How do Developers Talk about GitHub Actions? Evidence from Online Software Development Community

Yang Zhang*
yangzhang15@nudt.edu.cn
National University of Defense Technology
Changsha, China

Yiwen Wu*
wuyiwen14@nudt.edu.cn
National University of Defense Technology
Changsha, China

Tingting Chen
chentingting20@nudt.edu.cn
National University of Defense Technology
Changsha, China

Tao Wang†
wangtao2005@nudt.edu.cn
National University of Defense Technology
Changsha, China

Hui Liu
hliu@nudt.edu.cn
National University of Defense Technology
Changsha, China

Huaimin Wang
hmwang@nudt.edu.cn
National University of Defense Technology
Changsha, China

ABSTRACT

Continuous integration, deployment and delivery (CI/CD) have become cornerstones of DevOps practices. In recent years, GitHub Action (GHA) has rapidly replaced the traditional CI/CD tools on GitHub, providing efficiently automated workflows for developers. With the widespread use and influence of GHA, it is critical to understand the existing problems that GHA developers face in their practices as well as the potential solutions to these problems. Unfortunately, we currently have relatively little knowledge in this area. To fill this gap, we conduct a large-scale empirical study of 6,590 Stack Overflow (SO) questions and 315 GitHub issues. Our study leads to the first comprehensive taxonomy of problems related to GHA, covering 4 categories and 16 sub-categories. Then, we analyze the popularity and difficulty of problem categories and their correlations. Further, we summarize 56 solution strategies for different GHA problems. We also distill practical implications of our findings from the perspective of different audiences. We believe that our study contributes to the research of emerging GHA practices and guides the future support of tools and technologies.

CCS CONCEPTS

• **General and reference** → **Empirical studies**; • **Software and its engineering** → **Software development process management**; **Software maintenance tools**.

KEYWORDS

GitHub Actions, Empirical Study, Stack Overflow

* Yang Zhang and Yiwen Wu are both first authors, and contributed equally to this work.
† Tao Wang is the corresponding author.

ICSE '24, April 14–20, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0217-4/24/04…$15.00
https://doi.org/10.1145/3597503.3623327

ACM Reference Format:
Yang Zhang*, Yiwen Wu*, Tingting Chen, Tao Wang†, Hui Liu, and Huaimin Wang. 2024. How do Developers Talk about GitHub Actions? Evidence from Online Software Development Community. In *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE '24), April 14–20, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3597503.3623327

1 INTRODUCTION

With the dramatic increase in the adoption of Continuous integration (CI) and Continuous deployment/delivery (CD) in software development [11, 56], GitHub, one of the most widely used source code repository providers, integrates with many third-party CI/CD tools [10, 57], such as Travis CI [2], Jenkins [1], and Docker [53, 54]. To facilitate a state-of-the-art CI/CD workflow platform in-house, GitHub announced the beta version of GitHub Actions[1] (GHA) in October 2018 and publicly released GHA product in November 2019 based on popular demand. GHA allows developers to create or leverage existing '*actions*' (a reusable extension) to customize workflows and make building, testing, deployment and management of software projects more automated [11]. Due to its deep integration into GitHub, GHA can be used for a variety of purposes beyond just running test suites and releasing new versions, as in the case of traditional CI/CD services [16]. These include facilitating code reviews, communication, dependency and security monitoring and management, etc.

Recently, GHA has replaced the traditional CI tools on GitHub as the dominant CI service [40]. Also, GitHub provides a large marketplace, allowing developers to easily adapt existing *action*, or create new ones based on predefined templates. As of March 2023, GHA marketplace has been growing at a high rate, reaching 18K of reusable *action*[2]. At the same time, a number of developers use question and answer (Q&A) forums such as Stack Overflow[3] (SO) to discuss problems regarding GHA [45]. Our initial observation finds that the number of GHA-related questions and answers in SO has increased dramatically in the last few years[4].

[1] https://github.blog/changelog/2018-10-16-github-actions-limited-beta/
[2] https://github.com/marketplace?category=&query=&type=actions&verification=
[3] https://stackoverflow.com/
[4] The distribution of posts has been provided in our replication package.

However, developers have encountered many challenges along with the mass adoption of GHA. For example, Wessel et al. [52] found that the largest group of discussion threads that mention GHA is predominantly asking for help in debugging GHA errors. In the survey by Saroar et al. [45], developers complained that GHA failed too often (especially on the Windows runner) with no clear explanation, and these errors were frequently not reproducible. Therefore, investigating the problems in GHA usage and understanding how GHA problems are resolved by developers in practice are important. Such analysis can benefit both the academic and industry communities. Researchers can perform studies towards solutions for the frequently reported problems of GHA, and practitioners can be allocated to address the most frequent and challenging problems identified through the empirical study.

A few studies have recently targeted this direction [8, 12, 15, 35, 52]. For example, Kinsman et al. [35] discussed the use of GHA and its impact on development practice. Chen et al. [12] analyzed the usage details of GHA, including its component scale and action sequences. Wessel et al. [52] investigated the impact of GHA on pull request process. Benedetti et al. [8] dissected the problems related to GHA security. Decan et al. [15] found that most of the GHA workflows used an outdated *action* release. Nevertheless, these efforts target only some specific problems of GHA and lack in-depth studies on GHA problem categories and characteristics. Furthermore, to our best knowledge, few attempts so far provide solutions for problems encountered by developers during GHA practices.

To fill this gap, we conduct a large-scale empirical study to explore developers' discussions of GHA practices by collating data-driven evidence from the online software development community. Our goal is to answer the following research questions:

RQ1: (GHA problems) *What GHA problems do developers discuss frequently during their development process?* - RQ1 aims to systematically identify and taxonomically classify the problems that GHA developers encounter.

RQ2: (Problem characteristics) *What problems are more popular/difficult among GHA developers?* - RQ2 seeks to analyze the characteristics of the identified problem categories in terms of popularity and difficulty.

RQ3: (Solution strategies) *What are the common solution strategies for different GHA problems?* - RQ3 aims to explore the solution strategies in each problem category, providing insights about principled ways to resolve GHA problems.

To that end, we focus on SO, one of the most popular Q&A communities, and GitHub, the largest public code repository host, enabling data mining for our study. Specifically, leveraging GHA-related tags, we collect a dataset of 6,590 SO questions (with 2,471 accepted answers) and 315 GitHub issues related to GHA. Based on the dataset, we manually summarize GHA problems from SO questions and GitHub issues' descriptions. We construct a problem hierarchy by repeated grouping of similar problems into subcategories and categories. Then, we measure the popularity and difficulty of our GHA problem categories using several well-known metrics [6, 7, 30, 38] and analyze their correlation. Additionally, we analyze the GHA-related accepted answers and distill common solution strategies for different GHA problems encountered by developers. Finally, we discuss the implications of our findings for GHA developers, researchers, and service/tool providers.

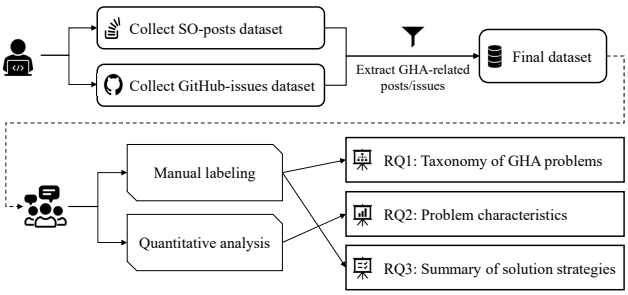

Figure 1: Overview of our methodology.

In summary, our paper has the following contributions:

- We present a comprehensive taxonomy of GHA problems based on SO questions and GitHub issues, including 4 categories and 16 sub-categories.
- We analyze the popularity and difficulty of GHA problem categories, and find that GHA has accumulated a lot of unresolved problems. We also find that categories' difficulty and popularity are negatively correlated.
- We summarize 56 solution strategies for different problems of GHA, which can be adapted to facilitate manual and automated resolution of GHA problems.

We publicly release a replication package at https://github.com/yangzhangs/gha-so-artifact.

The remainder of the paper is organized as follows. Section 2 elaborates on our study methodology. In Sections 3, 4, and 5, we present our study results. Section 6 discusses the implications and threats to validity. We introduce the related works in Section 7 and conclude this paper in Section 8.

2 METHODOLOGY

To understand how developers talk about GHA, we analyze the relevant SO posts and the relevant GitHub issues. Figure 1 illustrates the overview of the methodology used in our study.

2.1 Data Collection

Our empirical study involves mining two types of sources: (a) SO data, i.e., Stack Overflow posts; and (b) GitHub data, i.e., issues from GitHub projects.

2.1.1 SO-posts dataset. First, we extract the post data leveraging the Stack Exchange Data Explorer[5]. This initial dataset D covers 3,545,605 questions and answers posted over a time span of over 4 years from October 1, 2018 to October 31, 2022. Among these posts, 1,625,053 (45.8%) are questions and 1,920,552 (54.2%) are answers, and 600,419 (36.9%) questions have accepted answers.

Next, we identify all GHA-related posts by checking if they have GHA-related tags. Similar to previous work [30], we develop a set of GHA-related tags T as follows: (a) We set the initial set of tags as T_0={*"github-action", "github-actions"*}. (b) Then, we go through the dataset D and extract a subset of posts P whose tags match at least one tag in T_0. (c) We obtain the tags of all posts in P (denoted as T_1) to extend the initial tag set T_0, including as many relevant tags as possible. (d) Following prior studies [6, 31, 39], we use two

[5]https://data.stackexchange.com/stackoverflow/query/

heuristics S and R to refine T_1 by keeping tags that are significantly relevant to GHA and excluding others. The S and R measure the significance and relevance of a tag t in T_1 to GHA, respectively:

$$(Significance)\ S(t) = \frac{\text{number of posts with tag } t \text{ in } P}{\text{number of posts with tag } t \text{ in } D} \quad (1)$$

$$(Relevance)\ R(t) = \frac{\text{number of posts with tag } t \text{ in } P}{\text{number of posts in } P} \quad (2)$$

A tag t is significantly relevant to GHA if its two heuristics are higher than specific thresholds. To avoid omitting relevant tags, we adopt the lowest thresholds used in previous work [6], and only the tags whose S is higher than 0.05 and whose R is higher than 0.005 are kept in T_1. After these processing steps, we obtain the final tag set T={*"github-actions", "github-action", "github-package-registry", "building-github-actions", "github-actions-runners", "github-actions-self-hosted-runners"*}. Then, we extract all posts with at least one tag in T. The final SO-posts dataset for our study consists of 6,590 questions (Q_S) with 2,471 accepted answers (A_S).

2.1.2 GitHub-issues dataset. We use the GitHub Search API[6] to obtain the GHA-related issues. Specifically, we search issues tagged with t∈{*"github action", "github-action", "github actions", "github-actions"*} and created between October 1, 2018 and October 31, 2022. This yields 668 issues from 230 repositories. Prior work [34] pointed out that there is a risk of including inactive or abandoned repositories when working with GitHub data. To mitigate this risk, we only consider non-forked repositories that have more than 10 commits in the last three months (since August 1, 2022). We also remove 10 issues that are not written in English. Finally, we obtain 315 GHA-related issues (Q_G) from 89 repositories. 217 of these issues (A_G) are closed (resolved).

2.2 Manual Labeling

To distill the GHA problems (**RQ1**) and solution strategies (**RQ3**), we manually label the question data (Q_S and Q_G) and accepted answer data (A_S and A_G). The manual labeling is conducted by two annotators, i.e., the first two authors with 4-year experience in CI/CD, following the open coding procedure [46]. There are no predefined categories and categories are developed during the labeling process. Before the labeling, we conduct a training session based on the official GHA documentation[7] to help annotators familiarize themselves with the key GHA components and knowledge.

2.2.1 GHA problems classification. First, we follow prior work [37]'s labeling process and conduct a two-round pilot labeling on a part of the question data (with 200 random samples in each round). In the first round, two annotators independently tag each question item with a label. The initial inter-rater agreement is 0.45, measured by Cohen's kappa coefficient [13]. Then, they compare the labeling results during an in-person meeting and introduce an arbitrator (i.e., the fourth author with 10 years of CI/CD experience) to resolve the conflicts. After discussion, a set of labels is summarized to mitigate bias and ensure labeling consistency. Based on these labels, the annotators perform the second-round labeling independently. The kappa score of this round is increased to 0.87, which indicates a high confidence of agreement. The annotators and arbitrator resolve some inconsistencies and further refine the labels. This pilot analysis helps us to (a) reach an agreement on the labeling process and (b) reduce inconsistencies among the authors.

Next, the annotators work together to formally label the remaining 6,505 question items following the same process used in the pilot analysis. All the authors check the final results and group similar codes into categories in multiple iterations. Finally, we construct a hierarchical taxonomy of GHA problems.

2.2.2 Solution strategies extraction. Same to the method in RQ1, two annotators conduct a two-round pilot analysis (200 random samples per round) before the large-scale formal labeling of the accepted answer data. During labeling, they manually mark each accepted answer item with a short descriptive phrase. When an agreement cannot be reached between the two annotators, all the authors participate in the discussion and resolve the conflict. Based on the final labeling results, all the authors proceed to group similar phrases into solution strategies. The grouping process is iterative, in which they continuously go back and forth between the phrases and accepted answer items to refine the solution strategies. They follow this procedure until there is an agreement on all solution strategies. After such a rigorous process, we get a refined list of solution strategies for different GHA problems.

2.3 Quantitative Analysis

To explore the problem characteristics (**RQ2**), we use several well-known metrics used by prior work [6, 7, 30, 38] to measure the *popularity* and *difficulty* of GHA problem categories. Compared to the SO questions in our dataset, GitHub issues lack relevant characteristic data such as view and score. Thus, we only focus on the SO questions in this RQ.

2.3.1 Popularity metrics. There are 4 metrics for popularity measurement, including (a) *avgView*, the average number of views for all the questions of a category; (b) *avgFav*, the average number of favorites for all the questions of a category; (c) *avgScore*, the average score for all the questions of a category; and (d) *avgAns*, the average number of answers for all the questions of a category. Intuitively, a category with a higher number of views, favorites, and answers and a higher score is more popular.

2.3.2 Difficulty metrics. We use 5 difficulty metrics, including (a) *ansRate*, the percentage of questions of a category with at least one answer; (b) *acceptRate*, the percentage of questions of a category that have accepted answers; (c) *timeFA*, the median time needed for questions of a category to receive the first answers, in hours; (d) *timeAA*, the median time needed for questions of a category to receive the accepted answers, in hours; and (e) *textSize*, the average number of description characters for questions of a category. Intuitively, a category is more difficult if it receives fewer answers over a longer amount of time and has longer descriptive text.

3 RQ1: TAXONOMY OF GHA PROBLEMS

Figure 2 provides the hierarchical taxonomy of GHA problems, including 4 categories, 16 sub-categories, and 50 frequent problem examples. The broad scope of GHA problems indicates the prevalence and the diversity of GHA-related challenges. Moreover, we

[6] E.g., https://api.github.com/search/issues?q=label:"github actions"+is:issue
[7] https://docs.github.com/actions

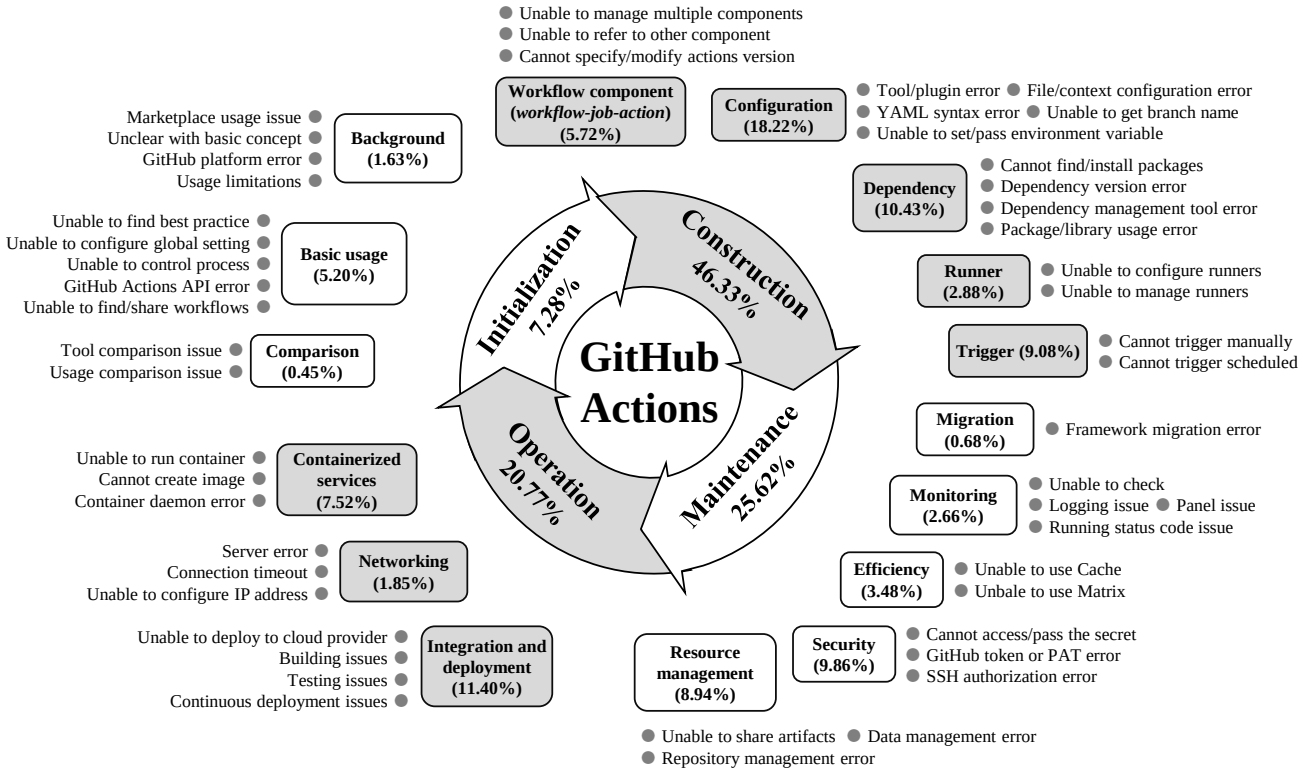

Figure 2: The taxonomy of GHA problems.

find that developers ask the most questions in the *Construction* category (46.33%), followed by *Maintenance* (25.62%), *Operation* (20.77%), and *Initialization* (7.28%).

> **Finding 1.** *Developers discuss a wide range of GHA problems, which can be grouped into a hierarchical taxonomy of 4 categories and 16 sub-categories. Problems in the Construction category are discussed the most by developers (46.33%).*

We next discuss each of the categories along with their sub-categories and examples for a better explanation of taxonomy.

3.1 Initialization

Initialization category represents the problems in the initial phase where developers look for the primary GHA-related knowledge.

3.1.1 Background (1.63%). The problems in *Background* include lack of clarity on basic GHA-related concepts (e.g., question Q807 in our dataset: "*What is 'uses' directive in GitHub Actions used for?*"), usage limitations (e.g., Q982: "*What are GitHub Actions minutes/-month?*"), and Marketplace usage issue (e.g., Q2530: "*How can I scrape more than 1,000 results from GitHub Marketplace?*"). These questions suggest that developers, especially inexperienced ones, are unfamiliar with GHA and try to have a thorough understanding of GHA-related concepts.

3.1.2 Basic usage (5.20%). It covers the problems related to process control, global settings, GHA APIs, best practices, and workflow finding/sharing. Some examples include Q13: "*How do I re-run GitHub Actions?*", Q466: "*Is there a way to delete a GitHub workflow?*", Q638: "*GitHub Actions v4 API?*", and Q1430: "*Best Practices for building Releases with GitHub Actions/Workflows*". These questions indicate that the existing official GHA documentation is still inadequate in guiding developers in basic usage.

3.1.3 Comparison (0.45%). As aforementioned, GHA replaces the traditional CI/CD tools on GitHub [40]. However, some developers are still unclear about GHA features and seek to compare GHA with other tools, e.g., GitLab CI (Q75: "*Difference between GitLab CI and GitHub Actions Beta*"), Jenkins (Q1: "*What are the differences between GitHub Actions and other CI tools like Jenkins?*"), and Azure Pipeline (Q3377: "*Does GitHub Actions have an equivalent concept to Azure Pipeline's custom build number syntax?*").

> **Finding 2.** *In the initial phase, developers look forward to obtaining the guidance of GHA's background and basic usage (6.83% in total). Also, the differences in features between GHA and other CI/CD tools confuse developers.*

3.2 Construction

Construction category represents the problems in the process of building a GHA workflow, where developers encounter the most problems. Specifically, a *workflow* consists of one or more *jobs*. A *job* is a set of steps executed on the same *runner* that defines a list of *steps* that will be executed sequentially, with the *steps* being the smallest work units in the workflow. An *workflow* is configured in a YAML file that can be stored in a GitHub repository. Once a code

change is pushed, or a pull request is made, a *trigger event* can be set in GHA to trigger the workflow [25].

3.2.1 Workflow component (5.72%). This sub-category focuses on GHA workflow components' usage and management errors. Some questions like Q320: "*How can I create a job to run after all jobs in GitHub Actions?*", Q994: "*GitHub Actions - No jobs defined in 'jobs'?*", and Q53: "*How to automatically select the latest tagged version of a GitHub Action?*". These problems indicate that developers are experiencing challenges in assembling GHA components. GHA usually performs complex but often repetitive tasks and can be custom defined for various purposes [16, 36]. This makes it potentially difficult to choose components in the GitHub Marketplace that are suitable for use in a workflow, for example, Q2647: "*Git: Using actions/checkout@v2 instead of appleboy/ssh-action@master to clone repository to a server*".

3.2.2 Configuration (18.22%). Of all the GHA problems, configuration problems receive the most discussion from developers. This indicates that configuration plays a critical role in the GHA lifecycle. In particular, workflows are defined in the *.github/workows/* directory and use YAML syntax, having either a *.yml* or *.yaml* file extension [52]. The successful adoption and proper use of GHA workflow depends on the YAML files. However, as developers stated in the survey [45]: "*YAML is great to read, and a nightmare to write*". The most common problem we find in this sub-category is related to YAML syntax error, e.g., Q1410: "*GitHub Actions - You have an error in your yaml syntax?*" and Q1708: "*You have an error in your yaml syntax on line 10*".

In addition to YAML configuration problems, we also find some problems related to file configuration error (e.g., Q6416: "*How to set a env file in a GitHub Repository for a React App*") and tool/plugin error (e.g., Q1022: "*NextJS baseUrl configuration not working on GitHub Actions?*"). Input/output and environment parameter's passing/usage problems are also discussed by developers. Some examples like Q149: "*How to pass variable between two successive GitHub Actions jobs?*", Q174: "*How to make GitHub Actions communicate with each other?*", and Q49: "*GitHub Actions, how to share a calculated value between job steps?*". There are also some questions about the context configuration error, for instance, Q3428: "*GitHub Context Variables Not Evaluating for Reusable Workflow Reference*".

3.2.3 Dependency (10.43%). The operation of GHA workflows depends on plugins/frameworks, installed libraries/packages, running system architecture, etc. Dependency acquisition is a common problem in this sub-category (e.g., Q74: "*How to apt-get install in a GitHub Action?*"). Previous study found that dependency errors are the most difficult in the package management [32], which are usually due to the developer using the wrong version of a package or tool, or installing it in an incorrect location. The cause of the error can usually be initially determined from the reported error message, for example, Q546: "*GitHub Actions for specific php version error*", Q2427: "*GitHub Action dependency installation failing for Windows*", and Q2808: "*Error run GitHub Action Execution failed for task ':app:kaptDebugKotlin'*". Also, GHA workflow often involves package and dependency management tools such as Maven, Gradle, npm, and Yarn. Their usage errors are common problems discussed by developers. Two examples are Q3583: "*How to install maven dependency from Github packages?*" and Q197: "*Gradle fails to download package from Github Package Registry*".

3.2.4 Runners (2.88%). The machines that execute jobs in a GHA workflow are called *runners*. A runner could, for instance, install testing software, clone the repository locally, and then execute commands that evaluate the code. GHA workflows can be run on fresh, newly-provisioned virtual machines using GitHub's runners, which are available for Ubuntu Linux, Windows, and macOS (i.e., GitHub-hosted runners). Developers can host the runner themselves (i.e., self-hosted runners) if they need a specific hardware configuration or a different operating system. Compared to GitHub-hosted runners, self-hosted runners provide more control over the hardware, operating system, and software tools. Each job in the workflow has an independent runner. While most jobs use GitHub-hosted runners (88.26%) [12], we find that developers have more problems using self-hosted runners than GitHub-hosted runners, for instance, Q2494: "*How to limit CPU/Memory - GitHub Actions self-hosted runners?*", Q1547: "*Does GitHub Action supports the on-demand self-hosted runner?*", and Q2969: "*GitHub API permission name for self-hosted GitHub Actions runners?*". This suggests that using self-hosted runners is more challenging for developers. Also, runner management is an important problem, including runner reuse (e.g., Q228: "*Reuse GitHub Actions self-hosted runner on multiple repositories*"), multi-runner collaboration (e.g., Q2928: "*GitHub Actions how to configure two runners in two servers*"), and runner update (e.g., Q286: "*How to auto-update GitHub Action runner?*").

3.2.5 Trigger (9.08%). Trigger events are specific activities in the repository that trigger a workflow to run, which enable CI tools to run custom workflows in a dynamic environment that is constantly changing [33]. Developers can configure external triggers using a repository dispatch webhook or many other webhooks, e.g., *deployment*, *workflow_dispatch*, and *check_run*. In our statistics, questions about automatic trigger events account for the vast majority of the *Trigger* sub-category. The events that trigger a workflow can be chosen from a large list of events corresponding to the different ongoing activities within a repository (e.g., commits pushed, pull requests created or updated, comments created) [16]. Two instances related to trigger issues are Q126: "*Trigger a GitHub Action when another repository creates a new release*" and Q145: "*How to trigger a GitHub Action job or workflow based on modified file or directory on last commit?*". Developers also try to trigger workflows manually to keep them running on demand, for example, Q167: "*How to trigger a step manually with GitHub Actions?*" and Q3082: "*Manually triggered GitHub Action is always being skipped*".

Finding 3. *Construction category covers the most common problems encountered by developers, including workflow component, configuration, dependency, runner, and trigger. In particular, the problems in the Configuration sub-category have been discussed the most (18.22%).*

3.3 Maintenance

Maintenance category represents the problems in the management or optimization process of GHA, which aims to keep GHA workflows running correctly, efficiently, and securely.

3.3.1 Migration (0.68%). As Golzadeh et al. [40] observed, migration indicates that GHA is rapidly replacing the traditional CI/CD tools on GitHub. However, developers may encounter problems during migrations from other frameworks (e.g., Azure Pipelines, CircleCI, GitLab CI/CD, Jenkins, Travis CI, etc.) to GHA, such as Q1829: "*Migrating to GitHub Actions from Travis keeps failing*", Q787: "*Using GitHub Actions like GitLab CI/CD*", and Q2723: "*How do I convert Travis CI configuration to GitHub Actions and test with several versions of Python?*". It may be necessary for service/tool providers to break the barriers between different CI/CD tools and provide standardized DevOps services for developers.

3.3.2 Monitoring (2.66%). This sub-category aims to provide support for system behavior data analysis and workflow maintenance. Some important problems are related to the logging issue (e.g., Q191: "*How to see logging of canceled steps in GitHub Actions?*"). The robustness of the logging and monitoring framework is essential for mature software [41]. Developer in (Q521: "*How do I 'unrun' all scheduled GitHub Actions at once?*") emphasized that "*Logging is important when something bad happens...*". We also find some other problems related to the panel issue (e.g., Q958: "*Automating boards in GitHub*"), checking issue (e.g., Q3343: "*How to check multiple workflows are successful or not in GitHub Actions?*"), and running status code issue (e.g., Q1810: "*How can I get the Passing/Failing status of a GitHub Action Workflow?*").

3.3.3 Efficiency (3.48%). Modern software is evolving rapidly and efficiency improvement becomes more important for software development [17]. Jobs on GitHub-hosted runners start in a clean runner image and must download dependencies each time, causing increased network utilization, longer runtime, and higher cost. Previous studies [35, 52] found that *action/cache* is one of the most commonly used actions across repositories, which allows caching of dependencies and build output to improve workflow execution time. Hence, developers can create and use caches for frequently reused files to help speed up the time it takes to recreate them, making workflows faster and more efficient. Three cache-related problem examples are Q8: "*How do I cache steps in GitHub Actions?*", Q36: "*GitHub-actions: cache repo to speed up maven builds?*", and Q1373: "*Caching npm dependency with GitHub Action*". The *matrix* is also an important tool for increasing efficiency [27]. It allows developers to create multiple jobs by making variable substitutions in the job definition. In practice, developers encounter problems such as Q25: "*GitHub Action build matrix: How does the $ work?*", Q387: "*GitHub Actions: How to use strategy/matrix with a script?*", and Q1619: "*How to access matrix variables in Github actions*".

3.3.4 Security (9.86%). The GHA ecosystem is likely to suffer from security issues [16]. In particular, as the number of reusable actions continues to grow rapidly, the attack surface of security issues increases by several orders of magnitude due to the widespread reliance on reusable software repositories [5, 14]. Benedetti et al. [8] developed and applied a security assessment automation tool that uncovered 25K security issues in GHA. Also, developers expressed concerns about the GHA security, for example, Q66: "*GitHub Actions: Are there security concerns using an external action in a workflow job?*" and Q2047: "*Are GitHub Actions safe and private for PRIVATE Repositories?*". Through our observation, GitHub secret and authentication errors are two common problems in the *Security* sub-category. GitHub secrets, which are encrypted variables created in an organization, repository, or repository environment, can be used in GHA workflows. In the process of using GitHub secrets, developers encounter problems such as Q7: "*How to access the value of SECRETS in GitHub Actions?*" and Q33: "*How to set secrets in GitHub Actions?*".

Authentication is related to a set of discussions, i.e., how to allow a user or entity to verify its identity or credentials to access GitHub resources and functionalities, involving GitHub token (or PAT: Personal Access Token) and SSH. By default, each repository always has a secret called *GITHUB_TOKEN*, which is a GitHub API token with write access to that repository (e.g., Q2690: "*How to use GITHUB_TOKEN to clone a private repository?*"). Moreover, developers can use the SSH protocol to connect and authenticate to remote servers or services. With SSH keys, they can connect to GitHub without supplying the username and personal access token each time they visit, and they can even use SSH keys to sign commit files. However, developers sometimes have SSH-related problems, such as Q154: "*How to use ssh identity file with GitHub Actions?*" and Q402: "*Self-connecting via SSH on GitHub Actions*".

3.3.5 Resource management (8.94%). This sub-category consists of artifact management, repository management, and data management problems. An artifact is a file or collection of files produced during a workflow run, allowing developers to share data between jobs and store data once a workflow is complete. Developers usually ask questions about artifacts versioning (e.g., Q5: "*How to version build artifacts using GitHub Actions?*"), uploading (e.g., Q2456: "*How to upload an artifact (a pdf file associated to a tex file) in GitHub Action to an external repository in GitHub?*"), downloading (e.g., Q1172: "*Is it possible to download GitHub-Actions artifacts directly?*"), deleting (e.g., Q2833: "*How to delete old snapshot artifacts from GitHub packages*"), and sharing (e.g., Q455: "*Share artifacts between workflows/ GitHub Actions*"), as well as some other common issues (e.g., Q924: "*GitHub Actions Build Artifact Folder Location*").

Repository management problems involve branch creation (e.g., Q585: "*How to create an orphan (unrelated) branch with GitHub Actions?*"), pull request rejection (e.g., Q141: "*How to reject a pull request if tests are failed GitHub Actions?*"), and issue management (e.g., Q1216: "*How to check if an issue has no labels in a GitHub workflow?*"). Data management issue is also an important problem, for example, Q2664: "*Github action not uploading data*" and Q5663: "*Get the data from Github Action Artifacts*".

Finding 4. *The maintenance of GHA is a challenge for developers, especially in the management of security issues (9.86%) and resources (8.94%). And in the Security sub-category, two common problems discussed by developers are related to GitHub secret and authentication.*

3.4 Operation

Operation category represents the problems that arise during the execution and deployment of GHA workflows.

3.4.1 Containerized services (7.52%). Containerized services provide developers with an easy and portable way to test or operate the applications in their workflows. Developers can configure service

containers for each job in their workflow, connecting databases, web services, memory caches, and other tools to their workflow. This sub-category mainly includes several problems such as container image creation errors and Docker image usage errors.

The container image creation problem deals with image pulling (e.g., P892: "*How to pull a Docker image in GitHub Actions for a compute engine VM?*"), image building (e.g., Q1668: "*Unable to Build Docker Image Using GitHub Actions*"), image publishing (e.g., Q1546: "*How to build and push a Docker image?*"), and image caching (e.g., Q1132: "*How to cache a Docker image from a GitHub Action?*"). We also find some other issues during container usage. These issues are often related to the functions and features of Docker itself, so we put them into a separate category, namely Docker image usage error. Specifically, we observe that developers have difficulties using Docker in GHA, involving Docker daemon (e.g., Q3057: "*Docker daemon access denied while running from GitHub Actions*"), Docker volume (e.g., Q199: "*Docker container volume does not get mounted in GitHub Actions*"), parameter passing (e.g., Q1762: "*Pass Variable From GitHub Action to Docker image build*"), and version control (e.g., Q282: "*Using a specific Docker version in GitHub Actions*").

3.4.2 Networking (1.85%). Networking covers a wide range of problems developers may encounter with network connections, involving server error (e.g., Q1024: "*Laravel & PEST Php | GitHub Actions | cURL error 7: Failed to connect to localhost port 8000*"), network bridging (Q1460: "*How to bridge github ci network and docker container network*"), and connection timeout (e.g., Q1400: "*Failed to execute goal maven-antrun-plugin - Connection timed out on GitHub Actions*"). Also, configuring IP address is an important problem, such as static IP (Q2190: "*is it possible to use static ip when using GitHub Actions*") and VM IP (Q1723: "*Does every Virtual Machine that Github spins up for a workflow run get a new IP address?*"). The specific causes of networking errors are complex and depend on a variety of factors that require further analysis by researchers.

3.4.3 Integration and deployment (11.40%). CI/CD is the practice of using automation to release and deploy software updates [47]. Developers can set up a GHA workflow to build a containerized application and deploy their software product to the cloud provider, such as Amazon Elastic Container Service (ECS), Azure, and Google Kubernetes Engine. Some examples are Q715: "*Auto Deployment .Net Framework Application on AWS EC2*" and Q1136: "*I can't deploy my node js application to Microsoft Azure*".

To verify that the product is working as expected, the workflow can automatically build the code and run tests before deployment. During the building phase, developers are prone to encounter production build error (e.g., Q296: "*Cannot build for production in GitHub Actions using webpack*") and environment incompatibility issues (e.g., Q3254: "*ExoPlayer:extension-av1:generateJsonModelDebug breaks with NullPointer when building on GitHub Actions*"). Moreover, testing/running timeout is a common problem, which can be caused by excessive time consumption. Some questions include Q1771: "*Session timed out or not found while executing test in Selenoid using GitHub Actions*", Q2795: "*Testcafe workflow for IE on Window on GitHub Actions: Error: The action has timed out*", and Q2925: "*Gtest discover_tests failing in GitHub Actions: Process terminated due to timeout*". Also, some deployment problems are related to instruction publish errors (Q627: "*dotnet publish and deploy with ftp GitHub Actions*"), path/location errors (Q1911: "*gh-pages deployment issue, job fails on deploy. The directory you're trying to deploy ... doesn't exist*"), publish failing (Q377: "*GitHub hosted package fails when doing npm publish*"), and POM missing (Q966: "*Parent POM Missing on deployed GitHub package*").

Finding 5. *Containerized services, networking, and integration and deployment cover the problems that arise during the operational phase. The majority (11.40%) of these challenges are thrown with integration and deployment. And developers complain about the different types of errors in the dynamic build and publish process.*

4 RQ2: PROBLEM CHARACTERISTICS

Table 1 and Table 2 summarize characteristic metrics of different GHA problem categories, respectively.

4.1 Category Comparison

We find differences in popularity metric values across problem categories. *Construction* category has the most average views and answers, while *Initialization* category has the most average score and *Maintenance* category has the most average favorite. Surprisingly, the smallest sub-category (0.45%), *Comparison*, is the most popular among developers, with all four popularity metrics much larger than the other sub-categories. This is probably due to the fact that the *Comparison* sub-category has the simplest questions (i.e., minimum *textSize* in Table 2). Another small sub-category *Migration* has the smallest *avgView*, *avgScore*, and *avgAns*, indicating a lack of developer attention to its problems.

Also, we find that GHA has accumulated a lot of unresolved problems. The total answer acceptance rate in our dataset is 37.56%, which is less than the rate values in other SO discussion domains (SO community: 51.97% [58], Docker: 40.36% [30], Web: 52.00% [4]). Also, the average median time needed to receive the accepted answer per category is 9.61 hours, which is much longer than the median of 21 minutes in the total SO community [38]. Thus, the GHA questions are considerably more difficult compared to the other fields. Specifically, *Operation* and *Construction* are among the most and least difficult GHA categories in terms of the *acceptRate* and *timeAA*. *Comparison* sub-category still has the highest acceptance rate of answers (45.16%), which indicates that the problems in this sub-category are better solved on SO than others. However, *Migration* sub-category has the lowest *acceptRate*, as well as the longest *timeFA* and *timeAA*. This evidence may suggest that there may be a correlation between popularity and difficulty metrics.

Finding 6. *Only a small percentage of the GHA problems have been resolved (less than 38%), and they take longer time to get acceptable answers than the typical SO questions. Developers focus more on the Construction category and encounter the most difficulties in the Operation category. Comparison and Migration are among the most and least popular/easy GHA sub-categories.*

4.2 Correlation Analysis

We follow prior work [3] and use the Spearman's rank correlation [55] to verify the correlations between the popularity and difficulty metrics. We choose Spearman's rank correlation since it does

Table 1: Popularity of GHA problem categories.

Category	avgView	avgFav	avgScore	avgAns
Initialization	**1,994.87**	**0.41**	**3.88**	**0.95**
Background	956.31	0.23	1.76	0.86
Basic usage	2,178.55	0.40	4.09	0.95
Comparison	3,673.26	1.16	9.19	1.23
Construction	**2,779.28**	**0.48**	**3.72**	**1.02**
Workflow component	2,849.41	0.56	4.24	0.98
Configuration	3,471.75	0.53	4.03	1.13
Dependency	1,480.30	0.25	2.18	0.86
Runner	2,434.89	0.51	3.90	0.94
Trigger	2,868.62	0.56	4.37	1.03
Maintenance	**2,305.69**	**0.51**	**3.19**	**1.02**
Migration	617.98	0.25	1.05	0.73
Monitoring	1,894.35	0.38	2.63	0.76
Efficiency	2,473.40	0.78	4.78	1.10
Security	2,352.01	0.48	2.90	1.12
Resource management	2,425.97	0.49	3.24	0.97
Operation	**1,309.45**	**0.28**	**1.70**	**0.83**
Containerized services	1,690.91	0.36	2.21	0.88
Networking	1,022.63	0.25	1.25	0.75
Integration and deployment	1,083.78	0.24	1.41	0.82

Table 2: Difficulty of GHA problem categories.

Category	ansRate	acceptRate	timeFA	timeAA	textSize
Initialization	**0.66**	**0.36**	**6.36**	**7.21**	**1,177.12**
Background	0.69	0.38	5.56	1.66	1,372.31
Basic usage	0.63	0.34	6.88	10.01	1,129.39
Comparison	0.81	0.45	5.16	8.07	1,010.81
Construction	**0.70**	**0.41**	**6.40**	**6.18**	**1,582.68**
Workflow component	0.66	0.39	5.19	4.80	1,371.77
Configuration	0.75	0.44	4.84	5.96	1,513.44
Dependency	0.65	0.37	16.12	16.05	2,256.18
Runner	0.62	0.33	12.38	11.11	1,685.33
Trigger	0.71	0.41	5.65	4.60	1,107.76
Maintenance	**0.68**	**0.37**	**10.63**	**10.93**	**1,530.75**
Migration	0.57	0.20	24.12	70.96	1,199.93
Monitoring	0.56	0.29	13.11	4.53	1,293.70
Efficiency	0.68	0.38	15.47	15.47	1,575.48
Security	0.75	0.40	9.31	10.12	1,707.51
Resource management	0.65	0.37	10.14	12.22	1,406.25
Operation	**0.61**	**0.31**	**13.69**	**14.13**	**2,039.55**
Containerized services	0.64	0.35	11.15	10.69	1,884.60
Networking	0.55	0.25	15.02	18.36	2,127.13
Integration and deployment	0.60	0.30	15.10	16.70	2,136.55

not have any assumption on the normality of the data distribution. As shown in Table 3, we find a statistically significant correlation between the popularity and difficulty metrics, since almost all correlations' (except the '*timeAA-avgFav*' and '*textSize-avgAns*' pairs) p-values are less than 0.1.

The average answer rate (*ansRate*) and average accepted answer rate (*acceptRate*) have significant positive correlations with all popularity metrics (all p-value<0.01), suggesting that difficult problems (i.e., lower *ansRate* and *acceptRate*) are less popular among developers, and vice-versa. There are significant negative correlations between the *timeFA* and popularity metrics (all p-value<0.05). Also, the *timeAA* has significant negative correlations with three popularity metrics (except the *avgFav*). This is consistent with the finding of prior work [43], i.e., unpopular questions may prevent them from getting expected answers. We find that there is a significant negative correlation (p-value<0.1) between *textSize* and popularity metrics (except the *avgAns*). Hence questions with longer descriptive texts are correlated with lower popularity. Like previous work [51] proposed, the current SO incentive system motivates frequent answerers well, but such frequent answerers tend to answer short and easy questions.

Finding 7. *There is a statistically significant negative correlation between the popularity and difficulty of GHA problem categories (18 out of 20 groups). In particular, unpopular problems tend to have lower answer rates and get slower answers.*

5 RQ3: SUMMARY OF SOLUTION STRATEGIES

Table 4 illustrates the solution strategies for different GHA problems. The columns "Category" and "Sub-category" (part of) are consistent with our taxonomy in Figure 2. The column "Problem Description" corresponds to the problem examples found in RQ1, and the column "Solution Strategy" briefly describes the common solution strategies.

Overall, we distill 56 solution strategies for 20 problems in 12 subcategories. Among them, the *Construction* category has the most solution strategies (50%), followed by the *Maintenance* category (23.21%) and the *Operation* category (19.64%), while the *Initialization* category has the least number of solution strategies (7.14%). Next, we elaborate on the identified solution strategies for the different GHA problem categories. Note that, due to space limitations, the accepted answer examples and detailed discussion of each solution strategy can be accessed in our replication package.

Table 3: Correlations of popularity and difficulty metrics.

co-efficient/p-value	avgView	avgFav	avgScore	avgAns
ansRate	0.701***	0.626**	0.621**	0.887***
acceptRate	0.771***	0.670**	0.677**	0.890***
timeFA	-0.630**	-0.495*	-0.603**	-0.639**
timeAA	-0.454*	-0.324	-0.480*	-0.394.
textSize	-0.385.	-0.422.	-0.556*	-0.310

*** p <0.001, ** p <0.01, * p <0.05, .p <0.1

5.1 Solutions for Initialization

We identify 4 solution strategies for the *Initialization* problems. In practice, developers complain they cannot find or share workflows (e.g., Q1411: "*GitHub Actions - is there anywhere that lists them all?*"). As a solution, they can find all of the GHA in GitHub's marketplace under the Actions filter[8]. Developers can also reuse workflow [23] and workflow templates [20] to avoid duplication when creating a workflow. When developers encounter problems related to the inability to control the GHA process (e.g., Q487: "*How to force to exit in Github Actions step*"), they can use the conditional expression [29] to allow skipping subsequent steps if an earlier step failed. Developers can also use the concurrency support [28] to ensure that only a single job or workflow using the same concurrency group will run at a time.

5.2 Solutions for Construction

We identify 28 frequent solution strategies for the *Construction* problems. According to statistics, 17.86% of the solution strategies are related to the *Workflow component* sub-category. For the problem of being unable to manage multiple components (e.g., Q16: "*Can*

[8]https://github.com/marketplace?type=actions

Table 4: Solution strategies for GHA problems.

Category	Sub-category	Problem Description	Solution Strategies
Initialization (7.14%)	Basic usage (7.14%)	Unable to find/share workflows	① Reuse workflow and workflow templates ② Find all reusable GHA in GitHub's marketplace
		Unable to control the GHA process	① Use conditional expression ② Use concurrency
Construction (50.00%)	Workflow component (17.86%)	Unable to manage multiple components	① Store multiple workflow files in the '*.github/workflows*' directory ② Use '*strategy*' for parallel build and different variations ③ Use a single job with multiple steps ④ Use '*wildcard*' pattern to configure multiple paths in a step
		Unable to refer to other components	① Define '*outputs*' that need to be passed to subsequent jobs ② Use action composition to create a single action then reference it
		Actions version issues	① Chose the correct version of action ② Use semantic versioning and create a major version for users ③ Move the major version tag (such as v1, v2) ④ Set a dedicate '*tags*' branch in the '*on*' attribute
	Configuration (16.07%)	Unable to get branch name	① Use variable *${{ github.event.pull_request.base.ref }}* ② Use variable *${{ github.ref_name }}*
		YAML syntax error	① Directive should be properly indented ② Escape the '/' with a '\' for the filter pattern to work ③ Run multiple commands using a pipe '|' on the '*run*' attribute ④ Do not use '*if: condition*' in YAML because it is no logic attached
		Unable to set/pass environment variable	① Write the environment variable to the '*GITHUB_ENV*' file ② Use GitHub secrets to store environment variables ③ Add '*INPUT_*' to the name of the input variable
	Dependency (7.14%)	Cannot find/install packages	① Change '*actions/setup*' to a latest version ② Manually add the package source in a step ③ Add a personal access token with the '*read:packages*' scope ④ Reference the package's official documentation
	Runner (3.57%)	Unable to configure/share runners	① Make repositories in one organization to reuse a set of runners ② Use labels to organize self-hosted runners based on their characteristics
	Trigger (5.36%)	Cannot trigger GHA manually or scheduled	① Use '*workflow_dispatch*' and set workflow in the default branch ② Scheduled workflows must run on the latest commit on the default branch ③ Set the shortest interval to run scheduled workflows as 5 minutes
Maintenance (23.21%)	Monitoring (3.57%)	Unable to check workflow result status	① Use workflow run API '*[owner]/[repo]/actions/workflows/[workflowID]/runs*' ② Use the '*outcome*' property of each step
	Efficiency (3.57%)	Unable to cache steps or artifacts	① Use '*actions/cache*' and '*actions/setup-node@v2.*' actions ② Install package management tools before the cache step
	Security (12.50%)	Cannot access/pass the secrets	① Reference the GitHub secrets setting guidance ② Reference the environment secret variables in the job ③ Use the '*gh-secrets-action*'
		GitHub token or PAT error	① Set GITHUB_TOKEN to expire after the job is finished / Maxi. 24 hours ② Use GITHUB_TOKEN only on default branch ③ Use custom PAT (Personal Access Token) when running on PR branches ④ Use '*${{ secrets.pat }}*' to manage PAT secret
	Resource management (3.57%)	Unable to share artifacts	① Use the '*upload-artifact*' and '*download-artifact*' to share data between jobs ② Use artifact API '*/actions/artifacts/artifact_id/archive_format*'
Operation (19.64%)	Containerized services (8.93%)	Unable to run commands inside container	① Specify '*container*' instruction for a job ② Create *Dockerfile* or *docker-compose.yaml* then reference it in workflow
		Cannot create Docker image	① Tag/load the image with the commit sha ② Give docker build the correct path to the Dockerfile ③ Use '*setup-buildx-action*' step
	Integration and deployment (10.71%)	Unable to deploy to cloud provider	① Upgrade cloud provider account from a base free tier to a pay tiered ② Reference the cloud provider's official documentation ③ Deploy code with *rsync* over *ssh* (e.g., use *ssh deploy* action)
		Continuous deployment issue	① Keep the '*app*' within the root project ② Add the service principal with role contributor to the resource group ③ Add files to the repository or make them available on build agent

I have multiple GitHub Actions workflow files?"), developers can store multiple workflow files in the '*.github/workflows*' directory. Then, all files will be read and run as independent workflows. Also, developers can use strategy [26] for parallel build and different variations. When multiple paths need to be configured in a single step, developers can use wildcard pattern [21] to describe the paths. For the problem of being unable to reference other components (e.g., Q137: "*How can I reference other actions from my GitHub Action's action.yml file?*"), developers can use action composition [19] to create a single action then reference it. We also distill 4 solution strategies for developers to better specify or modify their actions version, e.g., using semantic versioning and creating a major version for users.

When developers are unable to get a branch name, can't set/pass an environment variable, or have a YAML syntax error, we list 9 common GHA configuration rules that developers can refer to, e.g., the directive should be properly indented. Developers are sometimes unable to find/install packages (e.g., Q893: "*Github actions go test cannot find package error. How can I fix this?*"). We summarize 4 solution strategies for them, e.g., they should change 'actions/setup' to a latest version. In the *Runner* problems, we find that developers

can make repositories in one organization or use labels to configure/share self-hosted runners. Further, we find that developers can use 'workflow_dispatch' [22] to trigger GHA manually. But they should notice that the shortest interval to run scheduled workflows is 5 minutes, as answered in Q6256 [48].

5.3 Solutions for Maintenance

We identify 13 solution strategies for the *Maintenance* problems. Specifically, developers can use the workflow run API[9] or the 'outcome' property of each step to resolve the problem of not being able to check workflow result status (e.g., Q1810: "*How can i get the Passing/Failing status of a Github Action Workflow?*"). When developers have problems with caching or sharing artifacts, they can use the assisted actions, e.g., 'actions/cache', 'actions/setup-node@v2.', 'upload-artifact', and 'download-artifact'. We also summarize 7 solution strategies from developers' answers to *Security* problems. For example, developers should use GITHUB_TOKEN only on the default branch and use PAT (Personal Access Token) when running PR branches, which will help them fix GitHub token or PAT errors. Moreover, the accepted answer of Q6018 [49] suggests that developers should refer to the GHA official security guides [18] and notice that the GITHUB_TOKEN expires when a job finishes or after a maximum of 24 hours.

5.4 Solutions for Operation

We identify 11 solution strategies for the *Operation* problems. For the problems of containerized service (e.g., Q180: "*GitHub Action: How do I run commands inside a docker container*", developers can specify the 'container' instruction for a job [24]. Also, they can create Dockerfile or docker-compose.yaml then reference it in the workflow. Sometimes developers encounter problems with not being able to deploy to cloud providers (for example, Q816: "*Deploy code directly to AWS EC2 instance using GitHub Actions*"). Our solution strategy suggests that developers can deploy code to any server that accepts *rsync* commands over *ssh* by using use the *ssh deploy* action. We also distill 3 solution strategies for resolving continuous deployment errors.

Finding 8. *We distill 56 common solution strategies for 20 problems, covering 12 out of 16 sub-categories in the GHA problem taxonomy. Each sub-category has an average of more than 4 solution strategies (ranging from 2 to 10), indicating that a non-negligible portion of problems are resolved case by case.*

6 DISCUSSION

Here we discuss the implications and the threats to validity.

6.1 Implications

The results of our empirical study can not only help GHA developers and their practice but also the researchers and service/tool providers to better decide where to focus their efforts.

6.1.1 For developers: Based on our findings, there are several suggestions for developers to optimize their GHA practices. Our study finds that GHA problem categories' difficulty and popularity are negatively correlated (RQ2). Developers may have a trade-off between popularity and difficulty when choosing to focus on GHA problems. Thus, developers can decide to focus on problems that match their current skill levels. E.g., novice developers with an interest in GHA may benefit from learning about problems in the more popular and less difficult category. In contrast, experienced developers may want to become more knowledgeable by tackling problems that belong to the less popular and more difficult category. Moreover, the manager of a development team can better prioritize their work by avoiding giving a very difficult task to a junior developer, and assigning more resources (development time) to tasks that belong to these difficult categories (e.g., *Operation* category and *Migration* sub-category).

Also, the problem taxonomy (RQ1) and solution strategies (RQ3) obtained from our study can serve as common guidance for both manual and automated GHA problem resolution. Developers can analyze which questions or issues are specific for which problems and what solution strategies are available. Hence, this enables them to address these problems more appropriately.

6.1.2 For researchers: Our study provides a rich resource of initial ideas for further study. The findings in RQ1 indicate that GHA practice is challenging due to the diversity in problem categories (RQ1). Also, our study shows that only 37.56% of the GHA problems in our dataset have been resolved (RQ2). Thus, researchers can further investigate guidelines and techniques for resolving GHA problems (especially the unresolved ones), covering as broad spectrum of the categories as possible. We recognize that there is no silver bullet to solve arbitrary types of GHA problems. Researchers may extend our solution strategies by mining more cases from online forums or GHA-related projects.

Our results confirm that GHA has many maintenance problems and gains a large attack surface (RQ1). Researchers may develop effective solutions to trace and monitor the execution process and related risks. E.g., researchers can investigate the historical build logs and authentication exceptions to distill the factors that affect GHA efficiency and security. Also, we find that it is difficult for developers to resolve the *Operation* problems (RQ2). Therefore, how to manage and help developers detect GHA workflow's build and release errors and fix them quickly, needs to be addressed. Based on our findings, researchers can further classify the different error types of GHA workflow operation and analyze the root causes.

Previous work [44] found that practical problems with other CI tools (e.g., Travis-CI) arise mainly from test failures, compilation errors, and VCS (git) interactions, all focusing on the build and operation phases. Similarly, we find developers encounter integration and deployment problems in the GHA context (see Section 3.4). However, since GHA is integrated inside GitHub, git errors (e.g., worker fails to fetch code) become less common in GHA problems. Besides, we find some problems specific to GHA's construction and maintenance phases, such as workflow component errors, GHA YAML syntax errors, and workflow cache issues. Researchers can further investigate the similarities/differences between GHA and other CI/CD tools to distill more evidence for developers.

6.1.3 For service/tool providers: We find that developers encounter the most problems in understanding and creating GHA workflows (RQ1). Also, *Construction* problems receive more answers and views

[9] https://api.github.com/repos/[owner]/[repo]/actions/workflows/[workflowID]/runs

than others (RQ2). Therefore, service/tool providers should give priority attention to the questions or issues related to the *Construction* category. There is a need for a list of "GHA best practices" for developers with different construction needs. Service providers can summarize more best practices to help newcomers adopt GHA more easily and quickly. Also, tool providers can deliver more supported techniques for better GHA construction, e.g., reusable actions recommendation and configuration quality checking. Moreover, the GHA configuration complexity should be simplified to lower the initiation obstacles for inexperienced developers.

6.2 Threats to Validity

The threats to the validity of the study include two types.

6.2.1 Internal validity. We acknowledge that only using the "github-actions" tag alone is not sufficient to fully identify posts related to GHA (i.e., some posts have no tags or miss the "github-actions" tag). To minimize this threat, we use previous work [30, 31]'s techniques to develop a set of GHA tags that are significantly relevant to GHA practices. Since RQ1 and RQ3 are based on manual labeling, possible subjectivity in this process may lead to bias in the results. To mitigate this effect, we ensure that each data item is labeled by at least two authors and a third arbiter is used to resolve conflicts. To ensure consistency and soundness of classification, we use Cohen's kappa coefficient [13] and find high inter-rater agreement among the annotators. Metrics used to measure the popularity and difficulty of GHA problems could be another threat. To minimize this threat, we use well-known metrics used by previous work to measure popularity [4, 6, 30] and difficulty [6, 7, 30].

6.2.2 External validity. The potential threat is that we only use SO and GitHub as the data source to study how developers talk about GHA. Although our dataset consists of a representative sample of SO posts and GitHub issues, there could be GHA problems that are never discussed on SO or GitHub. Hence our results may not be representative of all real-world GHA problems. We believe that this study can be improved by including more data sources (e.g., other GHA-related forums), looking at source code to understand the problems facing GHA developers, or asking for feedback directly from GHA developers (via survey or interview) to better understand their problems.

7 RELATED WORK

In recent years, several studies have quantitatively analyzed the basic adoption of GHA. Kinsman et al. [35] collected and analyzed data from 3,190 active GitHub repositories and studied how software developers use GHA to automate their workflows. They found that adopting GHA increased the number of rejected pull requests per month and decreased the number of commits per month for merged pull requests. Chen et al. [12] conducted a large-scale analysis of GHA adoption. They analyzed the changes in developers' development practices after adopting GHA by quantifying the impact of GHA on commit frequency, pull requests, and issue resolution efficiency. Decan et al. [16] conducted a study of 70,278 GHA workflows in 29,778 GitHub repositories and found that repositories that use GHA workflows tend to be more active (with more contributors, pull requests, commits, and issues). In addition, they found that a few actions concentrated most of the reuse, most often in the utility or continuous integration categories.

In addition, some studies have also conducted manual analyses of GHA workflow modifications. Valenzuela-Toledo et al. [50] manually examined 222 GHA workflow commits from 10 different open source repositories, revealing 11 types of modifications by classifying and tagging workflow modifications. They proposed that more tools are needed to support refactoring, debugging, and code editing of GHA workflows. Further, Wessel et al. [52] conducted a statistical analysis of a sample of 662 open source projects hosted on GitHub to understand how projects use GHA. They found that a large number of projects adopted GHA (nearly 30%) and that most threads were related to developers seeking help. Moreover, there have been studies that analyze the security of GHA. Koishybayev et al. [36] conducted a thorough analysis of the GHA workflows from 214K GitHub repositories. They found that most workflows were granted excessive privileges like read-write access to the repository. Furthermore, Benedetti et al. [8] proposed a security assessment methodology to analyze the effects of security issues on GHA workflows and the software supply chain.

Moreover, prior research has investigated the utilization of GHA for specific project types. For instance, Calefato et al. [9] conducted an exploratory analysis of MLOps practices within ML-enabled systems hosted on GitHub, with a specific focus on GHA. The results unveiled that the application of MLOps workflows in open-source GitHub projects is currently constrained. Pachev et al. [42] proposed a new framework that enabled the execution of GHA workflows on supercomputing resources. However, these studies lack an understanding of the challenges and difficulties developers face in practice. Therefore, we conduct an empirical study on SO posts and GitHub issues to seek data-driven evidence from the online software development community.

8 CONCLUSION

This paper provides a large-scale empirical study of 6,590 SO questions and 315 GitHub issues. Using manual labeling, we develop the first taxonomy of GHA problems, including 4 categories and 16 subcategories. We find that developers face diverse challenges when adopting GHA. Next, we investigate the popularity and difficulty of GHA problem categories. We find that GHA has accumulated a lot of unresolved problems; and categories' difficulty and popularity are negatively correlated. Furthermore, we distill 56 common solution strategies for 20 GHA problems. Finally, we distill many implications for different stakeholders.

Our study opens the door for GHA researchers and practitioners to further understand the GHA challenges. In the future, we plan to study the discussions of developers in other forums to draw more accurate and generalizable conclusions.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their insightful comments on the earlier version of this paper. This work was supported partially by the Program of A New Generation of Artificial Intelligence 2030 (Grant No.2021ZD0112904), National Natural Science Foundation of China (Grant No.62202480), Program of Young Innovative Scientist in Hunan Province (Grant No.2023RC3026), and Foundation of PDL (Grant No.2021-KJWPDL-06).

REFERENCES

[1] 2023. Jenkins. https://jenkins.io/ Accessed 2023-3-29.

[2] 2023. Travis CI. https://travis-ci.org/ Accessed 2023-3-29.

[3] Ahmad Abdellatif, Diego Costa, Khaled Badran, Rabe Abdalkareem, and Emad Shihab. 2020. Challenges in chatbot development: A study of stack overflow posts. In *Proceedings of the 17th International Conference on Mining Software Repositories (MSR)*. 174–185.

[4] M. Alshangiti, H. Sapkota, P.K. Murukannaiah, X. Liu, and Q. Yu. 2019. Why is Developing Machine Learning Applications Challenging? A Study on Stack Overflow Posts. In *Proceedings of the International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 1–11.

[5] A.Mahmoud, C.Diego Elias, and EShihab. 2021. Empirical analysis of security vulnerabilities in python packages. In *Proceedings of the International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 446–457.

[6] M. Bagherzadeh and R. Khatchadourian. 2019. Going Big: a Large-Scale Study on What Big Data Developers Ask. In *Proceedings of the European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. 432–442.

[7] K. Bajaj, K. Pattabiraman, and A. Mesbah. 2014. Mining questions asked by web developers. In *Proceedings of the International Conference on Mining Software Repositories (MSR)*. 112–121.

[8] G. Benedetti, L. Verderame, and A. Merlo. 2022. Automatic Security Assessment of GitHub Actions Workflows. In *ACM Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses (SCORED)*. 37–45.

[9] F. Calefato, F. Lanubile, and L. Quaranta. 2022. A preliminary investigation of MLOps practices in GitHub. In *Proceedings of the International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 283–288.

[10] N. Cassee, B. Vasilescu, and A. Serebrenik. 2020. The Silent Helper: the Impact of Continuous Integration on Code Reviews. In *Proceedings of the International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 423–434.

[11] C. Chandrasekara and P. Herath. 2021. Introduction to GitHub Actions. In *Hands-on GitHub Actions*. 1–8.

[12] Tingting Chen, Yang Zhang, Shu Chen, Tao Wang, and Yiwen Wu. 2021. Let's Supercharge the Workflows: An Empirical Study of GitHub Actions. In *Proceedings of the IEEE 21st International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. IEEE, 1–10.

[13] J. Cohen. 1960. A Coefficient of Agreement for Nominal Scales. *Educational Psychological Measurement* 20, 1 (1960), 37–46.

[14] A. Decan, T. Mens, and E. Constantinou. 2018. On the impact of security vulnerabilities in the npm package dependency network. In *Proceedings of the International Conference on Mining Software Repositories (MSR)*. 181–191.

[15] Alexandre Decan, Tom Mens, and Hassan Onsori Delicheh. 2023. On the outdatedness of workflows in the GitHub Actions ecosystem. *Journal of Systems and Software* (2023), 111827.

[16] A. Decan, T. Mens, P.R. Mazrae, and M. Golzadeh. 2022. On the Use of GitHub Actions in Software Development Repositories. In *Proceedings of the International Conference on Software Maintenance and Evolution (ICSME)*. 235–245.

[17] K. Gallaba. 2019. Improving the Robustness and Efficiency of Continuous Integration and Deployment. In *Proceedings of the International Conference on Software Maintenance and Evolution (ICSME)*. 619–623.

[18] GitHub. 2023. Automatic token authentication. https://docs.github.com/en/actions/security-guides/automatic-token-authentication, Accessed 2023-3-29.

[19] GitHub. 2023. Creating a composite action. https://docs.github.com/en/actions/creating-actions/creating-a-composite-action, Accessed 2023-3-29.

[20] GitHub. 2023. Creating starter workflows for your organization. https://docs.github.com/actions/using-workflows/creating-starter-workflows-for-your-organization, Accessed 2023-3-29.

[21] GitHub. 2023. Filter pattern cheat sheet. https://docs.github.com/en/actions/using-workflows/workflow-syntax-for-github-actions#filter-pattern-cheat-sheet, Accessed 2023-3-29.

[22] GitHub. 2023. Manually running a workflow. https://docs.github.com/en/actions/using-workflows/manually-running-a-workflow, Accessed 2023-3-29.

[23] GitHub. 2023. Reusing workflows. https://docs.github.com/en/actions/using-workflows /reusing-workflowsreusable-workflows-and-workflow-templates, Accessed 2023-3-29.

[24] GitHub. 2023. Running jobs in a container. https://docs.github.com/en/actions/using-jobs/running-jobs-in-a-container, Accessed 2023-3-29.

[25] GitHub. 2023. Understanding GitHub Actions. https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions, Accessed 2023-3-29.

[26] GitHub. 2023. Use jobs-job_id-strategy. https://docs.github.com/en/actions/using-workflows/workflow-syntax-for-github-actions#jobsjob_idstrategy, Accessed 2023-3-29.

[27] GitHub. 2023. Using a matrix for your jobs. https://docs.github.com/en/actions/using-jobs/using-a-matrix-for-your-jobs, Accessed 2023-3-29.

[28] GitHub. 2023. Using concurrency. https://docs.github.com/en/actions/using-jobs/using-concurrency, Accessed 2023-3-29.

[29] GitHub. 2023. Using conditions to control job execution. https://docs.github.com/en/actions/using-jobs/using-conditions-to-control-job-execution, Accessed 2023-3-29.

[30] M.U. Haque, L.H. Iwaya, and M. Babar. 2020. Challenges in Docker Development: A Large-Scale Study using Stack Overflow. In *Proceedings of the International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 1–11.

[31] F. Hassan and X. Wang. 2018. Hirebuild: An Automatic Approach to History-Driven Repair of Build Scripts. In *Proceedings of the International Conference on Software Engineering (ICSE)*. 1078–1089.

[32] S. Islam, R.G. Kula, C. Treude, B.Chinthanet, T. Ishio, and K. Matsumoto. 2021. Contrasting third-party package management user experience. In *Proceedings of the International Conference on Software Maintenance and Evolution (ICSME)*. 664–668.

[33] M. A. Jie, S. Q. Tang, W. Liu, and L. X. Xia. 2002. Using Triggers to Design Workflow. *Computer Engineering & Science* (2002).

[34] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M German, and Daniela Damian. 2016. An in-depth study of the promises and perils of mining GitHub. *Empirical Software Engineering* 21 (2016), 2035–2071.

[35] T. Kinsman, M. Wessel, M.A. Gerosa, and C. Treude. 2021. How Do Software Developers Use GitHub Actions to Automate Their Workflows?. In *Proceedings of the International Conference on Mining Software Repositories (MSR)*. 420–431.

[36] I. Koishybayev, A. Nahapetyan, R. Zachariah, S. Muralee, B. Reaves, A. Kapravelos, and A. Machiry. 2022. Characterizing the Security of Github CI Workflows. In *Usenix Security Symposium (USENIX Security)*. 2747–2763.

[37] Bonan Kou, Yifeng Di, Muhao Chen, and Tianyi Zhang. 2022. SOSum: a dataset of stack overflow post summaries. In *Proceedings of the 19th International Conference on Mining Software Repositories (MSR)*. 247–251.

[38] Lena Mamykina, Bella Manoim, Manas Mittal, George Hripcsak, and Björn Hartmann. 2011. Design lessons from the fastest q&a site in the west. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. 2857–2866.

[39] S. McIntosh, B. Adams, and A. Hassan. 2012. The evolution of Java build systems. *Empirical Software Engineering* 17 (2012), 578–608.

[40] M.Golzadeh, A. Decan, and T.Mens. 2022. On the rise and fall of CI services in GitHub. In *Proceedings of the International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 662–672.

[41] A.F. Nogueira and M. Zenha-Rela. 2021. Monitoring a CI/CD Workflow Using Process Mining. *SN Computer Science* 2 (2021), 1–16.

[42] B. Pachev, G. Stuart, and C. Dawson. 2022. Continuous Integration for HPC with Github Actions and Tapis. In *International Practice and Experience in Advanced Research Computing (PEARC)*. 1–4.

[43] Mohammad Masudur Rahman and Chanchal K Roy. 2015. An insight into the unresolved questions at stack overflow. In *Proceedings of the IEEE/ACM 12th Working Conference on Mining Software Repositories (MSR)*. IEEE, 426–429.

[44] Thomas Rausch, Waldemar Hummer, Philipp Leitner, and Stefan Schulte. 2017. An empirical analysis of build failures in the continuous integration workflows of java-based open-source software. In *Proceedings of the IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 345–355.

[45] Sk Golam Saroar and Maleknaz Nayebi. 2023. Developers' Perception of GitHub Actions: A Survey Analysis. In *Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering*. 121–130.

[46] C.B. Seaman. 1999. Qualitative Methods in Empirical Studies of Software Engineering. *IEEE Transactions on Software Engineering* 25, 4 (1999), 557–572.

[47] M. Shahin, M.A. Babar, and L. Zhu. 2017. Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices. *IEEE access* 5 (2017), 3909–3943.

[48] Stackoverflow. 2023. Github action on scheduled time not running at all? https://stackoverflow .com/questions/73930702/github-action-on-scheduled-time-not-running-at-all, Accessed 2023-3-29.

[49] Stackoverflow. 2023. secrets.GITHUB_TOKEN could be expired in github action? https://stackoverflow.com/questions/73643738/secrets-github-token-could-be-expired-in-github-action, Accessed 2023-3-29.

[50] P. Valenzuela-Toledo and A. Bergel. 2022. Evolution of GitHub Action Workflows. In *Proceedings of the International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 123–127.

[51] S.W. Wang, T. Chen, and A.E. Hassan. 2018. Understanding the factors for fast answers in technical Q&A websites: An empirical study of four stack exchange websites. *Empirical Software Engineering* 23 (2018), 1552–1593.

[52] M. Wessel, J. Vargovich, M.A. Gerosa, and C. Treude. 2022. GitHub Actions: The Impact on the Pull Request Process. *arXiv preprint arXiv:2206.14118* (2022).

[53] Y.W. Wu, Y. Zhang, T. Wang, and H.M. Wang. 2020. An Empirical Study of Build Failures in the Docker Context. In *Proceedings of the International Conference on Mining Software Repositories (MSR)*. 76–80.

[54] Yiwen Wu, Yang Zhang, Kele Xu, Tao Wang, and Huaimin Wang. 2022. Understanding and Predicting Docker Build Duration: An Empirical Study of Containerized Workflow of OSS Projects. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 1–13.

[55] J.H. Zar. 2005. Spearman rank correlation. *Encyclopedia of Biostatistics* 7 (2005).

[56] Yang Zhang, Bogdan Vasilescu, Huaimin Wang, and Vladimir Filkov. 2018. One size does not fit all: an empirical study of containerized continuous deployment workflows. In *Proceedings of the 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. 295–306.

[57] Y. Zhao, A. Serebrenik, Y. Zhou, V. Filkov, and B. Vasilescu. 2017. The Impact of Continuous Integration on Other Software Development Practices: a Large-Scale Empirical Study. In *Proceedings of the International Conference on Automated Software Engineering (ASE)*. 60–71.

[58] Z.P.Chen, H.H. Yao, Y.L. Lou, Y.B. Cao, Y.Q .Liu, H.Y. Wang, and X.Z. Liu. 2021. An Empirical Study on Deployment Faults of Deep Learning Based Mobile Applications. In *Proceedings of the International Conference on Software Engineering (ICSE)*. 674–685.