Explaining GitHub Actions Failures with Large Language Models: Challenges, Insights, and Limitations

Pablo Valenzuela-Toledo[1,2*], Chuyue Wu[1*], Sandro Hernández[1*], Alexander Boll[1]
Roman Machacek[1], Sebastiano Panichella[1], Timo Kehrer[1]
[1]*Software Engineering Group, University of Bern*, Bern, Switzerland
[2]*Universidad de La Frontera*, Temuco, Chile

Abstract—GitHub Actions (GA) has become the *de facto* tool that developers use to automate software workflows, seamlessly building, testing, and deploying code. Yet when GA fails, it disrupts development, causing delays and driving up costs. Diagnosing failures becomes especially challenging because error logs are often long, complex and unstructured. Given these difficulties, this study explores the potential of large language models (LLMs) to generate correct, clear, concise, and actionable contextual descriptions (or summaries) for GA failures, focusing on developers' perceptions of their feasibility and usefulness. Our results show that over 80% of developers rated LLM explanations positively in terms of correctness for simpler/small logs. Overall, our findings suggest that LLMs can feasibly assist developers in understanding common GA errors, thus, potentially reducing manual analysis. However, we also found that improved reasoning abilities are needed to support more complex CI/CD scenarios. For instance, less experienced developers tend to be more positive on the described context, while seasoned developers prefer concise summaries. Overall, our work offers key insights for researchers enhancing LLM reasoning, particularly in adapting explanations to user expertise.

Index Terms—CI/CD, GitHub Actions, Large Language Models, GitHub Action Run Failure Explanation

I. INTRODUCTION

GitHub Actions (GA) is an orchestration platform within GitHub that enables developers to automate tasks such as building, testing, and deploying code in integration and deployment (CI/CD) environments [1]. By minimizing manual intervention, GA streamlines development processes, boosts productivity, and is essential for maintaining efficient and stable workflows in modern software teams. Technically, GA workflows operate within specific environments, aka. *"run/runners"*, which can be hosted on GitHub or on self-hosted servers [2] [3].

Diagnosing issues in GA workflow execution involves examining workflow execution logs [4], which capture critical data—such as timestamps, success or failure indicators, error messages, and stack traces—that help identify unexpected behaviors and trace failures to their underlying causes. Additionally, environment configurations and system states provide context that supports accurate diagnosis. In today's competitive environment, where speed and reliability are essential, quickly addressing and resolving workflow run failures is essential to limit deployment delays, increased operational costs, and compromised software stability [5]–[8].

However, diagnosing GA run failures presents significant challenges due to the volume, lack of structure, and complexity of the logs [2], [9]. Critical error information is often buried within extensive entries, complicating the diagnostic process, while the absence of a standardized log structure and inconsistent formats hinder automation of log analysis [2]. The complexity is further exacerbated by cryptic error codes, system-specific terminology, and interwoven events that obscure failure sequences, requiring specialized knowledge for accurate interpretation [10]. Analyzing these detailed logs to uncover root causes necessitates a comprehensive understanding of both the logs and the system. This situation complicates pinpointing the source of failures and the actual underlying problems [11], [12]. Consequently, efficient filtering mechanisms are needed to isolate relevant data and facilitate the diagnosis of failures, reducing labor-intensive and error-prone tasks.

Diagnosing GA run failures demands specialized knowledge that goes beyond standard troubleshooting methods such as searching on Google or Stack Overflow. For example, developers in the *bids-standard/bids-validator* repository encountered the error message, *"Error: failed to load the Docker image 'bids/validator': No such file or directory"*. Resolving this error required them to analyze extensive metadata and understand both Docker configurations and the architecture of the *bids-validator* tool.[1] Similarly, developers working in

---

the *shirasagi/shirasagi* repository processed over 20,000 lines of logs, which recorded every variable, status, and error.[2] These examples demonstrate how verbose GA logs often demand domain-specific expertise to interpret their context, forcing developers to tackle challenges that standard troubleshooting tools cannot address.

Recent advancements in software engineering show that large language models (LLMs) perform tasks such as code generation, summarization, understanding, and review effectively [13]–[16]. We propose that developers can use LLMs to generate explanations (or summaries) that help diagnose run failures in GA workflows. LLMs recognize patterns and relationships within unstructured data, making them a promising tool for providing actionable insights to address run failures. These insights enable developers to understand and fix issues more efficiently. To our knowledge, no previous study has explored how LLMs can support the understanding of GA workflow failures.

We conducted a mixed-methods feasibility study to evaluate how LLMs explain run failures and to understand developers' perceptions of their effectiveness. We invited 811 developers to participate, and 31 accepted the invitation. This study examines four key aspects of developer perception—*correctness*, *conciseness*, *clarity*, and *actionability*—based on theoretical constructs from the literature [17], [18]. We chose these metrics because researchers have widely used them in studies on summary generation and human-based assessment, making them both relevant and well-validated.

Our feasibility study shows that LLMs can diagnose GA workflow failures effectively, especially in straightforward cases. Over 80% of developers rated LLM-generated explanations as correct and clear for smaller or simpler error logs. These explanations captured essential details and provided developers with insights that helped them diagnose issues in less complex scenarios. However, LLMs struggled with intricate failures and failed to deliver the depth of reasoning required for complex CI/CD cases. Junior developers praised the contextual descriptions from LLMs, while experienced developers preferred concise explanations.

The implications of our findings are relevant primarily to developers and researchers. For developers, integrating LLMs into GA run failure diagnoses could improve troubleshooting efficiency for simpler errors by reducing the need for manual log analysis. This improvement could result in faster resolution times and greater productivity for development teams. For researchers, these findings point to new avenues for exploration, particularly in enhancing the reasoning capabilities of LLMs to address complex diagnostic tasks more effectively.

The contributions of our paper are as follows:

- We conducted a **feasibility study** demonstrating LLMs' potential for interpreting GA run failures.
- We evaluated **prompt engineering techniques** and identified one-shot prompt tuning as the most effective approach for generating consistent and accurate GA run failure explanations.
- We provided **developer insights**, offering initial feedback on developers' satisfaction with LLM-generated explanations and identifying the most valued attributes of these explanations.
- We established a **foundation for future research**, setting the groundwork for studies on the application of LLMs in software debugging and fault localization, and encouraging further exploration and innovation in this area.
- We make available a **replication package** with (i) materials and datasets from our study, (ii) complete survey results, (iii) appendix with complete analysis, and (iv) raw data to facilitate replication and support future research [19]. DOI 10.5281/zenodo.14750197

II. STUDY DESIGN

The goal of this study is to evaluate the feasibility of using LLMs to explain failures in GA runs. Specifically, this evaluation examines how attributes such as *correctness*, *conciseness*, *clarity*, and *actionability* in LLM-generated explanations contribute to their diagnostic usefulness. Table I provides definitions for each attribute, adapted from established constructs in prior work [17], [18]. We outline three research questions that guide our study, as follows:

RQ1: *To what extent do LLMs correctly describe the context of GitHub Action run failures according to developers?* We investigate the developers' perceptions of the *correctness* of LLM-generated explanations in conveying the context of GA run failures. Here, *correctness* indicates that the explanation is technically sound and aligns with the system and its failure's behavior.

RQ2: *To what extent do developers find generated explanations of LLMs for GitHub Action run failures clear and concise?* We examine the clarity and conciseness of LLM-generated explanations from the developers' perspective. *Clarity* refers to how understandable the explanations are, allowing developers to identify the issue quickly, while *conciseness* assesses whether the explanations contain only essential information, avoiding superfluous details. These two qualities play a significant role in determining the accessibility of the explanations, as they influence how effectively developers can interpret and utilize them for diagnosing the failures and troubleshooting activities.

RQ3: *To what extent are the descriptions of GitHub Action run failures generated by LLMs considered ac-*

Attribute	Definition
Correctness	Measures the accuracy and reliability of the LLM-generated explanations in describing the actual behavior of the system, ensuring information is free from misleading content and inspires confidence in the diagnosis provided.
Conciseness	Reflects whether the explanation is efficient and avoids unnecessary information, presenting only essential details to understand and resolve the issue effectively.
Clarity	Assesses whether the explanation is presented in a clear and understandable manner, enabling developers to readily grasp the issue and the suggested steps.
Actionability	Assesses whether the explanation provides clear, step-by-step guidance that is directly implementable, enabling developers to efficiently address and resolve the failure without needing further clarification or external resources.

Survey Statements & Questions

(1) The explanation accurately reflects the details and context of the GitHub Actions run failure.

(2) The run failure explanation is helpful.

(3) There is a low likelihood of a misleading explanation.

(4) The explanation accurately diagnoses the run failure.

(5) The explanation contains no inappropriate or incorrect content.

(6) There is evident sound diagnostic reasoning.

(7) The explanation clearly and understandably communicates the run failure.

(8) The explanation clearly outlines the subsequent steps to take.

(9) The explanation specifically addresses my needs without being too general.

(10) I am confident in the diagnosis provided by the run failure explanation.

⋆**(11)** What attributes make an error explanation valuable and effective for addressing issues related to GitHub Actions runs?

⋆**(12)** Do you have any additional comments or suggestions on how we can enhance our run failure explanations?

Fig. 1. Survey definition: Statements 1 through 10 are closed-ended, while questions 11 and 12 are open-ended (indicated with a star ⋆).

tionable by developers? We examine the *actionability* of LLM-generated explanations from the developers' perspective, assessing whether these explanations provide specific and relevant information which developers can implement into a resolution for their run failures.

A. Data Collection

We surveyed developers to investigate their perceptions on the feasibility and effectiveness of using LLMs to interpret and explain GA run failures.

Survey Design.

The survey consisted of 10 closed-ended statements and 2 open-ended questions as ordered in Fig. 1. Statements **(1)**, **(3)**, **(4)**, **(5)**, **(6)** assessed *correctness* (RQ1). Statements **(2)**, **(7)**, **(8)**, **(9)**, **(10)** focused on *conciseness* and *clarity* (RQ2). Each of these closed-ended statement used a Likert scale from 1 (strongly disagree) to 5 (strongly agree), selected based on validated assessment criteria from relevant studies in the field [20]–[26]. Finally, open-ended (OE) questions 11 and 12 addressed *actionability* (RQ3).

We conducted a pilot study with experienced and novice developers to validate the survey's items, layout, and duration. Their feedback clarified the survey, confirmed a 45-60 minute completion time, and ensured clear platform guidance. The study balanced the survey's length with its completion time by combining structured, closed-ended questions for consistency and open-ended ones for qualitative insights.

Survey Management & Setting. We developed and deployed LogExp, a custom web tool, to effectively administer the survey and evaluate LLM-generated explanations (see Fig. 2). This tool allowed for side-by-side display of both the logs (left) and their corresponding, statically deployed explanations (right), enabling participants to easily compare and assess them within a unified interface. In sum, the LogExp tool was populated with the logs and explanations of ten GA failure cases, the selection of which will be described in the sequel.

Log Selection Procedure. Our study focused on GA workflow run failures in JavaScript software systems, as JavaScript ranked as the most popular programming language on GitHub in 2022 [27]. Using Dabic *et al.*'s tool [28], we identified non-forked JavaScript repositories with at least one commit between March and May 2024 to ensure compliance with GitHub's 90-day log access limitation [29]. We selected repositories with workflows and prioritized active, large-scale projects. To refine the dataset, we applied inclusion criteria of at least 20 contributors, 100 stars, and 20 recent commits, and we removed extreme outliers. We downloaded all failure logs and filtered out logs with fewer than 45 words. A pilot study using the knee method [30] empirically determined this threshold and ensured the logs provided enough context for actionable explanations. Logs exceeding this threshold proved more meaningful, leaving us with 348 logs. From these, we selected ten examples
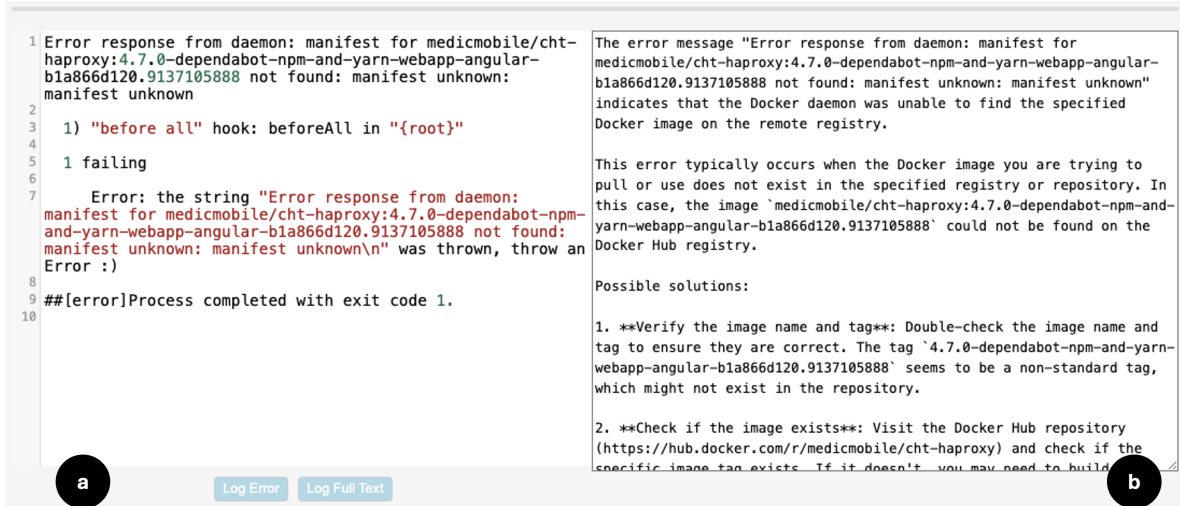
```
1  Error response from daemon: manifest for medicmobile/cht-
   haproxy:4.7.0-dependabot-npm-and-yarn-webapp-angular-
   b1a866d120.9137105888 not found: manifest unknown:
   manifest unknown
2
3    1) "before all" hook: beforeAll in "{root}"
4
5    1 failing
6
7       Error: the string "Error response from daemon:
   manifest for medicmobile/cht-haproxy:4.7.0-dependabot-npm-
   and-yarn-webapp-angular-b1a866d120.9137105888 not found:
   manifest unknown: manifest unknown\n" was thrown, throw an
   Error :)
8
9  ##[error]Process completed with exit code 1.
10
```
```
The error message "Error response from daemon: manifest for
medicmobile/cht-haproxy:4.7.0-dependabot-npm-and-yarn-webapp-angular-
b1a866d120.9137105888 not found: manifest unknown: manifest unknown"
indicates that the Docker daemon was unable to find the specified
Docker image on the remote registry.

This error typically occurs when the Docker image you are trying to
pull or use does not exist in the specified registry or repository. In
this case, the image `medicmobile/cht-haproxy:4.7.0-dependabot-npm-and-
yarn-webapp-angular-b1a866d120.9137105888` could not be found on the
Docker Hub registry.

Possible solutions:

1. **Verify the image name and tag**: Double-check the image name and
tag to ensure they are correct. The tag `4.7.0-dependabot-npm-and-yarn-
webapp-angular-b1a866d120.9137105888` seems to be a non-standard tag,
which might not exist in the repository.

2. **Check if the image exists**: Visit the Docker Hub repository
(https://hub.docker.com/r/medicmobile/cht-haproxy) and check if the
specific image tag exists. If it doesn't, you may need to build
```

Fig. 2. Partial view of the `LogExp` tool's interface. The log is displayed on the left, allowing participants to choose between viewing a summary or the full log. On the right, the corresponding textual explanation generated by the LLM is presented. Below these sections, participants encountered statements and questions specific to each case.

for our survey, which represented diverse failure cases from GA workflows, including continuous integration, deployment, and testing.

LLMs configuration. In our pilot study, we tested combinations of LLMs and prompt techniques to integrate them into `LogExp`. We compared three models: `Llama3` (70B), `Llama2` (70B), and `Mixtral` (8x7B). We chose `Llama3` and `Llama2` because they generated contextually accurate and structured explanations that processed CI/CD logs effectively [31]. We included `Mixtral` for its strong performance in few-shot prompting, which reduced the need for fine-tuning and handled varied error contexts [32]. We used a predefined template from the replication package [32] to generate explanations. We selected LLMs for their customizability and availability, ensuring they met our research needs. We excluded proprietary models such as GPT because their limited fine-tuning capabilities made them unsuitable for our exploratory research. Finally, we prioritized models that delivered high-quality explanations while reducing processing costs and complexity to ensure accessibility for widespread use [33], [34].

In the pilot study, we tested three prompting techniques—*zero-shot*, *one-shot*, and *few-shot*—on 348 GitHub Actions failure logs to evaluate their effectiveness. Prior research shows that these techniques help language models adapt to tasks with few or no examples [32], [35]. Five developers rated the explanations on *correctness*, *conciseness*, *clarity*, and *actionability*, and their feedback improved our prompts and models. Their evaluations identified limitations and refined our prompting techniques for each model. The pilot study showed that `Llama3` produced the most relevant and contex-

tually accurate explanations, with *one-shot* prompting providing the best balance of simplicity and accuracy.

Participants Sampling Strategy. We used purposive sampling to select contributors familiar with GA [36], [37]. This method focused on developers most likely to provide relevant insights. We targeted developers who actively created, maintained, and troubleshot GA run failures. We identified these developers through their recent contributions to selected JavaScript projects on GitHub, which served as sources for our GA failure cases. To recruit participants, we contacted developers who had recently contributed to these projects. Our recruitment strategy prioritized contributors from projects with high activity to ensure their engagement in workflow development and maintenance and selected developers with experience handling and troubleshooting GA run failures.

Demographics. We distributed the survey to 811 developers and received 31 responses, achieving a response rate of 3.82%. Although lower than the typical range for software engineering surveys (6% to 36%) [38], this response rate aligns with exploratory studies, where 30 to 50 responses often provide sufficient insights. Most participants completed the survey in the expected time, with a median of 45 minutes. We analyzed only surveys with at least 70% completion and retained missing data points as blanks to reduce bias.

Most respondents were male (90%), and 10% female. Over half (52%) had more than 11 years of experience in software development, while 16% had 6-10 years, 13% had around 2 years, 10% had 3-5 years, 6% had 1 year or less, and 3% reported no professional experience. In terms of education, 39% of participants held a Mas-

ter's degree in Computer Science, 6% had a Master's in Science, Technology, Engineering, and Mathematics (STEM), and 6% had a Master's in non-STEM fields. Bachelor's degrees were also common, with 23% in Computer Science or Software Engineering and 16% in other STEM fields. A smaller portion of respondents held a high school diploma (3%), some college or a 2-year degree (3%), or a PhD or other advanced degrees (3%).

Ethics Considerations. This study complies with the research ethics principles/regulations imposed by our university: Informed consent was obtained from all participants, and safeguards were implemented to minimize the risk of re-identification at a later time.

B. Data Analysis

To analyze the data, we employed a combination of quantitative and qualitative techniques, integrating responses from closed-ended statements with insights from open-ended questions allowing for detailed, free-form responses. Our mixed-methods approach aligns with practices in prior studies (e.g., [20]–[26]) and was chosen to capture both the measurable trends and the nuanced perspectives of developers regarding LLM-generated explanations.

Quantitative Analysis. Following best practices for survey data analysis as outlined by Kitchenham and Pfleeger [39], and Ralph *et al.* [40], we performed a quantitative analysis for both RQ1 and RQ2, computing descriptive statistics (mean, median, agreement). This methodological approach quantifies developers' perceptions of different aspects of LLM-generated explanations, offering a structured view of how these explanations are received.

Qualitative Analysis. To address RQ3, we analyzed open-ended responses from survey questions 11 and 12 using card-sorting based on Zimmermann's three-phase methodology [41]. In the preparation phase, we entered responses into an Excel sheet and split each one into cards, ensuring each card captured a single, relevant idea. During the execution phase, three authors grouped the cards independently by thematic similarity and resolved ambiguities through discussions to ensure consistency. In the analysis phase, we combined related groups into broader categories and identified key attributes that make LLM-generated explanations actionable for resolving GA run failures. We measured inter-rater agreement with Cohen's Kappa (0.74), which showed moderate to good agreement.

III. Correctness (RQ1)

To address RQ1, we examined developer responses to statements **(1)**, **(3)**, **(4)**, **(5)**, and **(6)**, each of which evaluated different aspects of the *correctness* of LLM-generated explanations for GA run failures.
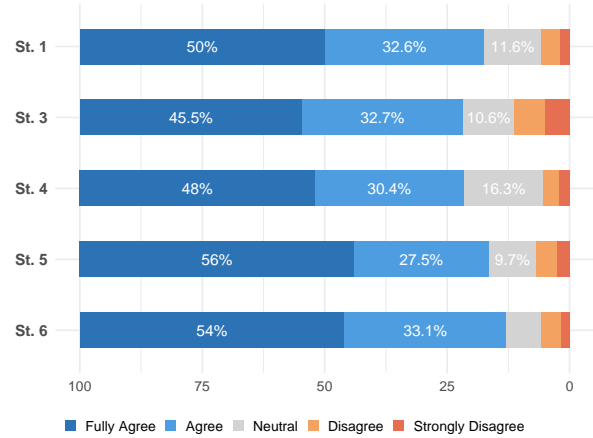

Fig. 3. The stacked bar chart shows the levels of agreement of participants to our statements **(1)**, **(3)**, **(4)**, **(5)**, and **(6)**.

Overall, developers responded positively to these statements on correctness (see Fig. 3). St. **(1)**, assessing whether explanations accurately captured the details and context of GA run failures, received 50% full agreement and 32.6% partial agreement, totaling 82.6% agreement, which suggests that most developers found the explanations accurate. For St. **(3)**, which focused on the likelihood of explanations being free from misleading content, 45.5% fully agreed and 32.7% partially agreed, resulting in an overall agreement of 78.2%, indicating that participants generally trusted the reliability of the explanations. St. **(4)**, which evaluated the precision of the explanations in diagnosing technical issues, shows 48% full agreement and 30.4% partial agreement, with a total of 78.4% agreement. St. **(5)**, which addressed the absence of incorrect content, showed 56% full agreement and 27.5% partial agreement, leading to 83.5% agreement overall.

Lastly, St. **(6)**, which assessed the logical coherence of the explanations, received 54% full agreement and 33.1% partial agreement, totaling 87.1% agreement, indicating strong approval of the explanations' logical soundness.

> **Answer to RQ1**
>
> In summary, developers rated the *correctness* of LLM-generated explanations positively, with over 80% agreement across statements on accuracy, diagnostic precision, and logical coherence.

IV. Conciseness and Clarity (RQ2)

To address RQ2, we analyzed developer responses to statements **(2)**, **(7)**, **(8)**, **(9)**, and **(10)**, each targeting distinct aspects of the conciseness and clarity of LLM-generated explanations in the context of GA run failures.
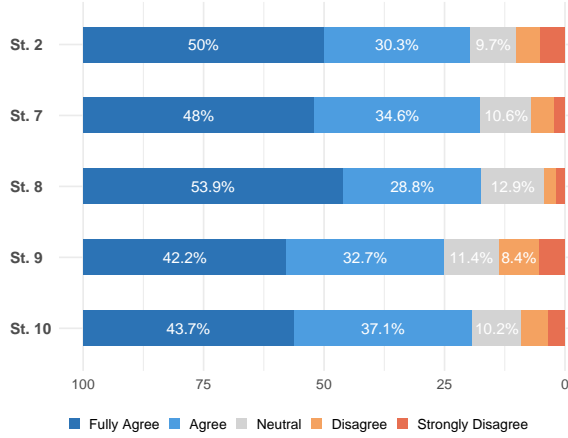
Fig. 4. The stacked bar chart shows the levels of agreement of participants to our statements (**2**), (**7**), (**8**), (**9**), and (**10**).

Overall, developers responded positively to statements evaluating the conciseness and clarity of LLM-generated explanations (See Fig. 4). St. (**2**), which assessed the helpfulness of the explanations, showed 50% full agreement and 30.3% partial agreement, with 80.3% agreement overall, indicating that developers found the explanations useful for diagnosing failures. St. (**7**), directly evaluating clarity, received 48% full agreement and 34.6% partial agreement, reaching 82.6% agreement overall, emphasis that the explanations were communicated clearly and understandably. St. (**8**), focusing on outlining actionable steps, had 53.9% full agreement and 28.8% partial agreement, with 82.7% agreement overall. St. (**9**), assessing the relevance and specificity of the explanations, obtained 42.2% full agreement and 32.7% partial agreement, resulting in 74.9% agreement overall, reflecting a moderate level of satisfaction with the explanations' ability to address developers' specific needs. Finally, St. (**10**), which measured developers' confidence in the provided diagnosis, achieved 43.7% full agreement and 37.1% partial agreement, with 80.8% agreement overall.

> **Answer to RQ2**
>
> Developers rated the LLM-generated explanations positively for both *conciseness* and *clarity*. For *clarity*, over 80% of participants found the explanations easy to understand,. In terms of *conciseness*, 74.5% agreed that the explanations were specific and not overly broad.

V. ACTIONABILITY (RQ3)

Participants evaluated the actionability of the explanations based on LLM attributes. Their responses revealed five key attributes that shape effective explanations.

Each attribute reflects a distinct aspect of actionability that developers considered valuable for diagnosing and resolving GA run failures. Below, we describe these attributes and explain how they collectively enhance the actionability of LLM-generated explanations.

A. Clarity of the Explanation (16%)

This category addresses the clarity of the understanding of the explanation, avoiding excessive technical jargon and providing clear information accessible to developers at various expertise levels. Here, 16% of the answers highlighted this quality, indicating its importance in making explanations universally understandable. Two key attributes stand out: *Clarity in error explanation* and *Clarity in the steps to follow*.

Clarity in error explanation refers to the transparency and comprehensibility of the explanation: The error is described in a straightforward manner, avoiding ambiguity. As one participant noted:

> ❝ *I believe that a useful error explanation should get straight to the point without saying a lot of unnecessary things. Furthermore the explanation should be easily understandable by people that are just getting started so they can become better at understanding errors. Last but not least the steps to fix the issue shouldn't be too general because then a google search is better. [ID:5]*"

Clarity in steps to follow refers to providing clearly defined steps to address the failure: Each step is presented in a logical order, without unnecessary details.

> ❝ *Clear resolution steps with examples fitting to the actual code that has one or more errors - Clear problem explanation - References to trustable sources. [ID:21]*"

B. Actionable Guidance (18%).

This category refers to how effectively the explanation provides developers with specific, implementable steps that are directly related to resolving an specific error. 18% of the answers highlighted the *Precision in instructions* and *Direct applicability of suggested steps*.

Precision in instructions includes instructions that are specific, precise, and directly relevant to the problem: The solution steps leave no room for doubt or interpretation, allowing the developer to apply the instructions directly. As one participant noted:

> ❝ *If the tool has access to the source code, (eg. the Actions config, project code, etc), the suggestions could actually include diffs designed to fix the issues. [ID:33]*"

Direct applicability of suggested steps describes the immediate use of the provided steps to resolve the issue:

Developers can follow the instructions without needing additional research or context.

> ❝ *1. Cutting fluff, going straight to the point. 2. Possible steps to take to fix the problem. [ID:2]*"

C. Specificity of Content (18%)

This category describes how well the explanation is tailored to the specific technical context in which the failure occurred. 18% of the answers highlighted this quality, indicating its importance in providing relevant and accurate information for troubleshooting. The attributes in this category are: *Adaptation of content to the context of the failure* and *Technical accuracy in describing the problem*.

Adaptation of content to the context of the failure refers to the altering of the explanation to its specific environment or configuration:

> ❝ *Detecting the right context is key for error detection. If the right context is used, I also such as the examples given as long as they are small snippets. If there is more background necessary, providing a link would be my preferred way. On CI/CD topics the model is potentially useful, as you could easily try to use the recommendation and check if it resolves the issue, but this will only be helpful if the root cause of the issue is correctly identified. [ID:4]*"

Technical accuracy in describing the problem involves detailed and accurate descriptions of the error, with all technical aspects correctly presented:

> ❝ *Narrowing down where the error happened and giving a brief explanation of what the command/process does. It helps also developers that are only 'consuming' tests to figure out faster if the failure is their fault or maybe ci/cd needs adjustments. [ID:16]*"

D. Contextual Relevance (27%)

This category addresses the inclusion of additional context or external resources to help developers understand the problem more fully. 27% of the answers emphasized the importance of contextual relevance in error explanations. The specific attributes in this category are *Inclusion of relevant links* and *Explicit mention of dependencies or technical conflicts*.

Inclusion of relevant links involves providing links to additional information about the error or its resolution:

> ❝ *Concise information with links to relevant GitHub Actions documentation, issue trackers, or other resources that provide more in-depth information. Also, include the line numbers of the file where the issue occurs. [ID:6]*"

Explicit mention of dependencies or technical conflicts involves identifying any dependencies or conflicts that may contribute to the failure:

> ❝ *Providing definitions of things that developers take for granted. Referencing components explicitly. Eg saying there is a dependency conflict between lib X and Lib Y. This is better than saying "there is a conflict between 2 files" You guys have nailed it. Very nice work. [ID:7]*"

E. Conciseness (18%)

This category refers to the provision of brief yet informative explanations that enable developers to understand the problem efficiently, focusing on essential information omitting unnecessary details. 18% of the answers highlighted the importance of conciseness in explanations. The specific attributes in this category are the *Ability to quickly summarize the cause of the failure* and the *Concise presentation of resolution steps*.

The *ability to quickly summarize the cause of the failure* involves identifying the root cause of the issue:

> ❝ *There are too many details in the procedure, the sentences such as "Edit the file" and "Commit your changes" are superfluous for a developer. [ID:34]*"

Concise presentation of resolution steps includes only the necessary actions to resolve the issue, presented clearly and directly. As another participant shared:

> ❝ *The LLM generated texts are a little bit of being too long, it could be briefer without losing content. [ID:27]*"

Answer to RQ3

Effective explanations for GitHub Actions run failures include five key attributes: *clarity*, which provides straightforward information; *actionable guidance*, offering precise steps for resolution; *specificity*, adapting explanations to the technical context; *contextual relevance*, adding links or details about dependencies; and *conciseness*, ensuring only essential information is presented.

VI. DISCUSSION AND IMPLICATIONS

This section summarizes key findings for each research question, discussing relevant confounding factors and implications for future research and practical applications. In addition to the results we systematically presented in the last section, we further analyzed various metrics that we present in this section.

RQ1: Correctness of LLM-Generated Explanations. Our results indicate that LLMs provide accurate

explanations for simpler failures, especially with structured and concice logs. This aligns with previous studies that stress the role of well-organized logs in facilitating error diagnosis in CI/CD environments. For instance, Vassallo *et al.* highlight that structured data in CI logs improves diagnostic efficiency, with LLMs benefiting from reduced ambiguity and focused log information [5].

LLMs struggle with verbose and unstructured logs that obscure key information. Sallou *et al.* highlight LLMs' inconsistent performance on unstructured data, impacting their accuracy in real-world scenarios [42]. In our work, excessive detail in log entries hid critical error messages, revealing the need for preprocessing to extract relevant information. For instance, one participant noted, *"... providing definitions of things that developers take for granted. - Referencing components explicitly. Eg saying there is a dependency conflict between lib X and Lib Y"*. would greatly enhance correctness. This suggests that developers cannot use LLMs and prompts directly; they must tailor preprocessing for specific contexts. Additionally, preprocessing is crucial for large logs due to LLMs' token limits [43].

Our analysis showed that 80% of participants used CI/CD in their primary activities. This widespread use shaped their expectations for error explanations, as CI/CD users gave higher median scores of 4.5 compared to 3.2 from participants who did not rely on CI/CD as a primary activity. CI/CD users also rated explanations more consistently, with a standard deviation of 0.5, while non-users had a standard deviation of 1.2. These findings suggest that familiarity with CI/CD practices unifies perceptions of error explanations. Prior research [5], [44] supports this observation, showing that structured input improves diagnostic efficiency in CI/CD contexts.

Our analysis of log length revealed that shorter logs received higher and more consistent ratings, while longer logs produced greater variability in responses. This result suggests that log length influences how participants perceive the explanations. These findings align with prior research showing the value of adaptive summaries tailored to user experience levels [17]. Customizing explanations based on developers' expertise and accounting for log-specific contextual factors could improve correctness.

RQ2: Conciseness and Clarity of LLM-Generated Explanations.

The study finds that LLMs generally provide clear and concise explanations for simple GA run failures, particularly when logs are structured and free from excessive complexity or irrelevant information. This observation supports the notion that conciseness is key to effective communication in technical contexts. Previous research by Vassallo *et al.* [5] emphasizes that well-organized logs facilitate error diagnosis in CI/CD environments, thereby enhancing clarity. Furthermore, Xia *et al.* [44] suggest that LLMs leverage structured input to produce outputs that are not only reliable but also easy to understand.

We observed no significant differences in conciseness or clarity based on log complexity (*i.e.*, cryptic error codes, system-specific terminology, and interwoven events that obscure the failure sequence). However, participants occasionally noted redundancy in the explanations. This aligns with the validity threats discussed by Sallou *et al.*, who warn that LLMs may introduce unnecessary details that obscure the main issue [42]. Similarly, Chen *et al.* highlight that verbosity in LLM-generated explanations can reduce their effectiveness when excessive information overshadows important details [45]. One participant remarked, *"narrowing down where the error happened and giving a brief explanation of what the command/process does helps developers figure out faster if the failure is their fault or maybe CI/CD needs adjustments." [ID:16]*.

RQ3: Actionability of LLM-Generated Explanations. The study identifies five key dimensions—clarity, actionable guidance, specificity, contextual relevance, and conciseness—as central to the actionability of LLM-generated explanations for GA failures, independent of error complexity or developer experience. This aligns with findings by Xia *et al.*, who emphasize that clarity and relevance are foundational for effective automated diagnostics in software engineering [44]. Additionally, Chen *et al.* note that without focused guidance, LLM explanations risk becoming too generalized, which can reduce their practical impact for both novice and experienced users [45]. Our results suggest that focusing on these dimensions may benefit all experience levels.

Interestingly, the study found minimal variation in actionability perceptions across experience levels, suggesting that LLMs can support a wide range of users with consistent guidance. However, Sallou *et al.*. discuss the limitations of static LLM outputs, especially in failing to provide the in-depth insights that advanced users might seek [42]. This suggests a potential improvement area: developing adaptive models that could provide variable explanation depth based on user experience, thereby addressing both basic and advanced informational needs.

Future research could explore *dynamic adaptation* in LLM explanations, as proposed by Ye *et al.*, who argue that personalized guidance based on familiarity enhances the value of automated explanations [46]. Such adaptability could ensure that LLMs provide universally useful guidance and cater to the specific cognitive demands of developers in complex CI/CD environments.

Furthermore, quality of generated output can be improved using numerous ways. Firstly, *consensus based generation* [47] would allow different models to unite their response and utilize different views. Secondly, models have shown improved reasoning and generation with

chain-of-thought prompting [48], for further explanation and improvement of results. Lastly, instruction prompting [49] can be utilized for more fine-grained control over generated text, for instance output generation based on the experience of the developer.

Implications for Researchers. Our results suggest several research directions for improving LLMs' ability to support diagnostics in CI/CD environments. For instance, researchers could focus on refining LLMs to better handle diverse log structures by developing preprocessing techniques to filter essential information within verbose or unstructured logs. An example of this can be seen in the GA log from the BIDS Validator project, which shows how clarity can be enhanced by emphasizing key details.[3] Such preprocessing could enable LLMs to generate accurate explanations even in complex log contexts. Additionally, adapting LLM outputs to user-specific contexts based on expertise level could enhance usability, providing concise, high-level insights for advanced users and step-by-step guidance for beginners. Investigating these adaptive mechanisms and fine-tuning LLM responses for both novice and expert users offers a promising approach for making LLM-powered diagnostics more universally applicable.

Implications for Developers. For developers, the findings suggest practical improvements for LLM-powered diagnostic tools that could increase explanation accuracy and relevance. Specifically, integrating log preprocessing features to structure data before analysis can help LLMs focus on the most critical information, resulting in clearer and more actionable explanations. Additionally, adjustable explanation levels could be defined by variables such as the developer's experience level, the complexity of the run failure, and the context of the log data. For instance, developers could toggle between concise summaries for routine errors and comprehensive guidance for more complex issues. Such flexibility would make LLM tools more adaptable to real-world CI/CD workflows, improving troubleshooting efficiency by delivering the appropriate level of detail tailored to each developer's needs.

Practical Applications. The findings from RQ1, RQ2, and RQ3 suggest that LLM-powered explanations are generally perceived as accurate, concise, clear, and actionable by both novice and expert users, with the potential to enhance CI/CD troubleshooting efficiency by reducing manual intervention. For instance, a GitHub Community discussion describes persistent run failure in GA due to frequent misconfigurations in permissions and authentication settings, resulting in a run failure[4]. Developers in the thread discuss solutions such as adjust-

---

[3]https://bit.ly/4fCDOGt
[4]https://bit.ly/3AoGKYr

ing permissions and optimizing job conditions to prevent these failures. This example shows how LLMs could provide automated diagnostics by identifying common misconfigurations and suggesting corrective actions for recurring issues. By offering context-sensitive insights, LLMs could help reduce the high rate of failures in CI/CD workflows, streamlining the troubleshooting process and improving workflow reliability.

VII. THREATS TO VALIDITY

In this section, we outline possible threats to the validity of our study and show how we mitigated them.

Construct Validity. Due to the fact that our study was performed in a remote setting in which participants could work on the tasks at their own discretion, we could not oversee their behavior. However, we anticipate the procedure of the experiments and all relevant details on how to conduct the experiments with a survey guiding the various steps of participants.

In addition, our study limited the assessment of the log summaries based on four main metrics (i.e., Correctness, Conciseness, Clarity, and Actionability), which may theoretically restrict the generalization of our findings to these specific metrics. However, we selected them because they are widely adopted in academia on previous summary generation and human-based assessment studies [17], [18].

Internal Validity. As LLMs generate their responses in a probabilistic manner, additional responses to the same prompts could have been perceived as better or worse than those from our survey. A setup where participants rate multiple explanations to the same logs and prompts could have controlled for this. However, our study-setup was already time-consuming with 45 minutes, and we wanted to ensure enough participants would complete our survey. Moreover, the study participants rated the log summaries based on their perceived quality of description generated by the approach. To limit the risks of biased assessments, we clarified the meaning of the criteria used to assess the log summaries before the experiments to the participants. Moreover, we specifically targeted developers who are actively involved in creating, maintaining, and troubleshooting GA run failures. Finally, the varying complexity of logs may lead to varying perceptions of the corresponding generated summaries. Rather than a threat this is more of a potential confounding factor we actually investigate in the context of our study.

Threats to External Validity. The selection of participants does not represent the general software developer population. Instead, we specifically targeted relevant developers (i.e., actively involved in creating, maintaining, and troubleshooting GA run failures), primarily by applying a sampling approach that selects a representative

set of contributors familiar with GA [36], [37]. To address other potential bias, we ensured diversity in terms of developers experience with GA, reducing the influence of factors beyond professional background. Furthermore, from our results one cannot predict the quality of explanations of other LLMs and different prompting techniques, which we did not test systematically. Our study does not aim to optimize LLM performance or determine the best model or prompting strategy. Instead, it explores how LLMs, as an emerging technology, can enhance developers' understanding of GitHub Actions run failures through actionable explanations. This exploratory work addresses a gap in CI/CD troubleshooting by identifying trends and providing foundational insights rather than exhaustive comparisons. . However, our internal pilot study showed that the three LLMs `Llama3`, `Llama2`, and `Mixtral` with various prompting techniques returned explanations of similar quality. Recent studies suggest that modern LLMs achieve hight quality in summaries and explanations in specific contexts [50]. Finally, further studies will be required to see whether results generalize to other logs of other programming languages, an effort which we leave for future work.

VIII. Related Work

This section reviews advancements in automatic summarization within software engineering.

(i) Build failure detection and resolution tools. Several studies address build failure detection in CI/CD environments. Vassallo *et al.* introduced BART to summarize and resolve Maven build failures and conducted a broad analysis identifying common CI failure causes across open-source projects, providing classification methods for typical errors [5], [51]. Rausch *et al.* also examined CI failures, focusing on error categories and the importance of build stability to support developer workflows [52]. Additionally, Alfaro *et al.* proposed Microprints, a visualization tool to identify errors in CI/CD logs, although it does not specifically address run failure explanations [11], [12].

(ii) Challenges and developments in source code summarization and metric evaluation. The lack of standardized datasets complicates source code summarization, as LeClair *et al.* highlight the challenges inconsistent data presents for research outcomes [53]. Moreno *et al.* advocate for improved benchmarks to maximize the utility of automatic summarization for stakeholders [54]. Additionally, Tarar and Zhang's work on bug report summarization demonstrates efficiency gains by leveraging syntactic and semantic similarities [55], [56]. Traditional metrics such as BLEU and ROUGE are critiqued by Stapleton *et al.* and Roy *et al.* for inadequacies in reflecting developer needs, while newer metrics such as

SIDE, which employ contrastive learning, provide better alignment with human evaluations [57]–[61].

(iii) The application of neural and federated learning models in summarization. Neural and federated learning models have significantly advanced summarization techniques. Iyer *et al.*'s CODE-NN, a neural attention model, surpassed previous methods for summarizing C# and SQL code [62]. Kumar *et al.* proposed FedLLM, a federated learning approach to code summarization that ensures data privacy while maintaining centralized model performance [63]. Structured summaries developed by Moreno *et al.* and by McBurney and McMillan aid developers in understanding Java code by using stereotypes and call graphs [64], [65]. Other works, such as those by Rastkar and Haiduc, have improved bug report handling and code entity summarization [66], [67]. Dabrowski's review of app review analysis emphasizes their growing importance in software development, and Nazar and Panichella show the utility of automated summarization in software maintenance and debugging [68]–[70].

Collectively, these studies indicate a need for diagnostic tools that not only identify errors but also provide actionable explanations for complex, unstructured log data. While previous work has advanced detection, categorization, and summarization, interpreting run failures in CI/CD workflows such as those in GA remains a challenge. Our research builds on these foundations, aiming to address this gap by leveraging large language models to enhance troubleshooting support.

IX. Conclusion

Our findings show that developers appreciate clear, concise LLM explanations for simpler issues. However, as error complexity grows, LLMs often lack accuracy and detail, underscoring the need for improvement in complex troubleshooting. Future work should focus on enhancing LLMs' capacity to contextualize complex errors in GA workflows and tailoring explanations for different expertise levels. Integrating log preprocessing and customizable explanation settings can potentially further improve usability, helping LLMs prioritize critical information and meet diverse troubleshooting needs. Advancing in these areas may lead to more effective diagnostic tools, boosting developer productivity and CI/CD stability.

References

[1] M. Golzadeh, A. Decan, and T. Mens, "On the rise and fall of CI services in GitHub," in *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering*. IEEE, 2022, pp. 662–672.

[2] S. G. Saroar and M. Nayebi, "Developers' perception of github actions: A survey analysis," in *Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering*, 2023, pp. 121–130.

[3] Y. Zhang, Y. Wu, T. Chen, T. Wang, H. Liu, and H. Wang, "How do developers talk about GitHub actions? evidence from online software development community," in *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*, 2024, pp. 1–13.

[4] H.-N. Zhu, K. Z. Guan, R. M. Furth, and C. Rubio-Gonzalez, "Actionsremaker: Reproducing github actions," in *2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings*. IEEE, 2023, pp. 11–15.

[5] C. Vassallo, G. Schermann, F. Zampetti, D. Romano, P. Leitner, A. Zaidman, M. Di Penta, and S. Panichella, "A tale of ci build failures: An open source and a financial organization perspective," in *2017 IEEE international conference on software maintenance and evolution*. IEEE, 2017, pp. 183–193.

[6] A. Miller, "A hundred days of continuous integration," in *Agile 2008 conference*. IEEE, 2008, pp. 289–293.

[7] B. Wilkes, A. M. P. Milani, and M.-A. Storey, "A framework for automating the measurement of devops research and assessment (dora) metrics," in *2023 IEEE International Conference on Software Maintenance and Evolution*. IEEE, 2023, pp. 62–72.

[8] Z. Zeng, T. Xiao, M. Lamothe, H. Hata, and S. McIntosh, "How trustworthy is your ci accelerator? a comparison of the trustworthiness of ci acceleration products," *IEEE Software*, 2024.

[9] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and M. R. Lyu, "Tools and benchmarks for automated log parsing," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice*. IEEE, 2019, pp. 121–130.

[10] Z. Chen, J. Liu, W. Gu, Y. Su, and M. R. Lyu, "Experience report: Deep learning-based system log analysis for anomaly detection," *arXiv preprint arXiv:2107.05908*, 2021.

[11] S. Alfaro, A. Bergel, and J. Simmonds, "Detecting ci/cd workflow errors through visual inspection of logs," *Authorea Preprints*, 2024.

[12] ——, "mu printgen: Supporting workflow logs analysis through visual microprint," in *2023 IEEE Working Conference on Software Visualization*. IEEE, 2023, pp. 45–49.

[13] A. Mastropaolo, M. Ciniselli, M. Di Penta, and G. Bavota, "Evaluating code summarization techniques: A new metric and an empirical characterization," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 2024, pp. 1–13.

[14] A. Mastropaolo, F. Zampetti, G. Bavota, and M. Di Penta, "Toward automatically completing GitHub workflows," in *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*, 2024, pp. 1–12.

[15] R. Tufano, O. Dabić, A. Mastropaolo, M. Ciniselli, and G. Bavota, "Code review automation: strengths and weaknesses of the state of the art," *IEEE Transactions on Software Engineering*, 2024.

[16] D. Nam, A. Macvean, V. Hellendoorn, B. Vasilescu, and B. Myers, "Using an llm to help with code understanding," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 2024, pp. 1–13.

[17] S. Panichella, A. Panichella, M. Beller, A. Zaidman, and H. C. Gall, "The impact of test case summaries on bug fixing performance: An empirical investigation," in *Proceedings of the 38th international conference on software engineering*, 2016, pp. 547–558.

[18] A. Di Sorbo, S. Panichella, C. V. Alexandru, J. Shimagaki, C. A. Visaggio, G. Canfora, and H. C. Gall, "What would users change in my app? summarizing app reviews for recommending software changes," in *Proceedings of the 2016 24th ACM SIGSOFT international symposium on foundations of software engineering*, 2016, pp. 499–510.

[19] P. Valenzuela-Toledo, C. Wu, S. Hernández, A. Boll, R. Machacek, S. Panichella, and T. Kehrer, "Explaining github actions failures with large language models: Challenges, insights, and limitations," Jan. 2025. [Online]. Available: https://doi.org/10.5281/zenodo.14750197

[20] S. Barke, M. B. James, and N. Polikarpova, "Grounded copilot: How programmers interact with code-generating models," *Proceedings of the ACM on Programming Languages*, vol. 7, no. OOPSLA1, pp. 85–111, 2023.

[21] R. Cheng, R. Wang, T. Zimmermann, and D. Ford, ""it would work for me too": How online communities shape software developers' trust in ai-powered code generation tools," *ACM Transactions on Interactive Intelligent Systems*, vol. 14, no. 2, pp. 1–39, 2024.

[22] A. M. Dakhel, V. Majdinasab, A. Nikanjam, F. Khomh, M. C. Desmarais, and Z. M. J. Jiang, "Github copilot ai pair programmer: Asset or liability?" *Journal of Systems and Software*, vol. 203, p. 111734, 2023.

[23] P. Denny, V. Kumar, and N. Giacaman, "Conversing with copilot: Exploring prompt engineering for solving cs1 problems using natural language," in *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, 2023, pp. 1136–1142.

[24] S. Imai, "Is github copilot a substitute for human pair-programming? an empirical study," in *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings*, 2022, pp. 319–321.

[25] D. Jayagopal, J. Lubin, and S. E. Chasins, "Exploring the learnability of program synthesizers by novice programmers," in *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology*, 2022, pp. 1–15.

[26] E. Jiang, E. Toh, A. Molina, K. Olson, C. Kayacik, A. Donsbach, C. J. Cai, and M. Terry, "Discovering the syntax and strategies of natural language programming with generative language models," in *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, 2022, pp. 1–19.

[27] GitHub, "Top programming languages of 2022 - github octoverse," 2022, accessed: 2024-11-07. [Online]. Available: https://octoverse.github.com/2022/top-programming-languages

[28] O. Dabic, E. Aghajani, and G. Bavota, "Sampling projects in github for msr studies," in *2021 IEEE/ACM 18th International Conference on Mining Software Repositories*. IEEE, 2021, pp. 560–564.

[29] GitHub, Inc., "Storing and sharing data from a workflow," https://docs.github.com/en/actions/writing-workflows/choosing-what-your-workflow-does/storing-and-sharing-data-from-a-workflow, 2024, accessed: 2024-08-26.

[30] N. R. Weeraddana, X. Xu, M. Alfadel, S. McIntosh, and M. Nagappan, "An Empirical Comparison of Ethnic and Gender Diversity of DevOps and non-DevOps Contributions to Open-Source Projects," *Empirical Software Engineering*, vol. 28, no. 150, p. 1–37, 2023.

[31] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix *et al.*, "Llama: Open and efficient foundation language models," in *arXiv preprint arXiv:2302.13971*, 2023.

[32] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 1877–1901.

[33] H. Zhang, Y. Sun, and Y. Qi, "Comparative analysis of pre-trained language models for natural language understanding," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 2021.

[34] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of Machine Learning Research*, vol. 21, no. 140, pp. 1–67, 2020.

[35] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," in *OpenAI Technical Report*, 2019.

[36] M. Nagappan, T. Zimmermann, and C. Bird, "Diversity in software engineering research," in *Proceedings of the 2013 9th joint meeting on foundations of software engineering*, 2013, pp. 466–476.

[37] S. Baltes and P. Ralph, "Sampling in software engineering research: A critical review and guidelines," *Empirical Software Engineering*, vol. 27, no. 4, p. 94, 2022.

[38] E. Smith, R. Loftin, E. Murphy-Hill, C. Bird, and T. Zimmermann, "Improving developer participation rates in surveys," in *2013 6th International workshop on cooperative and human aspects of software engineering*. IEEE, 2013, pp. 89–92.

[39] B. A. Kitchenham and S. L. Pfleeger, "Personal opinion surveys," in *Guide to advanced empirical software engineering*. Springer, 2008, pp. 63–92.

[40] P. Ralph, N. b. Ali, S. Baltes, D. Bianculli, J. Diaz, Y. Dittrich, N. Ernst, M. Felderer, R. Feldt, A. Filieri *et al.*, "Empirical standards for software engineering research," *arXiv preprint arXiv:2010.03525*, 2020.

[41] T. Zimmermann, "Card-sorting: From text to themes," in *Perspectives on data science for software engineering*. Elsevier, 2016, pp. 137–141.

[42] J. Sallou, T. Durieux, and A. Panichella, "Breaking the silence: the threats of using llms in software engineering," in *Proceedings of the 2024 ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results*, 2024, pp. 102–106.

[43] L. Da Silva, J. Samhi, and F. Khomh, "Chatgpt vs llama: Impact, reliability, and challenges in stack overflow discussions," *arXiv preprint arXiv:2402.08801*, 2024.

[44] C. S. Xia, Y. Wei, and L. Zhang, "Automated program repair in the era of large pre-trained language models," in *2023 IEEE/ACM 45th International Conference on Software Engineering*. IEEE, 2023, pp. 1482–1494.

[45] L. Chen, M. Zaharia, and J. Zou, "How is chatgpt's behavior changing over time?" *arXiv preprint arXiv:2307.09009*, 2023.

[46] W. Ye, M. Ou, T. Li, X. Ma, Y. Yanggong, S. Wu, J. Fu, G. Chen, H. Wang, J. Zhao *et al.*, "Assessing hidden risks of llms: an empirical study on robustness, consistency, and credibility," *arXiv preprint arXiv:2305.10235*, 2023.

[47] M. A. Bakker, M. J. Chadwick, H. R. Sheahan, M. H. Tessler, L. Campbell-Gillingham, J. Balaguer, N. McAleese, A. Glaese, J. Aslanides, M. M. Botvinick, and C. Summerfield, "Fine-tuning language models to find agreement among humans with diverse preferences," in *Proceedings of the 36th International Conference on Neural Information Processing Systems*, ser. NIPS '22. Red Hook, NY, USA: Curran Associates Inc., 2024.

[48] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. H. Chi, Q. V. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," in *Proceedings of the 36th International Conference on Neural Information Processing Systems*, ser. NIPS '22. Red Hook, NY, USA: Curran Associates Inc., 2024.

[49] B. Xu, A. Yang, J. Lin, Q. Wang, C. Zhou, Y. Zhang, and Z. Mao, "Expertprompting: Instructing large language models to be distinguished experts," *CoRR*, vol. abs/2305.14688, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2305.14688

[50] Y. L. Liu *et al.*, "Responsible AI considerations in text summarization research: A review of current practices," in *Findings of the Association for Computational Linguistics*. Association for Computational Linguistics, 2023, p. 413.

[51] C. Vassallo, S. Proksch, T. Zemp, and H. C. Gall, "Every build you break: developer-oriented assistance for build failure resolution," *Empirical Software Engineering*, vol. 25, pp. 2218–2257, 2020.

[52] T. Rausch, W. Hummer, P. Leitner, and S. Schulte, "An empirical analysis of build failures in the continuous integration workflows of java-based open-source software," in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories*. IEEE, 2017, pp. 345–355.

[53] A. LeClair and C. McMillan, "A dataset for studying the evolution of source code summarization," in *Proceedings of the 2019 IEEE/ACM 16th International Conference on Mining Software Repositories*. IEEE, 2019, pp. 377–388.

[54] L. Moreno *et al.*, "Automatic software summarization: A systematic literature review," *Journal of Systems and Software*, vol. 140, pp. 62–85, 2018.

[55] A. Tarar *et al.*, "An empirical study on bug report summarization," in *Proceedings of the 2019 35th IEEE International Conference on Software Maintenance and Evolution*. IEEE, 2019, pp. 103–113.

[56] L. Zhang *et al.*, "Rencos: Improving code summarization with retrieved similar codes," in *Proceedings of the 2020 ACM/IEEE 42nd International Conference on Software Engineering*. ACM, 2020, pp. 90–100.

[57] A. Stapleton *et al.*, "Human vs. machine-generated summaries: A comprehension study," in *Proceedings of the 2020 ACM/IEEE International Conference on Software Engineering*. ACM, 2020, pp. 232–242.

[58] S. Roy *et al.*, "Reassessing the use of bleu and meteor in source code summarization tasks," *Empirical Software Engineering*, vol. 26, no. 1, pp. 1–23, 2021.

[59] S. Banerjee and A. Lavie, "Meteor: An automatic metric for mt evaluation with improved correlation with human judgments," in *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, 2005, pp. 65–72.

[60] M. Haque *et al.*, "Semantic similarity metrics for evaluating code summarization techniques," in *Proceedings of the 2022 IEEE International Conference on Software Maintenance and Evolution*. IEEE, 2022, pp. 320–330.

[61] G. Mastropaolo *et al.*, "Side: A contrastive learning metric for code summarization evaluation," in *Proceedings of the 2024 ACM/IEEE International Conference on Software Engineering*, 2024.

[62] S. Iyer *et al.*, "Summarizing source code using neural attention models," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, 2016, pp. 207–215.

[63] V. Kumar *et al.*, "Fedllm: Federated learning-based large language models for code summarization," *Journal of Software Engineering*, 2024.

[64] L. Moreno *et al.*, "Automatic generation of natural language summaries for java classes," in *Proceedings of the 2013 IEEE/ACM 28th International Conference on Automated Software Engineering*, 2013, pp. 230–240.

[65] P. McBurney and C. McMillan, "Automatically summarizing java methods: A context-based approach," in *Proceedings of the 2016 IEEE/ACM 38th International Conference on Software Engineering*, 2016, pp. 499–510.

[66] S. Rastkar *et al.*, "Summarizing software artifacts: A bug report case study," in *Proceedings of the 2014 ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*, 2014, pp. 110–120.

[67] S. Haiduc *et al.*, "On the use of automated text summarization techniques for summarizing source code," in *Proceedings of the 2010 ACM/IEEE 32nd International Conference on Software Engineering*, 2010, pp. 223–233.

[68] L. Dabrowski *et al.*, "A systematic review of app review analysis in software engineering," *Journal of Systems and Software*, vol. 190, p. 110789, 2022.

[69] L. Nazar *et al.*, "Summarizing software artifacts using machine learning: a comprehensive review," *Journal of Software: Evolution and Process*, vol. 28, pp. 170–188, 2016.

[70] S. Panichella *et al.*, "Summarization techniques for software artifacts: a comprehensive review," *ACM Computing Surveys*, vol. 50, no. 2, pp. 1–34, 2018.