Addressing Compiler Errors: Stack Overflow or Large Language Models?

Patricia Widjojo^a, Christoph Treude^a

^aUniversity of Melbourne, Parkville, 3010, Victoria, Australia

Abstract

Compiler error messages serve as an initial resource for programmers dealing with compilation errors. However, previous studies indicate that they often lack sufficient targeted information to resolve code issues. Consequently, programmers typically rely on their own research to fix errors. Historically, Stack Overflow has been the primary resource for such information, but recent advances in large language models offer alternatives. This study systematically examines 100 compiler error messages from three sources to determine the most effective approach for programmers encountering compiler errors. Factors considered include Stack Overflow search methods and the impact of model version and prompt phrasing when using large language models. The results reveal that GPT-4 outperforms Stack Overflow in explaining compiler error messages, the effectiveness of adding code snippets to Stack Overflow searches depends on the search method, and results for Stack Overflow differ significantly between Google and StackExchange API searches. Furthermore, GPT-4 surpasses GPT-3.5, with "How to fix" prompts yielding superior outcomes to "What does this error mean" prompts. These results offer valuable guidance for programmers seeking assistance with compiler error messages, underscoring the transformative potential of advanced large language models like GPT-4 in debugging and opening new avenues of exploration for researchers in AI-assisted programming.

Keywords: Compiler errors, Stack Overflow, large language models

1. Introduction

Compiler error messages, designed to guide error resolution, have previously been described as "difficult to resolve" [51], "not very helpful" [52], and even "useless" [55]. In fact, a previous study conducted on software engineering students using eye tracking revealed that participants spent between 13 and 25% of their total task time reading error messages [6], which may suggest inadequacy [38] of standard compiler error messages. As a result, developers from novices to experts often rely on external sources during the debugging process [20].

In particular, the Q&A website Stack Overflow (SO) has often been mentioned as the go-to for programmers searching for coding help including on "why something is failing" [44], explanations for exceptions [56], and to repair bugs [31]. Where official documentation is inadequate, Stack Overflow has "often become a substitute for [it]" [7], and anecdotally reported as having "replaced web search and forums as [users'] primary resource for programming problems" [34].

However, there exist challenges when searching for Stack Overflow posts relevant to a user's current compiler error. Error messages are "not standardised for searching related documentation" [20], the same error may result in different messages and, conversely, the same error message can be produced by "entirely different and distinct errors" [35]. This means that it can be difficult to pinpoint the underlying cause and consequently to create the right Stack Overflow query. Eventually, this may lead to gaps between the user's intention and the textual query and poor search results [44]. Additionally, there are many cases where multiple people post what is essentially the same question with different words [19]. This, compounded by the sheer number of posts available in Stack Overflow, makes it harder to efficiently locate the information the user is seeking [31, 14].

More recently, the development and advancement of large language models (LLMs), such as GPT-4 [40], have transformed various aspects of technology, including programming and debugging [47]. These models have shown the ability to understand natural language, code comprehension, and provide relevant context-sensitive information [11]. As a result, they offer a promising alternative to traditional resources such as SO to address compiler error messages, despite their inability to provide references for their claims and occasional hallucinations [49], which led to ChatGPT's ban on SO [23]. Using the knowledge embedded in these AI models, programmers can potentially

access more targeted and precise solutions to their debugging challenges. Continuous improvement and expansion of LLMs could potentially revolutionise the way programmers approach and resolve compiler errors.

Our study aims to guide programmers in effectively addressing compiler error messages using two popular tools available right now, SO and LLMs. This paper systematically compares the use of SO and LLMs. We analyse a total of 100 compiler error messages and their context from three sources, aiming to answer key questions such as the optimal search method on SO and the impact of model version and question phrasing when using LLMs. With our first research question, we ask:

RQ1 How effective are Stack Overflow and large language models at explaining compiler errors?

We find that the LLMs considered in this study, GPT-3.5 and GPT-4, consistently outperform SO in providing explanations for compiler error messages. Additionally, we identify notable performance differences depending on factors such as the inclusion of code snippets in the search rather than just the error message, the method used to access SO content (Google or the StackExchange (SE) API), and the specific version of the LLM employed. To gain a deeper understanding of these aspects and their implications, we conducted a detailed investigation in RQs 2 and 3, examining potential strategies for optimising the debugging process using both SO and LLMs:

RQ2 To what extent does the query configuration influence Stack Overflow's ability to explain compiler errors?

When seeking assistance with compiler errors on SO, programmers face several decisions: whether to access SO through its own services or a general-purpose search engine like Google, whether to include the offending code in addition to the error message (and if so, how much code to include and whether to remove identifier names that are unlikely to be matched on SO), and whether to focus on accepted or highly voted answers. To answer our second research question, we systematically investigated the impact of these alternatives by designing 72 search strategies as unique combinations of these choices and evaluating their results.

Our findings indicate that when searching SO on Google, omitting code snippets yielded the highest percentage of relevant first answers. In contrast, direct SO searches benefited from the inclusion of code snippets, resulting in increased relevance of results, albeit with the trade-off of fewer results being returned. To measure the similarity of up to the first 10 links returned from a query, we calculate Rank-Biased Overlap (RBO) values [54]. Our analysis shows that searches using Google and the StackExchange (SE) API return entirely different results.

The type of code snippets added to the query significantly affected the similarity of the returned results. By simply removing or replacing the identifiers within the code snippets with "x", the RBO with the original query generally decreased to below 0.5. Interestingly, filtering for posts with positive and/or accepted results did not have a notable impact on the overall effectiveness of SO queries. In fact, filtering for results with an accepted answer often led to less relevant first answers.

RQ3 To what extent does the query configuration influence the ability of large language models to explain compiler errors?

Programmers also face similar decisions when turning to LLMs for help explaining compiler errors: Does the choice of LLM version (e.g., GPT-3.5 vs. GPT-4) matter, particularly considering the current cost of GPT-4 and initial evidence from other domains suggesting that ChatGPT-4 does not necessarily outperform ChatGPT-3.5 in specific tasks [27]? Do the LLM explanations improve if the offending code is included in the prompt? And what is the most effective way to communicate with the LLM? To address our third research question, we systematically investigate the impact of these alternatives by designing eight prompting strategies as unique combinations of these choices and evaluating their results.

Our findings reveal that GPT-4 outperforms GPT-3.5. When provided with the offending code snippet and the corresponding error message, GPT-4 successfully explains all 100 errors in our dataset, regardless of whether it is asked what the error means or how to fix it. This represents a significant improvement over GPT-3.5, which produced suitable answers in 87% of the cases for the prompt "What does the error mean?" and 91% for "How can I fix the error?". The trend of the "How can I fix the error?" prompt leading to more useful results also applies when the LLMs are prompted with only the error message (i.e., without the offending code). GPT-4 produced a suitable response in 84% of the cases, while GPT-3.5 did so in 75%. This is both higher than 82% and 72% received from their respective "What" queries.

These results demonstrate that within a short time span (ChatGPT/GPT-3.5 was released on November 30, 2022, and GPT-4 on March 14, 2023), the models' capabilities in explaining compiler error messages have improved significantly. In particular, the phrasing of the prompt becomes less critical as long as the prompt includes both the offending code and the error message.

In summary, in this paper, we:

- Conduct a systematic comparison between SO and LLMs (GPT-3.5 and GPT-4) to explain compiler error messages, demonstrating that GPT-4 outperforms both GPT-3.5 and SO in explaining compiler errors, with the inclusion of offending code significantly improving results.
- Investigate the impact of query configuration on SO's ability to provide relevant answers, considering 72 search strategies and their unique combinations.
- Assess the influence of prompt configuration on the ability of LLMs to explain compiler errors using eight prompting strategies.
- Discuss the practical implications of our work for both programmers and researchers.

2. Related Work

Programmers frequently conduct search sessions while coding, reflecting their reliance on external resources for assistance during development [44]. They tend to prefer general-purpose search engines, particularly Google, over code-specific search engines [56, 24, 43, 30]. However, search engines often serve as tools that lead users to other sites, such as SO, CSDN, or ZhiHu, where answers are primarily located [56].

2.1. Stack Overflow as a Primary Resource

SO has become an essential resource for programming help, with a large number of questions posted on various topics [44]. It often serves as a substitute when official documentation is inadequate, providing solutions to common programming problems [7]. However, despite its popularity, finding relevant SO posts can be challenging due to non-standardised error messages [20, 35] and the sheer volume of posts, along with duplicated questions [31, 19, 14].

Developers often paste error logs directly into search boxes when looking for answers [14], which may not be the most effective approach, as code queries are more complex and often reformulated or edited [44, 43]. They usually rely on Google to find software resources on SO [25, 16] and focus on accepted answers when looking at SO posts [30, 15, 12], even though these are not always the most voted-for answers.

Improving search results for error messages has been studied by various researchers, such as Hora, who conducted an empirical study of developer search queries [25], Monperrus and Maia, who focused on JavaScript code snippets in queries [37], Barzilay et al., who explored the types of questions asked and answered on SO [7], Xia et al., who observed developers' use of search engines for code and error explanations [56], and Li et al., who showed that both novice and expert programmers use SO for debugging [30]. Our research seeks to identify effective query types for SO, with the goal of reducing the need for reformulation and improving the overall coding experience.

Enhancing compiler error messages has been a focus of previous research, including studies on plugins that add summarised information from SO [46, 50] and approaches that improve compiler error messages specifically for Java without using SO [53, 26, 8]. Our study complements these findings by testing different factors and assessing the relevance of the results to the original compiler error.

2.2. Compiler Error Help from Large Language Models

Artificial intelligence and machine learning have been applied to locate and repair bugs in source code with promising results. DeepFix [21], a multi-layered sequence-to-sequence neural network with attention, partially fixed almost half of a set of programming tasks by predicting erroneous program locations and providing correct statements. Similar results were achieved with a reinforcement learning approach [22]. Santos et al. [45] used n-gram and LSTM language models to locate syntax errors and suggest fixes, achieving similar results. More recently, SYNFIX [5], which uses unsupervised pretraining and multi-label classification, outperformed previous methods such as DeepFix and Santos et al.

Despite these automatic error-fixing approaches, there has been little prior research on using LLMs to explain compiler error messages. In a recent study based on Codex [17], Leinonen et al. [29] found that the model produced explanations of error messages that were "quite comprehensible", but correct in only about half of all cases. Codex's fix suggestions were also

deemed correct in a similar number of cases. To the best of our knowledge, the potential of GPT-4 for explaining compiler error messages has not yet been studied.

3. Methodology

3.1. Source Code Collection

To cover the varying use cases of SO and LLMs when addressing different compiler errors, we gathered Java source code with compiler errors from multiple sources, totaling 100 pieces. First, we randomly obtained 55 pieces of code from the Blackbox dataset [13]. The Blackbox dataset comprises activity data from Blue JIDE users [53] who consented to have their activity recorded for research purposes. Using these pieces of code, we consider real-world instances where developers encountered compiler error messages during their coding projects. Additionally, we expanded the range of errors by creating 18 sets of code with unique error types not yet encountered in the BlackBox data. These pieces of code were designed to fail due to frequently occurring Java compiler errors, as documented in [26, 8], made loosely based on [4, 48], and had not been encountered in the first batch of snippets from Blackbox (code number 1-35). Lastly, 27 pieces of source code were sourced from a separate participant study conducted as part of another project by the first author. This project also focused on compiler errors, and the code was collected when three participants with diverse backgrounds and levels of Java experience completed exercises from code-exercises.com [42]. In the results and discussion sections, we will refer to these three sources of Java code as BLACKBOX, CUSTOM, and USER STUDY, respectively.

We sourced code snippets from three origins to increase confidence in generalisability and to ensure our dataset accurately represents a variety of real-world coding errors. As the codes were retrieved from singular files as opposed to a project with multiple files, the length of each is relatively short with a median Lines of Code (LOC) of 16. The complete LOC distribution is shown in Figure 1.

3.2. Evaluation Methodology

To analyse the impact of various factors on the overall effectiveness of SO and LLM queries, we evaluate the results obtained under different configurations. We define effectiveness as the degree of assistance provided by a result in resolving the first compiler error encountered in a Java program.

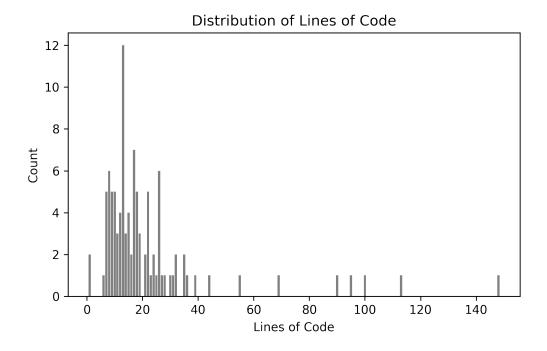


Figure 1: Lines of Code (LOC) distribution of code snippets

For SO queries, we classified the resulting initial posts into four distinct relevance categories, as detailed in Table 1. This rating scale was proposed and used by Mahajan et al. [33] in measuring the relevance of SO posts, and was based on [9, 32, 28]. We refer to this categorisation as the IHMU-category.

During the classification of the results, we considered the entire content of the page to mimic real-world usage, where developers are more likely to focus on the text of the post rather than just the titles [41, 30]. The first author conducted this manual annotation. A cross-check with the second author annotating a subsection (10%) of the results was also performed. The Cohen's Kappa [36] coefficient was then calculated to measure inter-rater reliability, ensuring that the overall annotation bias is acceptable and that the results can be considered valid. We obtained a Cohen's Kappa value of 0.860, indicating an agreement level of "almost perfect".

After completing all necessary SO annotations, we followed Mahajan et al.'s evaluation methodology and calculated three metrics: I-Score, IH-Score,

Table 1: IHMU-category relevancy classification

Classification	Definition
Instrumental (I)	Post relates to the underlying cause of the first compiler error perfectly and provides a complete fix
Helpful (H)	Post is generally informative about the compiler error but may not specifically target the underlying cause or provide an effective fix
Misleading (M)	Post is irrelevant to the compiler error.
Unavailable (U)	No results were returned from the query

and M-Score. Additionally, we introduced an IH/M-Score, which measures the percentage of instrumental and helpful results relative to the number of misleading results. This metric helps evaluate whether an increase in relevant results is accompanied by a trade-off of also increasing misleading results. These metrics and their respective formulas are listed in Table 2.

Table 2: Formulas used in measuring result relevancy

Classification	Definition
I-Score	$\frac{I}{I+H+U+M} \times 100\%$
IH-Score	$\frac{I+H}{I+H+U+M} \times 100\%$
M-Score	$\frac{M}{I+H+U+M} \times 100\%$
IH/M-Score	$\frac{I+H}{M} \times 100\%$

Moreover, we measured the similarity between the results of different SO combinations. To achieve this, we calculated the Rank-Biased Overlap (RBO) [54] of the top 10 links for each combination in the 100 scenarios. An RBO score of 1 indicates a completely "identical" list, while a score of 0 indicates a completely "different" list [54]. We then computed the median of these scores and present the corresponding heatmap.

The evaluation of the results obtained from LLMs followed a similar methodology to that of SO, with a few adjustments. Since ChatGPT, as a chatbot on top of GPT-3.5 and GPT-4, virtually always returns results that are or at least appear relevant when provided with error messages, we modified the IHMU-category to exclude the "Unavailable" category and more

narrowly defined the other categories, as shown in Table 3. We did not calculate similarities through RBO between the results obtained from LLMs as part of this study.

Table 3: Adjusted IHM-category relevancy classification

Classification	Definition
Instrumental (I)	Response perfectly targets underlying cause of error and provides a clear action on how to fix
Helpful (H)	Response provides general but not exact help. E.g. listing all potential common causes of error along with multiple fixes from which the coder would still have to analyse
Misleading (M)	Response does not provide a clear direction on how to fix the issue and/or causes confusion. E.g. irrelevant results, right code snippet however with a completely wrong explanation, returning other common causes without the root cause, and responses that provides fixes to errors in different parts of the code but would not fix the first compiler error.

3.3. Stack Overflow Combinations

Given the numerous ways to search SO, we conducted experiments with various query approaches to identify the most effective methods for obtaining relevant results. Each query corresponded to a different combination of query method, post filter, and query content, resulting in a total of $2 \times 3 \times 12 = 72$ unique combinations. These are described in Table 4, with additional definitions and clarifications outlined below.

First, to obtain results with "searching directly through SO" (SO-API) as a search method, we used the /search functionality of the SE API [2], with the corresponding queries saved under the "intitle" argument. We employed this method because of the absence of a full-text search option.

The Custom query content treatment refers to queries in which we replaced code specific identifiers from the original compiler error message with a more generic variable or removed it altogether. This modification was implemented for some common messages to make them more generic. For example, we replaced the variable and method name in the "variable [variable name] is already defined in method [method name]" errors, and the class name in the "class [class name] is public, should be declared in a file named [class name].java" errors with generic placeholders "X" and "Y".

Table 4: Description of the identifiers used for SO via API and SO via Google queries

Query Method							
Identifier	Query Search Method						
SO-API	Search request through StackExchange API as a proxy for searching Stack Overflow directly						
SO-Google	Search via Google with "site: stackoverflow" included in query						
	F	Post Filter					
Identifier	Query Result Filter						
First	Take first result (Stack Overflo	w post) without any filtering					
Positive	Take first result (Stack Overflo	w post) with a positively rated answer					
ACCEPTED	Take first result (Stack Overflo	w post) with an accepted answer					
	Qu	ery Content					
Identifier	Description	Example					
CUSTOM/ LINE	$\begin{array}{c} {\rm custom\ error\ message} + {\rm original\ error\ line} \\ \end{array}$	$\label{eq:context} \begin{array}{l} \text{non-static method cannot be referenced from a static context} \\ \text{System.out.println}(\text{reverse}(s)); \end{array}$					
CUSTOM/ LINEBLANKID	$\begin{array}{c} {\rm custom~error~message} + {\rm error} \\ {\rm line~with~identifiers~replaced} \\ {\rm with~blanks} \end{array}$	non-static method cannot be referenced from a static context $(());$					
CUSTOM/ NOSNIPPET	custom errror message only	non-static method cannot be referenced from a static context					
Custom/ Parent	custom error message + parent node of error line	non-static method cannot be referenced from a static context [public static void main(), String $s=$ "java interview";, System.out.println());,]					
CUSTOM/ PARENTBLANKID	custom error message $+$ parent node with identifiers replaced with blanks	non-static method cannot be referenced from a static context public static void main() String = "java interview";());					
CUSTOM/ PARENTXID	custom error message $+$ parent node with identifiers replaced with "x" non-static method cannot be referenced from a static cutter text public static void main() String $x = $ "java interview $x.x.x()$;						
ORIGINAL/ LINE	original error message $+$ original error line	non-static method reverse(String) cannot be referenced from a static context System.out.println(reverse(s));					
Original/ LineBlankId	original error message $+$ error line with identifiers replaced with blanks	non-static method reverse (String) cannot be referenced from a static context(());					
ORIGINAL/ NoSnippet	original error message only	$\begin{array}{c} \text{non-static method reverse}(\text{String}) \text{ cannot be referenced from} \\ \text{a static context} \end{array}$					
Original/ Parent	original error message + parent node of error line	$\label{eq:context} \begin{split} & \text{non-static method reverse}(String) \ cannot \ be \ referenced \ from \\ & a \ static \ context \ [\ public \ static \ void \ main(), String \ s = "java \ interview";, System.out.println());,] \end{split}$					
Original/ ParentBlankId	original error message $+$ parent node with identifiers replaced with blanks	non-static method reverse (String) cannot be referenced from a static context public static void main () String = "java interview";());					
ORIGINAL/ PARENTXID	original error message $+$ parent node with identifiers replaced with "x"	non-static method reverse (String) cannot be referenced from a static context public static void main () String $x=$ "java interview"; $x.x.x()$);					

Under query content, the "parent node" was obtained by creating a concrete syntax tree of the Java code snippet using the parser library tree-sitter [3] and then saving the parent node of the line where the compiler error occurred. The same parser library was used to identify parts of the code labelled as identifiers, which were used in defining LINEBLANKID, PARENT-BLANKID, and PARENTXID.

We tested each of these 72 combinations on the 100 Java codes as described in the Source Code Collection section, yielding a total of $72 \times 100 = 7,200$ query runs. From each run, we saved up to the first 10 SO links returned for further analysis of relevance and similarity.

To facilitate data collection, we created a Sublime Text 4 plugin. When called, the plugin automatically executes each of the 72 combinations described in Table 4 and logs the details related to each run. This information included the combination to which it corresponded, the source code on which it was run, the total number of results returned, and up to the first 10 SO links it returned. The total number of results returned for queries run on SO-API was simply counted as the number of items in the JSON file it returned, whereas for Google SO-GOOGLE queries, we scraped the resulting page to find the div element with the id "result-stats" from which the relevant value was saved. This represents the part "About [number] results" found in Google search results. When errors occurred while scraping the page, we placed a placeholder value of 0 instead. This placeholder value is not expected to affect the results, as the percentage of error occurrence was negligible at only 0.5%. We used the plugin on each of the 100 Java pieces of code with compiler errors as previously described.

Compiling the first links from all 7,200 query runs yielded a total of 573 unique pairs of source code and the corresponding first link. The relatively lower number of unique pairs was due to several combinations returning the same top link and many combinations, particularly those with long code snippets, not returning any results.

3.4. Large Language Model Combinations

To address research questions related to LLMs, we examined the impact of including the source code in the query versus only providing the error message, the influence of prompt phrasing on the relevance of the results, and the differences between using the "Default (GPT-3.5)" model and the newer "GPT-4" version. We selected the prompts "What does the error mean?"

Table 5: Description of the identifiers used for LLM queries

	Query Method							
T.1	• •							
Identifier	Query Search Method							
GPT-3.5	Query on https://chat.openai.c	com/ with "Default (GPT-3.5)" model						
GPT-4	Query on https://chat.openai.c	com/ with "GPT-4" model						
	Que	ery Phrasing						
Identifier	Query Phrasing							
What	Query content asked alongside	"What does the error mean?"						
How	Query content asked alongside "How can I fix the error?"							
	Query Content							
Identifier	Description	Example						
	original error message + full	test.java:5: error: cannot find symbol public class test public						
Original/	Java snippet from which the static void main(String args) System.ouch.println("Hello,							
FULLSNIPPET	first error message is thrown.	World!");						
Original/	original error only	test.java:5: error: cannot find symbol						
NoSnippet		y						

and "How can I fix the error?" based on experiments with GPT-3.5. Table 5 enumerates the $2 \times 2 \times 2 = 8$ LLM query combinations.

We collected LLM data by pasting each of the 100 error messages with and without code snippet on the ChatGPT page¹ along with the one of the two prompts. To maintain consistency, only the first responses i.e. no regeneration of the 800 queries were saved and evaluated.

3.5. Data Availability

We have made our data and scripts available in our online repository². This includes an Excel file containing the 100 code snippets tested along with their corresponding compiler errors, resulting SO links, LLM responses, authors' annotations, and resulting tables. Additionally, the two notebooks used to calculate the median number of results and RBOs, as well as the Sublime plugin created to obtain links based on the 72 combinations related to SO, have been made available.

4. Results

In this section, we present and discuss answers to our research questions. An important factor to consider is that SO-API queries frequently returned

¹https://chat.openai.com

²https://github.com/patwdj/java-compiler-error-help

no results, affecting the results. Out of 100 scenarios, the average number of times SE queries yielded no results was 83.1. When queries without code snippets are excluded, empty results become even more prominent, averaging about 91.1%. On the contrary, the average empty results for SO-GOGLE queries were significantly lower at 23.7% in general or 27.8% when excluding queries without code snippets.

Table 6 presents the median number of results for the 100 compiler errors when using different query combinations. Since the median values for SO-API queries are mostly 0, and LLMs provide one response per query, the discussion about the number of results will mainly focus on SO-GOOGLE.

Table 6: Median number of results returned

Identifier	Median
SO-GOOGLE/CUSTOM/NOSNIPPET	36350
SO-GOOGLE/ORIGINAL/NOSNIPPET	32400
SO-GOOGLE/CUSTOM/LINEBLANKID	2570
SO-GOOGLE/ORIGINAL/LINEBLANKID	2360
SO-GOOGLE/CUSTOM/LINEXID	996.5
SO-GOOGLE/ORIGINAL/LINEXID	813.5
SO-GOOGLE/ORIGINAL/LINE	471
SO-GOOGLE/CUSTOM/LINE	372.5
SO-GOOGLE/CUSTOM/PARENTBLANKID	163.5
SO-GOOGLE/ORIGINAL/PARENTBLANKID	8
SO-GOOGLE/CUSTOM/PARENT	6.5
SO-GOOGLE/ORIGINAL/PARENT	6
SO-API/CUSTOM/NOSNIPPET	3.5
SO-API/ORIGINAL/NOSNIPPET	2
SO-API/	0

More than the sheer number of results, when using SO or LLMs to resolve compiler errors, the relevancy of a post or response to the specific issue is important. We have summarised the values of the IHM(U) categories, as defined in Table 1 and 5, in Table 7. This summary only includes a subset of the combinations tested and will be discussed in relation to RQ1. A more detailed breakdown of other SO-GOOGLE and SO-API combinations, along with their corresponding metrics, will be presented as part of the RQ2 results, while GPT-3.5 and GPT-4 will be discussed as part of RQ3.

Table 7: Summary of Results

Search (all with original error message)	IH	M	U
SO-API/NoSnippet	15	40	45
SO-Google/NoSnippet	57	41	2
SO-API/PARENT	4	0	96
SO-GOOGLE/PARENT	22	48	30
GPT-3.5/What/NoSnippet	72	28	-
GPT-4/What/NoSnippet	82	18	-
GPT-3.5/What/FullSnippet	87	13	-
GPT-4/What/FullSnippet	100	0	-

[RQ1] How effective are Stack Overflow and large language models at explaining compiler errors?

Table 7 shows that SO queries exhibit mixed results in their ability to help explain compiler errors when searched directly through the API and when searched through Google. Without code snippets, searching through Google is more promising, with 57% IH compared to SO-API/NOSNIPPET's 15%. The high number of misleading compared to the relevant relevant results is also particularly concerning for SO-API/NOSNIPPET. In contrast, when the parent node of the error line is included in the query, the trend reverses: SO-API/PARENT does not return misleading results, compared to over 40% found by its SO-GOOGLE counterpart. However, although SO-API/PARENT did not give misleading results, it only returned results in 4 of the 100 scenarios.

In terms of seeking help for Java compiler errors, LLMs proves to be notably more effective than SO. LLMs consistently return a result with a higher relevancy rate than our SO searches. Specifically, when using GPT-4 and providing the full Java code snippet along with the original error message, a relevancy rate of 100% is achieved. This represents an improvement over the older GPT-3.5 version, which achieved a relevancy rate of 87% for the same queries. A consistent trend observed in both versions is that adding the full code snippet to the query positively impacts the results. In scenarios with or without code snippets in the query, LLM's results consistently outperform SO's.

[RQ2] To what extent does the query configuration influence Stack Over-flow's ability to explain compiler errors?

We evaluated 72 different SO query combinations. The percentage of results for the different combinations based on their respective IHMU categories is shown in Figure 2. The corresponding metrics can be found in Table 8.

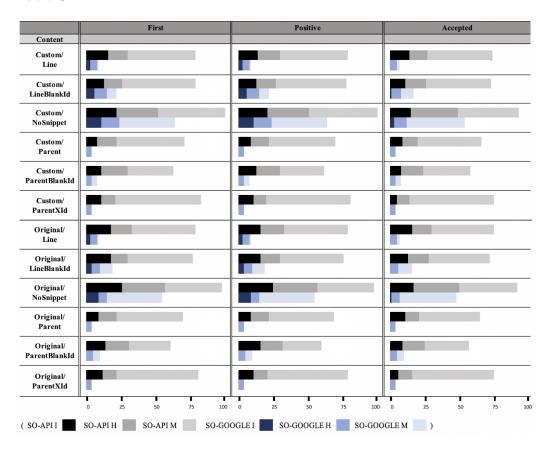


Figure 2: IHMU-category of resulting first links

Table 8: Calculated relevancy metrics

					۰							
		First	First (Scores)	<u> </u>	_	Positive (Scores)	Scor	es)	⋖	Accepted (Scores)	d (Scor	es)
	I	IH	M	M/H	Ι	ΗI	M	$_{ m IH/M}$	Ι	IH	\mathbf{M}	IH/M
SO-GOOGLE/CUSTOM/LINE	16%	30%	49%	61%	14%	30%	49%	61%	14%	27%	47%	27%
SO-API/Custom/Line	3%	8%	1%	800%	3%	8%	1%	%008	%0	2%	2%	250%
SO-GOOGLE/CUSTOM/LINEBLANKID	13%	36%	53%	49%	13%	27%	51%	53%	111%	26%	47%	25%
SO-API/Custom/LineBlankId	%9	15%	2%	214%	%9	15%	2%	214%	1%	8%	%6	%68
SO-GOOGLE/CUSTOM/NOSNIPPET	22%	52%	48%	108%	21%	51%	49%	104%	15%	49%	44%	1111%
SO-API/Custom/NoSnippet	111%	24%	40%	%09	111%	24%	40%	%09	3%	12%	42%	29%
SO-GOOGLE/CUSTOM/PARENT	8%	22%	49%	45%	%6	22%	48%	46%	%6	20%	46%	43%
SO-API/Custom/Parent	%0	4%	%0	inf	%0	4%	%0	inf	%0	4%	%0	inf
SO-GOOGLE/CUSTOM/PARENTBLANKID	11%	30%	33%	91%	13%	30%	32%	94%	8%	24%	34%	71%
SO-API/CUSTOM/PARENTBLANKID	%0	4%	4%	100%	%0	4%	4%	100%	%0	4%	4%	100%
SO-GOOGLE/CUSTOM/PARENTXID	11%	21%	62%	34%	11%	20%	61%	33%	2%	14%	61%	23%
SO-API/Custom/ParentXID	%0	4%	%0	inf	%0	4%	%0	inf	%0	4%	%0	inf
SO-Google/Original/Line	18%	33%	46%	72%	16%	33%	46%	72%	16%	30%	45%	%29
SO-API/Original/Line	3%	8%	1%	800%	3%	8%	1%	%008	%0	2%	2%	250%
SO-Google/Original/LineBlankId	18%	30%	47%	64%	16%	30%	46%	829	13%	28%	44%	64%
SO-API/Original/LineBlankId	4%	10%	%6	111%	4%	10%	%6	1111%	%0	%9	10%	%09
SO-Google/Original/NoSnippet	%97	21%	41%	139%	32%	22%	41%	139%	17%	20%	42%	119%
SO-API/Original/LineBlankId	%6	15%	40%	38%	%6	15%	40%	38%	1%	2%	41%	17%
SO-Google/Original/Parent	%6	22%	48%	46%	%6	22%	47%	47%	11%	21%	44%	48%
SO-API/Original/Parent	%0	4%	%0	inf	%0	4%	%0	inf	%0	4%	%0	inf
SO-Google/Original/ParentBlankId	14%	31%	30%	103%	16%	32%	28%	114%	%6	25%	32%	78%
SO-API/Original/ParentBlankId	%0	2%	2%	100%	%0	2%	2%	100%	%0	2%	2%	100%
SO-Google/Original/ParentXID	12%	22%	29%	37%	11%	21%	28%	36%	%9	16%	26%	27%
SO-API/Original/ParentXID	%0	4%	%0	inf	%0	4%	%0	inf	%0	4%	%0	$_{ m inf}$

Furthermore, we calculated the median RBOs between different combinations and present them in Figure 3 and 4. Although we calculated all RBOs between Google and SE queries, we found the median to be 0 for all combinations with each other. This indicates their dissimilarity and provides a rationale for presenting the results in two separate figures.

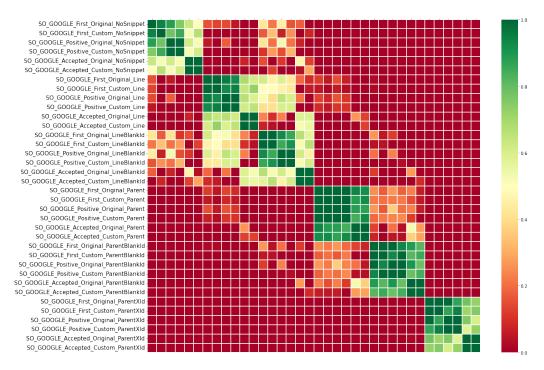


Figure 3: RBO values between Google SO-GOOGLE queries

4.0.1. Impact of Query Content

As expected, the addition of code snippets resulted in a lower median number of results. Although the median for Google queries without code snippets was more than 30,000, it was at least 10 times lower for all other combinations tested. The addition of a single line where the error was thrown reduced the median from 32,400 to 471 (-98.5%). Including the parent node of the error line instead led to a further decrease to a median value of 6.

Although the difference was not as large, removing or replacing identifiers within code snippets with "x" increased the number of results compared to their "as-is" code snippet equivalents. Interestingly, the median number was higher when the identifiers were replaced with



Figure 4: RBO values between StackExchange SO-API queries

"x" (SO-GOOGLE/PARENTXID) than when completely removed (SO-GOOGLE/PARENTBLANKID).

In terms of error message treatment, removing code specific identifiers from the error message (SO-GOOGLE/CUSTOM) resulted in a higher number of results compared to simply using the original compiler error message (SO-GOOGLE/ORIGINAL), with one exception.

Regarding relevancy, considering only the first link returned for the 100 scenarios from each combination, those returned from queries with no code snippets had the best IH-Score performance. This was observed irrespective of the query method used and whether post filtering was applied or not. The best overall results in terms of IH and IH/M-Score were achieved by both SO-GOOGLE/FIRST/ORIGINAL/NOSNIPPET and SO-GOOGLE/POSITIVE/ORIGINAL/NOSNIPPET queries at 57% and 139%, respectively.

For SE queries (SO-API), however, this came with the trade-off of a significantly higher M-Score compared to when code snippets were added. Their

M-Scores averaged around 40%, compared to other combinations, which had M-Scores between 0% and 9%. Consequently, this led to the IH/M-Score being less than 100%, meaning that most of the results were irrelevant and useless for fixing the original compiler error.

For Google queries, the inclusion of code snippets had a more varied impact. One of the more evident outcomes is that adding the parent node with identifiers replaced with "x" as the code snippet proved to be the worst addition, with only 14% to 22% IH-Score and up to 62% misleading results received from these types of queries. The addition of the unedited parent node (SO-GOOGLE/PARENT) performed slightly better, although their IH/M-Scores were still only 48% at best. On the other hand, when the parent node was added, but with the code specific identifiers removed entirely, it resulted in a significantly lower number of misleading results compared to all other (SO-GOOGLE) combinations. However, the overall IH/M-Scores remained lower than when using no code snippets.

Interestingly, this finding does not apply to SO-API queries, where the lowest M-Scores at 0% occurred when the parent node was included as is or with the identifiers replaced with "x". Instead, removal of the identifiers increased the M-Scores to 4% to 5%. Overall, adding more context to SE queries led to higher IH/M-Scores, where the reduction in misleading results had a more significant impact than the relatively smaller decrease in IH results. However, this comes with the caveat that the overall number of results from SE queries, excluding those with no code snippets, is relatively low, and most of the queries return nothing.

To analyse the similarity of the results, we examine the calculated RBO values. The inclusion or exclusion of code snippets in the query, along with the type if any was included, likely affects the similarity of the results. In Figure 3, the presence of green and yellow 6×6 squares on the diagonal indicates the relatively high correlation between the first 10 results of Google queries with the same code snippets, regardless of error message treatment and post filtering. This is most apparent between queries with the parent node included as code snippets, represented by the three green squares in the bottom right quadrant.

In contrast, there are no green boxes outside the 6×6 area that represent those with exactly the same type of code snippet. This means that removing or replacing the identifiers within the code snippets significantly changes the results, leading to lower similarity and, consequently, RBO. However, some correlation remains visible, particularly between combinations with the error

line as the code snippet (SO-GOOGLE/LINE, SO-GOOGLE/LINEBLANKID). For these, although deleting the identifiers affected the results, the impact was less than for the parent node code snippet.

There is also a difference between removing identifiers from the error line and not doing so. A correlation can still be observed between queries with the error line without identifiers SO-GOOGLE/LINEBLANKID and a plain query without a code snippet SO-GOOGLE/NOSNIPPET, albeit only within the yellow range. In contrast, the same cannot be said between queries using the error line as is SO-GOOGLE/LINE and those with no code snippet SO-GOOGLE/NOSNIPPET due to their low RBO with each other.

As mentioned earlier, SE queries often returned empty sets. Since the RBO value between two empty sets is 1, this resulted in a high overall RBO value and formed the large green square in Figure 4. The small green box in the upper left corner, on the other hand, represents the RBO values between the results when querying SE without any code snippets and only the original compiler error messages SO-API/NOSNIPPET. Similarly to Google queries, the results of these queries appear to be highly correlated with each other, and the removal of identifiers from the error message has no major impact.

4.0.2. Impact of Query Method

We discovered that the number of results is significantly affected by the query method used. The highest median number of results for SE SO-API queries was only 3.5, found when searching for the generic compiler error message with the identifiers removed without any code snippets SO-API/CUSTOM/NOSNIPPET. On the contrary, searching for the same query on Google SO-GOOGLE/CUSTOM/NOSNIPPET yielded a median of 36,350 results. When adding any kind of code snippet, regardless of whether it was the code error line or the parent snippet, the median number of SE results was 0.

Although SE queries with code snippets rarely return anything, when they do, the results are much more likely to be accurate, with IH/M-Scores above 100%. Queries with the parent node added as is or replaced with x (SO-API/PARENT, SO-API/PARENTXID) performed particularly well. Although they only returned a result four times within the 100 scenario runs, all of the first links returned were relevant.

Overall, when using SE queries with code snippets, the results are proportionally more relevant than their equivalents on Google, although the results were returned less frequently. Note that this trend does not apply to queries

without code snippets SO-API/NOSNIPPET, which have IH/M-Scores below 60%. For general searches without code snippets, Google queries had a higher percentage of relevant results, both in terms of overall numbers (IH-Score) and as a percentage over the number of misleading queries (IH/M-Score).

Taking into account the top 10 results of each query and their corresponding median RBO scores with each other, we found no similarities between the SE results and the Google results.

4.0.3. Impact of Post Filtering

Among the three factors discussed so far, post filtering arguably has the least noticeable impact on the relevancy of results. Specifically, for results obtained by filtering only for positive posts, no discernible impact can be observed compared to simply taking the first post returned. Interestingly, filtering for only posts with an accepted answer did not result in higher relevance; rather, it was lower in most combinations. Further analysis revealed that this occurred when the original first link found had a positively rated answer that correctly addressed the problem, but was not accepted as the answer. Consequently, the tool had to search further down the list of returned links to find a post with an accepted answer. In some cases, these posts may be less relevant to the original error, discussing a completely different problem and leading to a misleading result.

For Google results, most of the time, the first returned results are posts with positively rated answers. This is evidenced by the high RBO values between simply taking the first answer and filtering for positive answers (SO-GOOGLE/FIRST, SO-GOOGLE/POSITIVE) for the same query. Their RBO values range from 0.75 to 1.00, represented by the medium to dark green 4×4 boxes on the diagonal in Figure 3. Comparatively, the first results are less likely to be accepted answers, since filtering for only posts with accepted answers resulted in lower RBOs without filters. This is illustrated by the lighter green to yellow rectangles surrounding the green 4×4 boxes, with RBO values between 0.49 and 0.94.

[RQ3] To what extent does the query configuration influence the ability of large language models to explain compiler errors?

Unlike the results from Google and the SE API, where sometimes the number of results returned was 0, the LLMs provided a reply for all the scenarios we tested. However, this came with the caveat that there were

instances where the LLM produced "plausible-sounding but incorrect or non-sensical answers" [39]. In particular, there was an instance where the LLM gave the correct fix to the code, should a user choose to directly copy the recommended code fix, but provided a misleading explanation for it. Additionally, there were instances where the tool provided multiple answers, one being correct and the other misleading. Even when using the newer version, there was still a scenario where running the provided code would fix the compiler error but cause an ArrayIndexOutOfBoundsException at runtime.

Nevertheless, overall results highlighted LLM's potential, with the majority of responses received being either instrumental or helpful in fixing the queried compiler error.

As success was already seen in using LLM with full code snippets, we did not test further query alteration beyond inclusion or exclusion of whole snippets. Instead, we added a breakdown by code source to investigate whether there are particular types of code for which the LLM would work best. The results are summarised in Table 9.

4.0.1. Impact of Query Content

In all configurations and versions tested, adding the whole code snippet with compiler error as part of the query resulted in higher result relevancy. Especially in GPT-4, this led to a perfect result of 100% instrumental or helpful. When a code snippet is not available, LLMs often revert to listing common causes of the error message, which may or may not cover the underlying cause.

4.0.2. Impact of LLM Versions

GPT-4 shows a notable improvement over its predecessor. This is true for all cases except where the code used was based on common errors (CUSTOM) and the complete code snippet was provided. In this case, GPT-3.5 had already achieved perfect performance at 100%, leaving no room for improvement when using the newer version. For scenarios BLACKBOX and USER_STUDY and for the cases where no code snippets were provided, GPT-4 proved to be an advance. In particular, when the code snippet is provided (GPT-4/FULL), understanding the context and intent of the user was demonstrated, e.g., by creating a code implementation based on commented-out parts of the code beyond fixing errors. In contrast, GPT-3.5 did not take extra context into account and provided more straightforward solutions, e.g., by not changing commented-out parts of the code.

4.0.3. Impact of Query Phrasing

As OpenAI acknowledged to be one of the limitations of LLMs [39], we observed that input phrasing affected overall results returned by the tool. Among the two tested questions, "How can I fix the error?" generally performed better in terms of IH-Scores.

	Table of 111 count of farge handade insuce and south of ferror section s, course source								
ORIGINAL/FULLSNIPPET ORIGINAL/NOSNIPPET							PET		
		GP	T-4	GPT-3.5		GP	T-4	GPT	-3.5
Source	Ttl.	WHAT	How	WHAT	How	WHAT	How	WHAT	How
BLACKBOX	55	55	55	47	51	44	46	43	44
		(100%)	(100%)	(85%)	(93%)	(80%)	(84%)	(78%)	(80%)
CUSTOM	18	18	18	18	18	16	16	13	15
		(100%)	(100%)	(100%)	(100%)	(89%)	(89%)	(72%)	(83%)
USER	27	27	27	22	21	22	22	16	16
STUDY		(100%)	(100%)	(81%)	(78%)	(81%)	(81%)	(59%)	(59%)
	100	100	100	87	90	82	84	72	75
		(100%)	(100%)	(87%)	(90%)	(82%)	(84%)	(72%)	(75%)

Table 9: IH count of large language model and Stack Overflow results by code source

5. Implications

5.1. For Developers

Developers spend a significant amount of time searching the Web for help and reformulating queries [44, 43, 14]. Through our findings, we aim to improve this experience and increase overall efficiency when using LLMs or SO to find help fixing compiler errors.

When using SO, we find that searching SO via Google instead of a direct search impacts the type and number of results returned. When no code snippet is available, a Google search is considered a better approach. For queries that include only the compiler error message and NoSNIPPET, we found that Google queries had a higher percentage of helpful answers and a lower percentage of misleading results. Conversely, if a user has access to the original code, it may prove beneficial to search directly on SO while including a code snippet. We identified a notably higher percentage of helpful against misleading results for these types of queries, making this approach particularly useful for novices with less experience in gauging the relevancy and correctness of a post. However, we also note that SO-API rarely returns a result, so a Google search may still be needed in most cases. Moreover,

removing code-specific identifiers and/or replacing them with "x" was often seen to increase the number of results.

When examining the first link returned from queries, we did not observe a positive relationship between filtering for a post with an accepted answer and a more relevant answer. Therefore, we suggest not discounting posts just because they contain no accepted answers.

LLMs demonstrated great potential to help developers fix their compiler errors. When comparing results across types of source code, LLMs appear to work particularly well for shorter code snippets with common errors, given the perfect results found for CUSTOM code snippets, and also when the entire code snippet is made available as part of the query. Thus, for developers who create simple code snippets that need a quick fix, an LLM may be an excellent tool to try first, ensuring that the code is included when available. However, unlike SO, LLM answers do not have other users voting or providing comments on accuracy. This, combined with the fact that LLMs can make misleading answers sound plausible, means that users must exercise discretion and consider whether the answers provided are truly correct.

For tool developers working with compiler errors, many adopt the approach of imitating how programmers search the Web for solutions and then automating the process. For these types of approaches, the same recommendations apply. For instance, if a tool functions as an IDE plugin, it might be beneficial to first try including code snippets in the query and searching directly on SO. On the contrary, for tools that operate similarly to a search engine and do not have access to user code, implementing a search that works via Google may be more effective.

However, this might not necessarily hold true for all approaches, and hence our findings can be implemented depending on how a tool developer plans to interact with SO and user code. It is essential for a tool developer to test different query formats and methods before committing to one, due to the dissimilarity of the results, as shown in Figure 3 and 4.

As we did not test the ChatGPT API, we cannot comment on the differences compared to using it through the user interface. However, assuming similar performance, the ChatGPT API may be a viable option for developers, especially if access to the underlying code is available.

5.2. For Researchers

There are opportunities to further investigate the impact of different factors on query effectiveness for both SO and LLMs. Given that minor changes

can have a significant impact on results, this opens up possibilities for researchers to delve deeper into the topic, e.g., by testing various parts of code snippets and/or code pre-processing approaches. Unlike Monperrus and Maia's study [37], where both code pre-processing methods improved their overall debugging process, we found mixed results. Consequently, further research would be beneficial, particularly considering the high percentage of SO usage in debugging errors.

Additionally, our study has demonstrated LLMs' potential in assisting with fixing compiler errors. Further research in this area would be compelling and can be approached from various angles, such as focusing on other types of errors, identifying the best types of questions to ask, or even detecting the characteristics that distinguish relevant and correct answers from misleading and non-sensical ones.

6. Threats to Validity

We acknowledge that there exist limitations and threats that may impact the validity of our results.

- Construct Validity: A threat comes from the subjectivity of measuring the relevance of SO posts and LLM responses to the underlying compiler error. To mitigate this, we implemented a clearly defined set of classifications consistent with previous studies [33, 9, 32, 28]. To further build confidence in the ratings, we also had a subset of the results independently annotated by a different rater. The inter-rater agreement of 0.860 suggests "almost perfect" agreement.
- Only first links are considered: When analysing the scores and metrics used in the SO results and discussion (excluding RBOs), we evaluated the results based on the first links returned from each query. One could argue that the relevance of a query should consider more than just the first link, noting that experienced developers are likely to consider further links [20]. We recognise this as a limitation of our current approach and that our results are based on the assumption that the first link returned is an accurate representation of the relevancy of subsequent links.
- Only first error encountered are considered: For the Java code from which we collected the data, queries were created only based on

the first error encountered, even when their compilation leads to multiple errors. It has often been advised to fix the first error first before recompiling, and that many subsequent errors are caused by the first [10, 18]. Hence, fixing only the first error each compile likely mimics common behaviour when error fixing anyway, and we do not anticipate this threat to have a large impact on overall validity. However, aspects of fixing multiple errors at once, a behaviour shown a few times in the LLM responses we analysed, could be a potential area for future study.

- Limited Number of Results from SO-API Searches: The low number of results returned from SO-API searches can potentially impact the findings. However, developers often access Stack Overflow by querying search engines such as Google instead of directly visiting the website [25]. Considering that Google searches may better reflect actual search behavior, this limitation may be mitigated to some extent.
- Reproducibility: As Google's search algorithm is not public, the exact same query may return different results depending on location, time, and device type [1], among others. This is also true for LLM answers, which are "sensitive to tweaks to the input phrasing or attempting the same prompt multiple times" [39]. To minimise the potential bias introduced by prompting, simple prompts were used and alternatives were evaluated. Additionally, the consistency of the methodology and the environment was maintained during data collection.
- Generalisability: Our study focuses on Java compiler errors. This means that the same conclusions may not hold for other programming languages and types of errors. Additionally, even for Java compiler errors, we cannot claim that the findings will generalise beyond our dataset. Although we have tried to minimise this threat by including code from different sources, we acknowledge that the scale of the dataset itself is still small, covers relatively shorter code snippets, and may not cover all types of compiler errors. Lastly, although we observe trends, e.g., between the usefulness of crowd-curated compiler error help and automatically generated support, we cannot generalise to future versions of tools.

7. Conclusion

Compiler error messages often fail to provide enough targeted information to enable programmers to fix their code. Our study systematically compares SO and LLMs, such as GPT-3.5 and GPT-4, for their effectiveness in explaining compiler error messages. Our findings demonstrate that GPT-4 consistently outperforms both GPT-3.5 and SO, especially when provided with the offending code snippet along with the error message. Furthermore, we examine the influence of query configuration on SO's ability to yield relevant answers and the impact of prompt configuration on LLMs' capacity to explain compiler errors. The insights gained from this research have practical implications for programmers, who can leverage these findings to optimise their debugging process, and for researchers, who can build upon this work to explore the rapidly evolving landscape of AI-driven assistance in programming and debugging.

References

- [1] Why your google search results might differ from other people. URL https://support.google.com/websearch/answer/12412910?hl=en.
- [2] Usage of /search get. URL https://api.stackexchange.com/docs/search.
- [3] Tree-sitter Introduction tree-sitter.github.io. https://tree-sitter.github.io/tree-sitter/.
- [4] CS 111: Common Java Errors cs-people.bu.edu. https://cs-people.bu.edu/dgs/courses/cs111-old/assignments/errors.html, 2014.
- [5] Toufique Ahmed, Noah Rose Ledesma, and Premkumar Devanbu. Synfix: Automatically fixing syntax errors using compiler diagnostics. arXiv preprint arXiv:2104.14671, 2021.
- [6] Titus Barik, Justin Smith, Kevin Lubick, Elisabeth Holmes, Jing Feng, Emerson Murphy-Hill, and Chris Parnin. Do developers read compiler error messages? In 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE), pages 575–585, 2017. doi: 10.1109/ICSE. 2017.59.

- [7] Ohad Barzilay, Christoph Treude, and Alexey Zagalsky. Facilitating crowd sourced software engineering via stack overflow. Finding Source Code on the Web for Remix and Reuse, pages 289–308, 07 2014. doi: 10.1007/978-1-4614-6596-61_5.
- [8] Brett A. Becker. An effective approach to enhancing compiler error messages. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, SIGCSE '16, page 126–131, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450336857. doi: 10.1145/2839509.2844584. URL https://doi.org/10.1145/2839509.2844584.
- [9] Andrew Begel and Thomas Zimmermann. Analyze this! 145 questions for data scientists in software engineering. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, page 12–23, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450327565. doi: 10.1145/2568225.2568233. URL https://doi.org/10.1145/2568225.2568233.
- [10] Mordechai Moti Ben-Ari. Compile and runtime errors in java. January 2007. URL https://introcs.cs.princeton.edu/java/11cheatsheet/errors.pdf.
- [11] Som Biswas. Role of chatgpt in computer programming.: Chatgpt in computer programming. *Mesopotamian Journal of Computer Science*, 2023:8–16, 2023.
- [12] Amiangshu Bosu, Christopher S. Corley, Dustin Heaton, Debarshi Chatterji, Jeffrey C. Carver, and Nicholas A. Kraft. Building reputation in stackoverflow: An empirical investigation. In 2013 10th Working Conference on Mining Software Repositories (MSR), pages 89–92, 2013. doi: 10.1109/MSR.2013.6624013.
- [13] Neil Christopher Charles Brown, Michael Kölling, Davin McCall, and Ian Utting. Blackbox: A large scale repository of novice programmers' activity. SIGCSE '14, page 223–228, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450326056. doi: 10.1145/ 2538862.2538924. URL https://doi.org/10.1145/2538862.2538924.

- [14] Kaibo Cao, Chunyang Chen, Sebastian Baltes, Christoph Treude, and Xiang Chen. Automated query reformulation for efficient search based on query logs from stack overflow, 2021. URL https://arxiv.org/abs/2102.00826.
- [15] Preetha Chatterjee, Minji Kong, and Lori Pollock. Finding help with programming errors: An exploratory study of novice software engineers' focus in stack overflow posts, 10 2019.
- [16] Chunyang Chen and Zhenchang Xing. Towards correlating search on google and asking on stack overflow. In 2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC), volume 1, pages 83–92, 2016. doi: 10.1109/COMPSAC.2016.210.
- [17] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374, 2021.
- [18] Sally Fincher. *Studying programming*, page 68. Bloomsbury Publishing, 2006.
- [19] Rezvan Ghaderi. Improving the retrieval of related questions in stackoverflow. 2015.
- [20] Chase Greco, Tyler Haden, and Kostadin Damevski. Stackintheflow: Behavior-driven recommendation system for stack overflow posts. In 2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion), pages 5–8, 2018.
- [21] Rahul Gupta, Soham Pal, Aditya Kanade, and Shirish Shevade. Deepfix: Fixing common c language errors by deep learning. In *Thirty-First AAAI conference on artificial intelligence*, 2017.
- [22] Rahul Gupta, Aditya Kanade, and Shirish Shevade. Deep reinforcement learning for syntactic error repair in student programs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 930–937, 2019.
- [23] Mubin Ul Haque, Isuru Dharmadasa, Zarrin Tasnim Sworna, Roshan Namal Rajapakse, and Hussain Ahmad. "i think this is the most

- disruptive technology": Exploring sentiments of chatgpt early adopters using twitter data, 2022. URL https://arxiv.org/abs/2212.05856.
- [24] Raphael Hoffmann, James Fogarty, and Daniel Weld. Assieme: Finding and leveraging implicit references in a web search interface for programmers. pages 13–22, 01 2007. doi: 10.1145/1294211.1294216.
- [25] Andre Hora. Googling for software development: What developers search for and what they find. In 2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR), pages 317–328, 2021. doi: 10.1109/MSR52588.2021.00044.
- [26] Maria Hristova, Ananya Misra, Megan Rutter, and Rebecca Mercuri. Identifying and correcting java programming errors for introductory computer science students. volume 35, pages 153–156, 01 2003. doi: 10.1145/611892.611956.
- [27] Marcin P. Joachimiak, J. Harry Caufield, Nomi L. Harris, Hyeongsik Kim, and Christopher J. Mungall. Gene set summarization using large language models, 2023.
- [28] Barbara Kitchenham and Shari Pfleeger. *Personal Opinion Surveys*, pages 63–92. 01 2008. ISBN 978-1-84800-043-8. doi: 10.1007/978-1-84800-044-5_3.
- [29] Juho Leinonen, Arto Hellas, Sami Sarsa, Brent Reeves, Paul Denny, James Prather, and Brett A Becker. Using large language models to enhance programming error messages. arXiv preprint arXiv:2210.11630, 2022.
- [30] Annie Li, Madeline Endres, and Westley Weimer. Debugging with stack overflow: Web search behavior in novice and expert programmers. In 2022 IEEE/ACM 44th International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET), pages 69–81, 2022. doi: 10.1145/3510456.3514147.
- [31] Xuliang Liu and Hao Zhong. Mining stackoverflow for program repair. In 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER), pages 118–129, 2018. doi: 10.1109/SANER.2018.8330202.

- [32] David Lo, Nachiappan Nagappan, and Thomas Zimmermann. How practitioners perceive the relevance of software engineering research. In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, page 415–425, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450336758. doi: 10.1145/2786805.2786809. URL https://doi.org/10.1145/2786805.2786809.
- [33] Sonal Mahajan, Negarsadat Abolhassani, and Mukul Prasad. Recommending stack overflow posts for fixing runtime exceptions using failure scenario matching. pages 1052–1064, 11 2020. doi: 10.1145/3368089. 3409764.
- [34] Lena Mamykina, Bella Manoim, Manas Mittal, George Hripcsak, and Björn Hartmann. Design lessons from the fastest q&a site in the west. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '11, page 2857–2866, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450302289. doi: 10.1145/1978942.1979366. URL https://doi.org/10.1145/1978942.1979366.
- [35] Davin Mccall and Michael Kölling. Meaningful categorisation of novice programmer errors. volume 2015, 10 2014. doi: 10.1109/FIE.2014. 7044420.
- [36] Mary McHugh. Interrater reliability: The kappa statistic. *Biochemia medica : časopis Hrvatskoga društva medicinskih biokemičara / HDMB*, 22:276–82, 10 2012. doi: 10.11613/BM.2012.031.
- [37] Martin Monperrus and Anthony Maia. Debugging with the Crowd: a Debug Recommendation System based on Stackoverflow. Research Report hal-00987395, Université Lille 1 Sciences et Technologies, 2014. URL https://hal.archives-ouvertes.fr/hal-00987395.
- [38] P. G. Moulton and M. E. Muller. Ditran—a compiler emphasizing diagnostics. 10(1):45–52, jan 1967. ISSN 0001-0782. doi: 10.1145/363018. 363060. URL https://doi.org/10.1145/363018.363060.
- [39] OpenAI. Chatgpt: Optimizing language models for dialogue, Jan 2023. URL https://openai.com/blog/chatgpt/.
- [40] OpenAI. Gpt-4 technical report, 2023.

- [41] Cole S. Peterson, Jonathan A. Saddler, Natalie M. Halavick, and Bonita Sharif. A gaze-based exploratory study on the information seeking behavior of developers on stack overflow. In *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI EA '19, page 1–6, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450359719. doi: 10.1145/3290607.3312801. URL https://doi.org/10.1145/3290607.3312801.
- [42] Miguel Pimenta. Java Programming Exercises with Solutions Practice Online code-exercises.com. https://code-exercises.com/.
- [43] Md Masudur Rahman, Jed Barson, Sydney Paul, Joshua Kayani, Federico Andrés Lois, Sebastián Fernandez Quezada, Christopher Parnin, Kathryn T. Stolee, and Baishakhi Ray. Evaluating how developers use general-purpose web-search for code retrieval. In 2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR), pages 465–475, 2018.
- [44] Caitlin Sadowski, Kathryn T. Stolee, and Sebastian Elbaum. How developers search for code: A case study. In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, page 191–201, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450336758. doi: 10.1145/2786805.2786855. URL https://doi.org/10.1145/2786805.2786855.
- [45] Eddie Antonio Santos, Joshua Charles Campbell, Dhvani Patel, Abram Hindle, and José Nelson Amaral. Syntax and sensibility: Using language models to detect and correct syntax errors. In 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER), pages 311–322. IEEE, 2018.
- [46] Ben Shneiderman. Designing computer system messages. Commun. ACM, 25(9):610-611, sep 1982. ISSN 0001-0782. doi: 10.1145/358628. 358639. URL https://doi.org/10.1145/358628.358639.
- [47] Dominik Sobania, Martin Briesch, Carol Hanna, and Justyna Petke. An analysis of the automatic bug fixing performance of chatgpt. arXiv preprint arXiv:2301.08653, 2023.
- [48] Stackify. Java software errors: How to avoid 50 code issues in java, Mar 2023. URL https://stackify.com/top-java-software-errors/.

- [49] Weiwei Sun, Zhengliang Shi, Shen Gao, Pengjie Ren, Maarten de Rijke, and Zhaochun Ren. Contrastive learning reduces hallucination in conversations, 2022. URL https://arxiv.org/abs/2212.10400.
- [50] Emillie Thiselton and Christoph Treude. Enhancing Python compiler error messages via Stack Overflow. In *Proceedings of the International Symposium on Empirical Software Engineering and Measurement*, pages 1–12. IEEE, 2019.
- [51] V. Javier Traver. On compiler error messages: What they say and what they mean. 2010, jan 2010. ISSN 1687-5893. doi: 10.1155/2010/602570. URL https://doi.org/10.1155/2010/602570.
- [52] Mitchell Wand. Finding the source of type errors. POPL '86, New York, NY, USA, 1986. Association for Computing Machinery. ISBN 9781450373470. doi: 10.1145/512644.512648. URL https://doi.org/ 10.1145/512644.512648.
- [53] Christopher Watson, Fred Li, and Jamie Godwin. Bluefix: Using crowd-sourced feedback to support programming students in error diagnosis and repair. volume 7558, pages 228–239, 09 2012. ISBN 978-3-642-33641-6. doi: 10.1007/978-3-642-33642-3 25.
- [54] William Webber, Alistair Moffat, and Justin Zobel. A similarity measure for indefinite rankings. 28(4), nov 2010. ISSN 1046-8188. URL https://doi.org/10.1145/1852102.1852106.
- [55] Richard L. Wexelblat. Maxims for malfeasant designers, or how to design languages to make programming as difficult as possible. ICSE '76, page 331–336, Washington, DC, USA, 1976. IEEE Computer Society Press.
- [56] Xin Xia, Lingfeng Bao, David Lo, Pavneet Singh Kochhar, Ahmed E. Hassan, and Zhenchang Xing. What do developers search for on the web? *Empirical Software Engineering*, 22, 12 2017. doi: 10.1007/s10664-017-9514-4.