8. URL ?iiTb://QB;fRRyR3ef b9y9RR@yR3.@1yy8y@3 26, 2018, 4.1

[170] Sarah Nadi, Stefan Krüger, Mira Mezini, and Eric Bodden. Jumping through hoops: Why do java developers struggle with cryptography apis? In AMi2`M iBQM H *QM72`2M+ aQ7ir `2 1M;BM22`BM;- A*a1 ky (MA2a)1BM, pages 935–946, 2016. 2.1, 5.1

[171] N. J. D. NAGELKERKE. A note on a general definition of the coefficient of determination. "BQK2i,`B(F):691–692, 09 1991. ISSN 0006-3444. doi: 10:1093/biomet/78:3:691. URL ?iiTb://QB;fRRyNjf#BQK2efMB 2.4.5

[172] Daye Nam. API design implications of boilerplate client code. In j9i? A111f *J AM@ i2`M iBQM H *QM72`2M+2 QM miQK i2/ aQ7ir `2 1M;BM22`BM;- a1 la - LQp2K#2` RR@R8-1253 R15. IEEE, 2019. doi: 10:1109/ASE:2019:00153. URL ?iiTb://QB;fRRyRyNf.ky1RyNyR.8ij, 2, 3

[173] Daye Nam, Amber Horvath, Andrew Macvean, Brad Myers, and Bogdan Vasilescu. Marble source code and the result. ?iiTb://QB;fR8yk3Rfx2Nj9y/3Qd, 2019. 4.1, 4.4.2

[174] Daye Nam, Amber Horvath, Andrew Macvean, Brad A. Myers, and Bogdan Vasilescu. MARBLE: mining for boilerplate code to identify API usability problems. In j9i? A111f *J AMi2`M iBQM H *QM72`2M+2 QM miQK i2/ aQ7ir `2 1M;BM22`BM;- a M .B2;Q- * - la - LQp2K#2`, pages 6R5-627. IEEE, 2019. doi: 10:1109/ASE:2019:00063. URL ?iiTb://QB;fRRyRyNf.ky1RyNyyej1, 2, 3, 3.4.5, 6, 5.1, 6.8

[175] Daye Nam, Baishakhi Ray, Seohyun Kim, Xianshan Qu, and Satish Chandra. Predictive synthesis of api-centric code. In Swarat Chaudhuri and Charles Sutton, editors, J Sa!SG.A kykk, ei? *J aA:SG L AMi2`M iBQM H avKTQbBmK S`Q;` KKBM;- a M .B2;Q- * - la - RjGmM 29.kyK, pages 4M-29. ACM, 2022. doi: 10:1145/3520312:3534866. URL ?iiTb://QB;fRRyR98fj8kyjjR9k3e2, 16

[176] Daye Nam, Andrew Macvean, Vincent J. Hellendoorn, Bogdan Vasilescu, and Brad A. Myers. In-ide generation-based information support with a large language model. *Q_,_abs/ 2307.08177, 2023. doi: 10:48550/ARXIV:2307:08177. URL ?iiTb://QB;fR9y388yf `sBkjy:dy3R.dd, 4.20, 8.1

[177] Daye Nam, Brad A. Myers, Bogdan Vasilescu, and Vincent J. Hellendoorn. Improving API knowledge discovery with ML: A case study of comparable API methods. In 98i? A111f *J AMi2`M iBQM H *QM72`2M+2 QM aQ7ir `2 1M;BM22`BM;- A*a1 ky J vR9@ky,-kyek, pages 1890–1906. IEEE, 2023. doi: 10:1109/ICSE48619:2023:00161. URL ?iiTb://QB;fRRyRyNfA*a1ky3kejyN.e2, 3, 4, 3.5.2, 8

[178] Daye Nam, Andrew Macvean, Vincent Hellendoorn, Bogdan Vasilescu, and Brad Myers. Artifacts for Using an LLM to Help With Code Understanding, January 2024. URL ?iiTb, ff/Q:BQ`;fR8yk3Rfx2:MRQ/9QeRj7:8

[179] Daye Nam, Andrew Macvean, Brad Myers, and Bogdan Vasilescu. Understanding documentation use through log analysis: A case study of four cloud services. In S`Q+22/BM;b Q7 i?2 kyk9 *>A *QM72`2M+2 QM >mK M 6 +iQ`b BM *QKTmiBM; avbi2K, 2024. 1, 3.1

[180] Anh Tuan Nguyen, Hoan Anh Nguyen, Tung Thanh Nguyen, and Tien N. Nguyen. Statistical learning approach for mining API usage mappings for code migration. In Ivica Crnkovic, Marsha Chechik, and Paul Grünbacher, editors, AMi2`M iBQM H *QM72`2M+2 miQK i2/ aQ7ir `2 1M;BM22 ?B-Ma-MA 6à1M+Bb+Q- * - la - J v, pages 457–468. ACM, 2014. 5.1

[181] Christina Niklaus, Matthias Cetto, André Freitas, and Siegfried Handschuh. A survey on open information extraction. In AMi2`M iBQM H *QM72`2M+2 QM *QKTmi iBQ (*PGAL, pages 3866–3878, 2018. 5.1

[182] Nan Niu, Anas Mahmoud, Zhangji Chen, and Gary Bradshaw. Departures from optimality: understanding human analyst's information foraging in assisted requirements tracing. In David Notkin, Betty H. C. Cheng, and Klaus Pohl, editors, j8i? AMi2`M iBQM H *QM72`2 QM aQ7ir `2 1M;BM22 ?B-Ma-MA 6à1M+Bb+Q- * - la - J v, pages 572–581. IEEE Computer Society, 2013. doi: 10:1109/ICSE:2013:6606603. URL ?iiTb, ff/Q:BQ`;fRRyRyNfAk*yR9eyee9. 2.2

[183] Maxwell I. Nye, Luke B. Hewitt, Joshua B. Tenenbaum, and Armando Solar-Lezama. Learning to infer program sketches. 97:4861–4870, 2019. URL ?iiT,ff T`Q+22/BKMH;Tb`2bbfpNdfM?vi2KRN. 2.2

[184] Maxwell I. Nye, Yewen Pu, Matthew Bowers, Jacob Andreas, Joshua B. Tenenbaum, and Armando Solar-Lezama. Representing partial programs with blended abstract semantics. 2021. URL ?iiTb,ffQT2M`:2Mp2Bf27rQ?`Bm+K*i /[At.R6. 2.2

[185] Janet Nykaza, Rhonda Messinger, Fran Boehme, Cherie L. Norman, Matthew Mace, and Manuel Gordon. What programmers really want: results of a needs assessment for SDK documentation. S`Q+22/BM;b Q7 i?2 kyi? MMm H BMi2`M iBQM H +QM /Q+mK2Mi iBQM @yka, pages 133–141, 2002. doi: 10:1145/584955:584976. 2.1, 3.1

[186] Augustus Odena, Kensen Shi, David Bieber, Rishabh Singh, Charles Sutton, and Hanjun Dai. BUSTLE: bottom-up program synthesis through learning-guided exploration. 2021. URL ?iiTb,ffQT2M`:2Mp2Bf27rQ?`Bm+Kv>2;9S#6. 2 ?

[187] Stephen Oney and Joel Brandt. Codelets: linking interactive documentation and example code in the editor. In Joseph A. Konstan, Ed H. Chi, and Kristina Höök, editors, *>A *QM72`2M+2 QM >mK M 6 +iQ`b BM *QKTmiBM; avbi2K-la *>@ J v y8 @ Ry-, pages 2697–2706. ACM, 2012. doi: 10:1145/2207676:2208664. URL ?iiTb, ff/Q:BQ`;fRRyR98fkkyykkyd3ee. 2.9

[188] Amantia Pano, Daniel Graziotin, and Pekka Abrahamsson. Factors and actors leading to the adoption of a javascript framework. 1KTB`X aQ7ir 2X(61:M03X3534, 2018. doi: 10:1007/S10664-018-9613-X. URL ?iiTb,ff/:QB;fRRyyydfbRyee9@yR3. 1@NeRj@t

[189] Emilio Parisotto, Abdel-rahman Mohamed, Rishabh Singh, Lihong Li, Dengyong Zhou, and Pushmeet Kohli. Neuro-symbolic program synthesis. 2017. URL ?iiTb,ff QT2M`:2Mp2Bf27rQ?`Bm+KCyCr6. +62t

[190] Chris Parnin, Christoph Treude, Lars Grammel, and Margaret-Anne Storey. Crowd documentation: Exploring the coverage and the dynamics of api discussions on stack overflow. Technical Report GIT-CS-12-05, Georgia Institute of Technology, 2012. 2.4, 5.1

[191] Nate Parsons. Boilerplate code definition of stackoverflow. ?iiTb,ffbi +FQp2+ QHKQr [m2biBQMbfjNNkRNNfr? i@Bb@.#4Q,B4H2.2`, 4T+2 i2@+Q/2

[192] Mateusz Pawlik and Nikolaus Augsten. Tree edit distance: Robust and memory-efficient. AM7Q`K iBQM56 a57i-2K3, 2016. 4.3.3

[193] David N Perkins, Chris Hancock, Renee Hobbs, Fay Martin, and Rebecca Simmons. Conditions of learning in novice programmers. CQm`M H Q7 1/m+ iBQM H *,2KTmiBM; _ (1):37–55, 1986. 7.5.3

[194] Neil Perry, Megha Srivastava, Deepak Kumar, and Dan Boneh. Do users write more insecure code with AI assistants? *Q_,_abs/2211.03622, 2022. doi: 10:48550/arXiv:2211:03622. URL ?iiTb,ff/:QB;fR9y388yf `:kRyjek k2.4

[195] Dimitrios Pierrakos, Georgios Paliouras, Christos Papatheodorou, and Constantine D. Spyropoulos. Web usage mining as a tool for personalization: A survey. lb2` JQ/2HX lb2` / TiX AMi2(4):31X-372, 2003. doi: 10:1023/A:1026238916441. URL ?iiTb, ff/:QQ`;fRRyykjf ,Rykekj3N:5e199R

[196] David Piorkowski, Austin Z. Henley, Tahmid Nabi, Scott D. Fleming, Christopher Scaffidi, and Margaret Burnett. Foraging and navigations, fundamentally: developers' predictions of value and cost. In S`Q+22/BM;b Q7 i?2 kyRe k9i? *J aA:aP6h A iBQM H avKTQbBmK QM 6QmM/ iBQM bFSE72016 Q7 i?2 `27 10M, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450342186. doi: 10:1145/2950290:2950302. URL ?iiTb,ff/:QQB;fRRyR98fkN8kyk8ywjyk7.1

[197] David J Piorkowski, Scott D Fleming, Irwin Kwan, Margaret M Burnett, Christopher Scaffidi, Rachel K E Bellamy, and Joshua Jordahl. The whats and hows of programmers' foraging diets. S`Q+22/BM;b Q7 i?2 aA:*>A *QM72`2M+2 QM >mK M 6 +iQ avbi2Kb, pages 3063–3072, 2013. doi: 10:1145/2470654:2466418. 8.1

[198] Peter Pirolli and Stuart Card. Information foraging. Sbv+?QHQ;B,+106(4):643,1999. 2.2

[199] Peter L. T. Pirolli. AM7Q`K iBQM 6Q`;BM; h?2Q`v, / TiBp2 AMi2` AM7Q`K iBQM. Oxford University Press, 05 2007. ISBN 9780195173321. doi: 10:1093/acprof:oso/9780195173321:001:0001. URL ?iiTb,ff/:QQB;fRRyyNjf +T`Q7, QbQfNd3yRN:8yR:Byjyk R.5.2

[200] Reinhold Plösch, Andreas Dautovic, and Matthias Saft. The value of software documentation quality. In kyR9 R9i? AMi2`M iBQM H *QM72`2M+2 QM Zm HBiv a la - P+iQ#2` k@, pages 333–342. IEEE, 2014. doi: 10:1109/QSIC:2014:22. URL ?iiTb,ff/:QQB;fRRyRyNfZkyAR*9k 3.1

[201] Luca Ponzanelli, Alberto Bacchelli, and Michele Lanza. Seahawk: Stack overflow in the ide. In AMi2`M iBQM H *QM72`2M+2 QM (aQ7i, pages 1295–1298. IEEE, 2013. 2.4

[202] Luca Ponzanelli, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, and Michele Lanza. Mining stack overflow to turn the ide into a self-confident programming prompter. In AMi2`M iBQM H *QM72`2M+2 QM JBMBJM;, pages 102–211, 2014. bBiQ`B2b

2.4

[203] Christi-Anne Postava-Davignon, Candice Kamachi, Cory Clarke, Gregory Kushmerek, Mary Beth Rettger, Pete Monchamp, and Rich Ellis. Incorporating usability testing into the documentation process. h2+?MB+ H *QKK, 51:636–644, 01 2010. 2.2, 3.4.1

[204] Sruti Srinivasa Ragavan, Mihai Codoban, David Piorkowski, Danny Dig, and Margaret Burnett. Version control systems: An information foraging perspective. A111 h` MbX aQ7ir `2 ,47(8):1644–1655, 2021. doi: 10:1109/TSE:2019:2931296. URL ?iiTb,ff /Q:BQ`;fRRyRyNfkyRkNNjRkN2e

[205] Nikitha Rao, Chetan Bansal, Thomas Zimmermann, Ahmed Hassan Awadallah, and Nachiappan Nagappan. Analyzing web search behavior for software engineering tasks. In Xintao Wu, Chris Jermaine, Li Xiong, Xiaohua Hu, Olivera Kotevska, Siyuan Lu, Weija Xu, Srinivas Aluru, Chengxiang Zhai, Eyhab Al-Masri, Zhiyuan Chen, and Jeff Saltz, editors, kyky A111 AMi2`M iBQM H *QM72`2M+2 QM "B; "B;.i ky,kyiH Mi-:-la-.2+2K#2` Ry,@Ris- 7k8-k7. IEEE, 2020. doi: 10:1109/BIGDATA50022:2020:9378083. URL ?iiTb,ff/:QB;fRRyRyNf "B;.i 8yykykyNjd3y3j. 2.1, 3.1, 3, 8.1

[206] Irum Rauf, Pekka Perälä, Jouni Huotari, and Ivan Porres. Perceived Obstacles by Novice Developers Adopting User Interface APIs and Tools. kyRe A111 avKTQbBmK QM oBb G M;m ;2b M/ >mK M@ *2Mi(`oBG+f>Q*K Tm i2BM2, pages 23–27, 2016. doi: 10:1109/vlhcc:2016:7739689. 4, 3.5.2

[207] Martin Reddy. SA .2bB;M 7Q Elsevier, 2011. 4.1, 4.1, 4.2, 4.2.1, 4.2.2, 4.2.2

[208] Xiaoxue Ren, Zhenchang Xing, Xin Xia, Guoqiang Li, and Jianling Sun. Discovering, explaining and summarizing controversial discussions in community q&a sites. In AMi2`M @ iBQM H *QM72`2M+2 QM miQK i2/ aQ7ir, pages 151–162. IEEE, 2019. 2.4

[209] Xiaoxue Ren, Jiamou Sun, Zhenchang Xing, Xin Xia, and Jianling Sun. Demystify official api usage directives with crowdsourced api misuse scenarios, erroneous code examples, and patches. In AMi2`M iBQM H *QM72`2M+2 QM aQ7ir, pages 925–936, 2022. 2.4, 5.1

[210] Martin P. Robillard. What makes apis hard to learn? answers from developers. A111 aQ7ir,26(6):27–34, 2009. doi: 10:1109/MS:2009:193. URL ?iiTb,ff/:QB;fRRyRyNf Jakyy.RNj. 2.1, 3.1

[211] Martin P. Robillard and Yam B. Chhetri. Recommending reference API documentation. 1KTB`X aQ7ir,20(6):1558–1586, 2015. doi: 10:1007/s10664-014-9323-y. URL ?iiTb, ff/:QB;fRRyydfbRyee9@yR9@Njkj@v

[212] Martin P. Robillard and Robert DeLine. A field study of API learning obstacles. 1K@ TB`X aQ7ir,16(1):703–732, 2011. doi: 10:1007/S10664-010-9150-8. URL ?iiTb, ff/:QB;fRRyydfbRyee9@yR9@3NR85y@3

[213] Martin P Robillard, Wesley Coelho, and Gail C Murphy. How effective developers investigate source code: An exploratory study. A111 h` Mb +iBQMb QM bQ7ir,30 `2 2M;BM2 (12):889–903, 2004. 1.1.1

[214] Martin P Robillard, Eric Bodden, David Kawrykow, Mira Mezini, and Tristan Ratchford. Automated API property inference techniques. h` Mb +iBQMb QM aQ7ir,32 1M;BM22`

(5):613–637, 2013. 4.1

[215] Martin P. Robillard, Andrian Marcus, Christoph Treude, Gabriele Bavota, Oscar Chaparro, Neil A. Ernst, Marco Aurélio Gerosa, Michael W. Godfrey, Michele Lanza, Mario Linares Vásquez, Gail C. Murphy, Laura Moreno, David C. Shepherd, and Edmund Wong. On-demand developer documentation. In kyRd A111 AMi2`M iBQM H *QM72`2M+2 Q J BMi2M M+2 M/ 1pQHmiBQM- A*aJ1 kyRd- a? M;? B, pages 479–483. IEEE Computer Society, 2017. doi: 10:1109/ICSME:2017:17. URL ?iiTb,ff /Q:BQ`;fRRyRyNfA:*kayJ1Rd. 3.1, 8.1

[216] Reudismam Rolim, Gustavo Soares, Loris D'Antoni, Oleksandr Polozov, Sumit Gulwani, Rohit Gheyi, Ryo Suzuki, and Björn Hartmann. Learning syntactic program transformations from examples. pages 404–415, 2017. doi: 10:1109/ICSE:2017:44. URL ?iiTb,ff/:QB;fRRyRyNfAk*aR9l. 6.2

[217] Charles Romesburg. *Hmbi2` M HvbBb 7.Q. Lulu.com, 2004. 2.3.1

[218] Christoffer Rosen and Emad Shihab. What are mobile developers asking about? a large scale study using stack overflow. 1KTB`B+ H aQ7ir(X):1M2–1223, jun 2016. ISSN 1382-3256. doi: 10:1007/s10664-015-9379-3. URL ?iiTb,ff/:QB;fRRyydfbRyee9@ yR8@Nj.2N@j

[219] Steven I. Ross, Fernando Martinez, Stephanie Houde, Michael J. Muller, and Justin D. Weisz. The programmer's assistant: Conversational interaction with a large language model for software development. In S`Q+22/BM;bQ7i?2k3i? AMi2`M iBQM H *Q AMi2HHB;2Mi lb2` AMi2`7 +2b- AlA kykj- av/M2v- Laq, pages 491–514. ACM, 2023. doi: 10:1145/3581641:3584037. URL ?iiTb,ff/:QB;fRRyR98f j83Rej8R39y.j2. 2.4, 7.3, 8.1

[220] Chanchal Kumar Roy and James R Cordy. A survey on software clone detection research. Zm2`bMa+?QQH Q7 *QK7m(iB)M, 541:64–68, 2007. 4.2.3, 4.2.3

[221] Riccardo Rubei, Claudio Di Sipio, Phuong T Nguyen, Juri Di Rocco, and Davide Di Ruscio. Postfinder: Mining stack overflow posts to support software developers. AM7Q`K iBQM M/ aQ7ir `2 h2+?M7Q+Q6Q7, v2020. 2.4

[222] Advait Sarkar, Andrew D Gordon, Carina Negreanu, Christian Poelitz, Sruti Srinivasa Ragavan, and Ben Zorn. What is it like to program with artificial intelligence? `sBp, 2022. doi: 10:48550/arxiv:2208:06213. 2.4

[223] Sami Sarsa, Paul Denny, Arto Hellas, and Juho Leinonen. Automatic generation of programming exercises and code explanations using large language models. In *J *QM@ 72`2M+2 QM AMi2`M iBQM H *QKTmiBAM 1 1/ky,kRbQ MQ2bM2/ `+B?`@ im H 1p2Mi- arBix2`H M/- m;mbi d @ pages 27–43. ACM, 2022. doi: 10:1145/3501385:3543957. URL ?iiTb,ff/:QB;fRRyR98fj8y:Rj9 8N.82d. 2.4

[224] M. Schuster and K.K. Paliwal. Bidirectional recurrent neural networks. A111 h` Mb +iBQMb QM aB;M H S`Q+2bbBM, 45(11):2673–2681, 1997. doi: 10:1109/78:650093. 18

[225] SeanCurtis-TRI. Collect GeometrySystem ! drake_visualizer boilerplate by SeanCurtis-TRI pull request #8526 RobotLocomotion/drake. ?iiTb,ff;Bi?+mQ#Kf _Q#QiGQ+QKQiBQMf/` .F21f 1.2mHHf38ke

[226] David E. Shaw, William R. Swartout, and C. Cordell Green. Inferring lisp programs from examples. In S`Q+22/BM;bQ7i?29i? AMi2`M iBQM H CQBMi*QM72`2M+2

@ oQ H mK2AIR75, pages 260–267, San Francisco, CA, USA, 1975. Morgan Kaufmann Publishers Inc. 6.2

[227] Mary Shaw. Prospects for an engineering discipline of software. A 1 1 1 aQ 7,i7(6):25–24, 1990. 5.1

[228] Xuehua Shen, Bin Tan, and ChengXiang Zhai. Implicit user modeling for personalized search. S`Q+22/BM;b Q7 i?2 R9i? *J BMi2`M iBQM H +QM72`2M+2 QM A K M ;2K2Mi @'yAE1, pages 824–831, 2005. doi: 10:1145/1099554:1099747. 3.5.3

[229] Kensen Shi, David Bieber, and Rishabh Singh. Tf-coder: Program synthesis for tensor manipulations. *Q _,_ abs/2003.09040, 2020. URL ?iiTb,ff `tBQp`;f #bfk:yyNy.9y (document), 6.1, 6.1, 6.1, 6.1, 6.2, 6.5, 6.6.1, 6.6.3, B.2

[230] Peng Shi and Jimmy Lin. Simple bert models for relation extraction and semantic role labeling. `sBp T`2T`BMi `sBp,2R1N.y29Xy8k88

[231] Richard Shin, Illia Polosukhin, and Dawn Song. Improving neural program synthesis with inferred execution traces. pages 8931–8940, 2018. URL ?iiTb,ffT`Q+22/B2Mn;bB+b T T2`fkyR3f? b?fddde233#y+R3N8jNyN3Rde?8i3Nk28y#+# @ #bi` +i

[232] Richard Shin, Neel Kant, Kavi Gupta, Chris Bender, Brandon Trabucco, Rishabh Singh, and Dawn Song. Synthetic datasets for neural program synthesis. 2019. URL ?iiTb, ffQT2M`2pMB2Mf7Q?Bm+K v2PaM.pu3K

[233] Jonathan Sillito, Kris De Voider, Brian Fisher, and Gail Murphy. Managing software change tasks: An exploratory study. In kyy8 AMi2`M iBQM H avKTQbBmK QM 1KTB`B 1M;BM22`BM, pages 10–pp. IEEE, 2005. 1.1.1

[234] Jonathan Sillito, Gail C. Murphy, and Kris De Volder. Asking and answering questions during a programming change task. A111 h` Mb +iBQMb QM aQ7i,r34(2):1M;BM22` 434–451, 2008. doi: 10:1109/TSE:2008:26. 2.1, 3.1

[235] Calvin Smith and Aws Albarghouthi. Mapreduce program synthesis. pages 326–340, 2016. doi: 10:1145/2908080:2908102. URL ?iiTb,ff/:QB;fRRyR98fkNyk3Ny3y R6y1k

[236] Armando Solar-Lezama, Liviu Tancau, Rastislav Bodík, Sanjit A. Seshia, and Vijay A. Saraswat. Combinatorial sketching for finite programs. pages 404–415, 2006. doi: 10:1145/ 1168857:1168907. URL ?iiTb,ff/:QB;fRRyR98fRR:eR3R83Ny2d

[237] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. CX J +?X G2 `,M15(1):1929–1958, 2014. doi: 10:5555/2627435:2670313. URL ?iiTb,ff:/H+KQ`;f/QB:f8Ry88fkek:dk9ejd8yj.Ry4.2

[238] Cuyler Stuwe. Boilerplate code definition of quora. ?iiTb,ffr:r[mQ:`+QKfq? i@Bb@ #QBH2`TH i2@2.2Q/2

[239] Jeffrey Stylos and Brad A Myers. The implications of method placement on API learnability. In AMi2`M iBQM H avKTQbBmK QM 6QmM/ iBQMb 10–112. Q7ir `2 2M ACM, 2008. 4.1, 5, 5.1

[240] Ning Su, Jiyin He, Yiqun Liu, Min Zhang, and Shaoping Ma. User intent, behaviour, and perceived satisfaction in product search. In Yi Chang, Chengxiang Zhai, Yan Liu, and Yoelle Maarek, editors, S`Q+22/BM;b Q7 i?2 1H2p2Mi? *J AMi2`M iBQM H q2# a2 `+? M/ . i JBMBM;- qa.J kyR3- J `BM .2H _2v- * - la - 62#

k y R 3 , pages 547–555. ACM, 2018. doi: 10:1145/3159652:3159714. URL ? i i T b , f f / : Q B ; f R : y R R 9 8 f j R 8 N R e 8 N k d 8 R 1 9

[241] Harsh Suri. Purposeful sampling in qualitative research synthesis. Z m H B i i B p 2 ` 2 b 2 ` + ? D Q m , 1 M ( 2 ) : 63–75, 2011. ISSN 1443-9883. 3.2.1

[242] Jeniya Tabassum, Mounica Maddela, Wei Xu, and Alan Ritter. Code and named entity recognition in Stack Overflow. In M M m H J 2 2 i B M ; Q 7 i ? 2 b b Q + B i B Q M 7 Q ` * Q G B M ; m B b i G B + b , pages 4913–4926. Association for Computational Linguistics, 2020. 5.4, 14

[243] Hengzhu Tang, Yanan Cao, Zhenyu Zhang, Jiangxia Cao, Fang Fang, Shi Wang, and Pengfei Yin. Hin: Hierarchical inference network for document-level relation extraction. In S + B } + @ b B * Q M 7 2 ` 2 M + 2 Q M E M Q r H 2 / ; 2 . B b + Q , pages 127–209. M / . i J B M Springer, 2020. 5.4.7

[244] taynaud. Community detection package. ? i i T b , f f T v i ? Q M @ : H 2 Q m i ? 2 5 Q B - Q 4.4.1

[245] Kyle Thayer, Sarah E Chasins, and Amy J Ko. A Theory of Robust API Knowledge. * J h ` M b + i B Q M b Q M * Q K T m , 2 B ( M ) : 1–32, 2021. i B Q M 10:1145/3444945. 3.1, 5.1, 7.2.2, 7.3

[246] Haoye Tian, Weiqi Lu, Tsz On Li, Xunzhu Tang, Shing-Chi Cheung, Jacques Klein, and Tegawendé F. Bissyandé. Is chatgpt the ultimate programming assistant – how far is it?, 2023. Retrieved from ? i i T b , f f ` t B Q p ; f # b f k j R y R N j 3 4

[247] Yuan Tian, Ke Zhou, and Dan Pelleg. Characterization and prediction of mobile tasks. * J h ` M b X A M 7 X ( 1 ) : 1–13:39, 2023. doi: 10:1145/3522711. URL ? i i T b , f f / Q : B Q ` ; f R R y R 9 8 f j 8 k k d 3 2 R

[248] Tim. How to avoid writing duplicate boilerplate code for requesting permissions? ? i i T b , f f b i + F Q p 2 + Q H K Q [ m 2 b i B Q M b f j N y 3 y y N 8 f ? Q r @ i Q @ p Q B / @ / m T H B + i 2 @ # Q B H 2 ` T H i 2 @ + Q / 2 @ 7 Q ` @ ` . 2 4 [ m 2 2 b i B M ; @ T 2 ` K B b b B Q M b

[249] Nava Tintarev and Judith Masthoff. Evaluating the effectiveness of explanations for recommender systems - methodological issues and empirical studies on the impact of personalization. l b 2 ` J Q / 2 H X l b 2 ` / T i X ( 4 ) : 399–439, 2012. doi: 10:1007/s11257-011-9117-5. URL ? i i T b , f f / : Q B ; f R R y y y d f b R R k 8 d @ y R 3 5 @ N R R d @ 8

[250] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023. 1.1.2, 2.3

[251] Christoph Treude and Martin P Robillard. Augmenting API documentation with insights from stack overflow. A*a, 2016. doi: 10:1145/2890000/2884800/p392-treude:pdf. 1.1.2, 2.4, 3.1, 5.1

[252] Christoph Treude, Ohad Barzilay, and Margaret-Anne Storey. How do programmers ask and answer questions on the web? (nier track). In S`Q+22/BM;bQ7i?2jj`/AMi2`M iBQM 72`2M+2 QM aQ7ir `2 1M;BM22`BM, ICSE '11, pages 804–807, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450304450. doi: 10:1145/1985793:1985907. URL ?iiTb,ff/:QB;fRyR98fRN:3RdN8.N2ld, 8.2.1

[253] Jaroslav Tulach. S` +iB+ H SA.2bB;M, *QM72bbBQMbQ7. Apress, 6` K2rQ`F September 2008. 4.1

[254] Gias Uddin and Foutse Khomh. Opiner: an opinion search and summarization engine for apis. In Grigore Rosu, Massimiliano Di Penta, and Tien N. Nguyen, editors, AMi2`M iBQM H *QM72`2M+2 QM miQK i2/aQ7ir`2 1M;BM22`BM, ASE, pages 978–982. IEEE, 2017. 2.4

[255] Gias Uddin and Martin P. Robillard. How API documentation fails. A111 aQ7ir, 32(4): 68–75, 2015. doi: 10:1109/MS:2014:80. URL ?iiTb,ff/:QB;fRyRyN:kJyR9y, 2.1, 8.1

[256] Gias Uddin, Olga Baysal, Latifa Guerrouj, and Foutse Khomh. Understanding how and why developers seek and analyze api-related opinions. A111 h`Mb +iBQMbQM aQ7ir 1M;BM22`BM, 27(8): 694–735, 2019. 2.1

[257] Gias Uddin, Foutse Khomh, and Chanchal K Roy. Mining api usage scenarios from stack overflow. AM7Q`K iBQM M/aQ7ir, 122: 106277, 2020. 2.4

[258] Gias Uddin, Foutse Khomh, and Chanchal K Roy. Automatic api usage scenario documentation from technical q&a sites. *Jh`Mb +iBQMbQM aQ7ir `2 1M;BM22`B J2i?Q/Q(HQP;a/1), 30(3):1–45, 2021. 2.4, 5.4.6

[259] Abhishek Udupa, Arun Raghavan, Jyotirmoy V. Deshmukh, Sela Mador-Haim, Milo M. K. Martin, and Rajeev Alur. TRANSIT: specifying protocols with concolic snippets. pages 287–296, 2013. doi: 10:1145/2491956:2462174. URL ?iiTb,ff/:QB;fRyR98fk9NR:N8ekRd9, 6.2, 6.5

[260] user2999943. How to avoid boilerplate code when loading images with picasso library. ?iiTb,ffbi +FQp2+ QKQ[m2biBQMbfjkRedN93f?Qr@iQ@ #QBH2`TH i2@+Q/2@r?2M@HQ /BM;@BK ;2b@ 4.4.2

[261] Priyan Vaithilingam, Tianyi Zhang, and Elena L Glassman. Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models. *>A *QM72`2M+2 QM >mK M 6 +iQ`b BM *QKTmiBM, pages 1–7, avb1i2Kb 1ti2M/2 2022. doi: 10:1145/3491101:3519665. 2.4, 7.1, 7.8

[262] Enrique Larios Vargas, Maurício Finavaro Aniche, Christoph Treude, Magiel Bruntink, and Georgios Gousios. Selecting third-party libraries: the practitioners' perspective. In Prem Devanbu, Myra B. Cohen, and Thomas Zimmermann, editors, 1a1*f6a'1ky, k3i? *J CQBMi 1m`QT2 M aQ7ir `2 1M;BM22`BM; *QM72`2M+2 M/avKTQbB aQ7ir `2 1M;BM22`BM;-oB`im H 1p2Mi-la, pages 245–256. ACM@ Rj-kyky 2020. doi: 10:1145/3368089:3409711. URL ?iiTb,ff/:QB;fRyR98fjje3jy93NdRR 4, 3.5.2

[263] Marcello Visconti and Curtis R. Cook. Assessing the state of software documentation practices. In Frank Bomarius and Hajimu Iida, editors, S`Q/m+i 6Q+mb2/ aQ7ir `2 S`Q+ AKT`Qp2K2Mi- 8i? AMi2`M iBQM H *QM72`2M+2- S_P61a kyy9- E m T`BH 8@3- kyy9-, volume 2009 of LNCS, pages 485–496. Springer, 2004. doi: 10:1007/978-3-540-24659-6n_35. URL ?iiTb,ff/:QB;f R:yRyydfNd3@j@89y@.k9e8N@enj8 1.1.2

[264] Han Wang, Chunyang Chen, Zhenchang Xing, and John Grundy. DiffTech: a tool for differencing similar technologies from question-and-answer discussions. S`Q+22/BM;b Q7 i?2 k3i? *J CQBMi J22iBM; QM 1m`QT2 M aQ7ir `2 1M;BM22` avKTQbBmK QM i?2 6QmM/ iBQMb Q, pages 1576–1581, 2020. doi: 10:1145/3368089:3417931. (document), 1.1.2, 5.1, 5.3, 5.4.4, 5.4.4, 5.4.4, 5.4.4

[265] Roger C S Wernersson. Can java help me avoid boilerplate code in equals()? ?iiTb,ffbi +FQp2+ 7HKQ[m2biBQMbfk8R3j3dkf+ M@D p @?2HT@K2@ #QBH2`TH i2@+Q/2.@.BM2@2[m Hb 4.3.2

[266] Jake Wharton. Butterknife. ?iiTb,ffD F2r?:;`BQ?mB#Qf#mii2`F.MB.7 4.3.2

[267] Ronald J. Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. L2m` H *Q,KT2m2`, 1:270–280, 1989. doi: 10:1162/neco:1989:1:2:270. URL ?iiTb,ff/:QB;fRyRekfMR2N9Nkd.y 6.1

[268] Alexandra Wood, Micah Altman, Aaron Bembenek, Mark Bun, Marco Gaboardi, James Honaker, Kobbi Nissim, David R O'Brien, Thomas Steinke, and Salil Vadhan. Differential privacy: A primer for a non-technical audience. o M/2`#BHi DQm`M H Q7 2Mi2`i BMK i2+?MQH,Q, 21(1):209–, 2018. ISSN 1942-678X. 3.2.3

[269] Yuhao Wu, Shaowei Wang, Cor-Paul Bezemer, and Katsuro Inoue. How do developers utilize source code from stack overflow? 1KTB`B+ H aQ7ir `2, 12M, 3B–67, 2019. 7.8

[270] Xin Xia, Lingfeng Bao, David Lo, Pavneet Singh Kochhar, Ahmed E Hassan, and Zhenchang Xing. What do developers search for on the web? 1KTB`B+ H aQ7ir `2, 1M;BM22` 22:3149–3185, 2017. 1, 7.1, 7.6.2, 8.1

[271] Tao Xiao, Christoph Treude, Hideaki Hata, and Kenichi Matsumoto. Devgpt: Studying developer-chatgpt conversations. *Q_,_ abs/2309.03914, 2023. doi: 10:48550/ARXIV:2309:03914. URL ?iiTb,ff/:QB;fR9388yf `:kBpNyjNR9 2.1

[272] Frank F Xu, Uri Alon, Graham Neubig, and Vincent J Hellendoorn. A Systematic Evaluation of Large Language Models of Code. `sBp, 2022. doi: 10:48550/arxiv:2202:13169. 1.1.2, 2.4

[273] Navid Yaghmazadeh, Yuepeng Wang, Isil Dillig, and Thomas Dillig. Sqlizer: query synthesis from natural language. S`Q+X *J S`Q;` K, 1(OOPSLA):63:1–63:26, 2017. doi: 10:1145/3133887. URL ?iiTb,ff/:QB;fRyR98fjRjj 6.3 d

[274] Navid Yaghmazadeh, Xinyu Wang, and Isil Dillig. Automated migration of hierarchical data to relational tables using programming-by-example. S`Q+X oG." 1, 11(OOPSLA):580–593, 2018. doi: 10:1145/3187009:3177735. URL ?iiT,ffrrpH/#Q`;fTpH/#fpQHRRfT83y@ v ;?K x /2?. 7.6.2

[275] Litao Yan, Alyssa Hwang, Zhiyuan Wu, and Andrew Head. Ivie: Lightweight anchored explanations of just-generated code. `sBp T`2T`BMi `sBp, 2024. 9.2 yk9NR

[276] Yuan Yao, Deming Ye, Peng Li, Xu Han, Yankai Lin, Zhenghao Liu, Zhiyuan Liu, Lixin Huang, Jie Zhou, and Maosong Sun. DocRED: A large-scale document-level relation extraction dataset. In S`Q+22/BM;b Q7 i?2 8di? MMm H J22iBM; Q7 i?2 bbQ Tmi iBQM H G,BMm76b77B.+b, pages 764–777. Association for Computational Linguistics, 2019. 5.4.1

[277] Xing Yi, Liangjie Hong, Erheng Zhong, Nanthan Nan Liu, and Suju Rajan. Beyond clicks: Dwell time for personalization. In S`Q+22/BM;b Q7 i?2 3i? *J *QM72`2M+2 QM _2 avbi2 RecSys '14, pages 113–120, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450326681. doi: 10:1145/2645710:2645724. URL ?iiTb,ff/:QB;f R:yR R98fke9k8e9R8ydk.9 k.9.1, 3.2.2, 3.5.3

[278] Peifeng Yin, Ping Luo, Wang-Chien Lee, and Min Wang. Silence is also evidence: interpreting dwell time for recommendation from psychological perspective. In Inderjit S. Dhillon, Yehuda Koren, Rayid Ghani, Ted E. Senator, Paul Bradley, Rajesh Parekh, Jingrui He, Robert L. Grossman, and Ramasamy Uthurusamy, editors, h?2 RNi? *J aA:E.. AM@ i2`M iBQM H *QM72`2M+2 QM EMQrH2/;2 .Bb+Qp2`v M/ . i JBMBM AG-la - m;mbi RR,@Ry9 98k9R9j ACM, 2013. doi: 10:1145/2487575:2487663. URL ?iiTb,ff/:QB;f RyR98fk93k9R8e.j 1.1, 3.5.3

[279] Pengcheng Yin and Graham Neubig. A syntactic neural model for general-purpose code generation. pages 440–450, 2017. doi: 10:18653/v1/P17-1041. URL ?iiTb,ff/:QB;f R:yR3e8jfpRfSR.d6@ Ry9R

[280] J. D. Zamfirescu-Pereira, Richmond Y. Wong, Bjoern Hartmann, and Qian Yang. Why johnny can't prompt: How non-ai experts try (and fail) to design LLM prompts. In Albrecht Schmidt, Kaisa Väänänen, Tesh Goyal, Per Ola Kristensson, Anicia Peters, Stefanie Mueller, Julie R. Williamson, and Max L. Wilson, editors, S`Q+22/BM;b Q7 i?2 kykj *>A *QM72`2M+2 QM >mK M 6 +iQ`b BM *QKTmiBM; avbi2Kb- *>A kykj- T`BH kj@k,3 kyk437:1–437:21. ACM, 2023. doi: 10:1145/3544548:3581388. URL ?iiTb,ff/:QB;f RyR98fj89:98 9Rj.37 4.1, 8.1

[281] Tianyi Zhang, Ganesha Upadhyaya, Anastasia Reinhardt, Hridesh Rajan, and Miryung Kim. Are code examples on an online q&a forum reliable?: A study of api misuse on stack overflow. In AMi2`M iBQM H *QM72`2M+2 QM (aQ7 1r `2 ICSE, pages 886–896. IEEE, 2018. 2.4, 4.1, 5.1, 5.5, 6.8, 7.8

[282] Tianyi Zhang, Björn Hartmann, Miryung Kim, and Elena L. Glassman. Enabling data-driven API design with community usage data: A need-finding study. In Regina Bernhaupt, Florian 'Floyd' Mueller, David Verweij, Josh Andres, Joanna McGrenere, Andy Cockburn, Ignacio Avellino, Alix Goguey, Pernille Bjøn, Shengdong Zhao, Briane Paul Samson, and Rafal Kocielnik, editors, *>A'ky, *>A *QM72`2M+2 QM >mK M 6 +iQ *QKTmiBM; avbi2Kb- >QMQHmHm- >A, pages 1–13. ACM, 2020. doi: kyky 10:1145/3313831:3376382. URL ?iiTb,ff/:QB;f RyR98fjjR:jj3 jRej3k 3.1

[283] Sha Zhao, Julian Ramos, Jianrong Tao, Ziwen Jiang, Shijian Li, Zhaohui Wu, Gang Pan, and Anind K. Dey. Discovering different kinds of smartphone users through their application usage behaviors. In Paul Lukowicz, Antonio Krüger, Andreas Bulling, Youn-Kyung Lim, and Shwetak N. Patel, editors, S`Q+22/BM;b Q7 i?2 kyRe *J AMi2`M iBQM *QM72`2M+2 QM S2`p bBp2 M/ l#B[mBiQmb *QKTmiBM;- l#B*QKT K Mv- a2Ti2K#2` Rk@Re498-509. ACM, 2016. doi: 10:1145/2971648:2971696.

URL ?iiTb,ff/:QB;fRRyR98fkNdkRNeo9R3.e3N.2, 3.3.2

[284] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, /pM+2bBML2m`HAM7Q`KiBQMS`Q+2bbBM;avbi2Kbje, MMm AM7Q`KiBQMS`Q+2bbBM;avbi2Kbkykj-L2m`ASakykj-L2rP`H2M Ry@Re-, 2023 kykj URL ?iiT,ffT T:2MB:+bfT T2`n7BH2bfT T2`fkykjf?b?f NR7R3Rk3d#jN3/jd327kk8y8#79R3jk@#bi`+i@.:?iik2ibn M/n"2M+ 8.3.1, 8.3.2

[285] Albert Ziegler, Eirini Kalliamvakou, Shawn Simister, Ganesh Sittampalam, Alice Li, Andrew Rice, Devon Rifkin, and Edward Aftandilian. Productivity Assessment of Neural Code Completion. `sB, 2022. doi: 10:48550/arxiv:2205:06537. 2.4, 7.1