# GHARuns: A Large Dataset of GitHub Actions Workflow Execution Metadata

Aref Talebzadeh Bardsiri
*Software Engineering Lab*
*University of Mons*
Mons, Belgium
aref.talebzadehbardsiri@umons.ac.be

Tom Mens
*Software Engineering Lab*
*University of Mons*
Mons, Belgium
tom.mens@umons.ac.be

Alexandre Decan
F.R.S.-FNRS Research Associate
*Software Engineering Lab*
*University of Mons*
Mons, Belgium
alexandre.decan@umons.ac.be

***Abstract*—GitHub Actions is the integrated CI/CD workflow automation service of GitHub, the most popular collaborative software development platform. While the evolution of workflow configuration files has been actively studied, there are considerably less large-scale empirical studies involving execution and runtime behaviour of those workflows. This is due to the data retention policy imposed by GitHub and the resulting lack of longitudinal data involving large numbers of repositories. We propose GHARuns, an evolving dataset and associated extraction tool to enable such large-scale studies. Version 2026-07-08 of the dataset contains the metadata of 71.7M+ workflow runs for 197K+ GitHub Actions workflows across 45K+ active public GitHub repositories over a period of 146 days, well exceeding GitHub's retention period. It also contains the metadata for for 122.7M+ job runs and 1362.3M+ step runs for a subset of 29.4M+ workflow runs.**

***Index Terms*—CI/CD automation, GitHub Actions, execution traces, workflow runs**

## I. Introduction

GitHub is the largest platform for hosting and collaborating on software projects, and GitHub Actions (GHA) is the dominant CI/CD workflow configuration service [1]. GHA workflows are seamlessly integrated within the GitHub platform, allowing repository maintainers to automate a wide range of tasks related to software development and project collaboration. Despite the demonstrated value of CI/CD, and of GHA workflow configurations in particular, prior research reported that developers face challenges when using workflow configuration files [2], [3]. Maintaining them is a complex endeavour [4] and developers often struggle with writing, testing, and debugging them [2]. They frequently encounter workflow execution failures [5], reducing the reliability of the CI/CD pipeline and increasing the maintenance effort.

Many studies focused on understanding and improving the structure and maintenance of workflow configurations, by studying the evolution of workflow file contents [6]–[8]. Studies involving the execution of workflows are equally important to address the challenges maintainers face when GHA workflows fail. Such studies are less prominent, partly because they require considerably more data that is challenging to obtain at a large scale.

Bouzenia et al. [9] and Khatami et al. [10] relied on the workflow execution metadata of only 952 GitHub repositories to study resource usage and workflow failure patterns. Moriconi et al. [11] provided a 142GB dataset with the raw execution logs in 2023 of GHA workflow runs across 25K repositories, but it was limited to only five runs per workflow and was never updated. This motivates the need for a structured dataset of GHA execution metadata covering a large number of GitHub repositories and their constituent workflows.

This paper provides three main contributions:
(1) a tool to continuously collect GHA workflow execution metadata at scale using GitHub's REST and GraphQL API, taking into account GitHub's rate limits and retention periods;
(2) a large dataset of GHA workflow run metadata containing (in its version 2026-07-08) the historical metadata of 71.7M+ workflow runs from 197K+ workflows across 45K+ active GitHub repositories over a period of 146 days [12];
(3) fine-grained metadata for 122.7M+ job runs and 1362.3M+ step runs for a large subset of 29.4M+ workflow runs.

## II. Related Work

While several studies have analyzed the static configurations of workflow files, such as their evolution [8], [13], the presence of code clones [14] and security vulnerabilities [15], [16], there is a notable lack of large-scale research focusing on the dynamic execution of these workflows. Bouzenia and Pradel [9] studied resource usage of GHA runs on a dataset of 952 GitHub repositories. Based on this dataset, Khatami et al. [10] analyzed workflow failure patterns to understand how developers respond to these failure patterns and how this relates to project characteristics. Due to the limited data, they could only formulate hypotheses for future work rather than providing definitive conclusions, and explicitly called for further large-scale validation to confirm their findings.

Moriconi et al. [17] proposed GHALogs, a larger and more comprehensive dataset comprising 513K complete workflow execution logs and metadata of 116K GHA workflow files across 25K GitHub repositories spanning 20 programming languages. However, log extraction was limited to only five runs per workflow in the last 90 days, relative to the extraction period of early October 2023. This makes the dataset outdated and insufficient for longitudinal analysis of execution patterns.

In addition, the above datasets [9], [17] were not updated recently. Today they can no longer be completed without data

loss due to GitHub's strict retention and rate limits, these restrictions will be detailed in the next section.

Another relevant dataset, proposed by Cardoen et al. [7], [18], contains 4.1M+ workflow histories (snapshots) related to 308K+ GHA workflow files from 52K+ GitHub repositories. Earlier versions of this dataset have been used to study the evolution of workflow files [8] and the prevalence of code clones in workflows [14]. GHARuns, the dataset presented in the current paper, can be considered as its "runtime" companion, as it targets the same repositories. enabling future analyses that combine investigations of the workflow file contents and their runtime characteristics.

## III. Data Collection

To collect GHA workflow run metadata, one must first select the repositories to monitor. Instead of querying a random sample of GitHub repositories, we monitored the 52,956 repositories contained in the aforementioned GHA workflow history dataset [18]. It is based on GitHub repositories with at least 100 stars, at least 300 commits and at least one commit after March 26th, 2025.

Our data collection tool, however, is generic and can operate on any given list of repositories. For each monitored repository, it aims to collect metadata for all workflow runs executed during the observation period. This includes coarse-grained metadata at the level of workflow runs, and more fine-grained metadata at the level of job runs and step runs.

Collecting such workflow run information at scale is challenging for two reasons. First, the GitHub REST API imposes a strict rate limit of up to 5,000 authenticated requests per hour. A naive collection strategy relying only on REST requests would require a huge number of calls to retrieve all three levels (run-level, job-level, and step-level) metadata for all monitored workflows. Combining the REST API with GitHub's GraphQL API would reduce the number of API calls needed to retrieve fine-grained execution metadata, while the separate rate limits of both APIs allow increasing effective collection throughput. A second challenge is that GitHub does not retain detailed execution data indefinitely. GitHub's official documentation reports that artifacts and build logs for public repositories are retained for 90 days by default before being permanently deleted.[1] As a result, historical execution data is gradually lost once the retention period expires.

These limitations motivated us to design a continuously running collection pipeline to retrieve metadata for a large number of workflow runs fast enough to keep pace with newly generated workflow runs, while archiving them before they get deleted. We implemented an extraction pipeline in Python combining the REST and GraphQL APIs. To support large-scale extraction, the tool features a built-in sharding mechanism to run multiple distributed worker instances in parallel, where distinct partitions of the repository list can be assigned to each worker. This strategy makes continuous collection of GHA execution metadata practically feasible.

[1]https://docs.github.com/en/actions/how-tos/manage-workflow-runs/remove-workflow-artifacts [last access on 2026-07-09]

TABLE I
Main metadata collected for workflow, job and step runs

| attribute | description |
|---|---|
| **workflow runs** | |
| id | Unique id of the workflow run |
| repository | The repository in which the workflow was run |
| path | Path to the workflow file in the repository |
| name | Name of the workflow being run |
| event | The event type that triggered the run |
| run_attempt | Number of attempts made for the run |
| head_sha | Hash of the workflow commit |
| head_branch | Branch of the commit |
| status | Status of the run |
| conclusion | Conclusion of the run |
| run_started_at | Start time of the workflow run |
| updated_at | Last updated time of the workflow run |
| **job runs** in workflow runs | |
| id | Unique id of the workflow job being run |
| name | Name of the job being run |
| status | Status of the job run |
| conclusion | Conclusion of the job run |
| started_at | Start time of the job run |
| completed_at | End time of the job run |
| **step runs** in job runs | |
| step_number | Relative position of the step in the job being run |
| name | Name of the step being run |
| status | Status of the step run |
| conclusion | Conclusion of the step run |
| started_at | Start time of the step run |
| completed_at | End time of the step run |

In a first phase, the extraction tool queries the REST endpoint /repos/{owner}/{repo}/actions/runs to retrieve workflow-run metadata. In a second phase, it uses the GraphQL API to retrieve more fine-grained metadata of job runs (represented as CheckRun nodes) and step runs within each job. However, GraphQL query complexity limits[2] impose a fixed upper bound on the number of job runs and step runs per workflow run. This causes runs with complex execution structures to return truncated results, or even cause the query to fail entirely. To ensure completeness of the collected metadata, in a third phase the extraction tool retrieves the complete job-run and step-run metadata of such runs individually, by querying the paginated REST API endpoint /repos/{owner}/{repo}/actions/runs/{run_id}/jobs.

## IV. Dataset description

Using this process we collected data from 2026-02-01 to 2026-06-27, covering a period of 146 days at the publication date of this paper, corresponding to version 2026-07-08 of the GHARuns dataset [12]. We plan to extend this observation period in future versions.

GHARuns includes 71,663,725 workflow runs from 197,803 workflows across 45,989 active GitHub repositories (i.e., repositories that had at least one workflow run during the observation period). It also includes metadata for 122,736,464 job runs and 1,362,348,471 step runs of 29,383,424 of workflow runs (i.e., 41% of all workflow runs).

[2]https://docs.github.com/en/graphql/overview/rate-limits-and-query-limits-for-the-graphql-api [last access on 2026-07-09]

TABLE II
DESCRIPTIVE CHARACTERISTICS OF GHARUNS (VERSION 2026-07-08).

| characteristic | value |
|---|---|
| # monitored repositories | 52,956 |
| # active repositories* | 45,989 |
| # unique workflows | 197,803 |
| # unique workflow runs | 71,663,725 |
| Average runs per day | 490,847 |
| # runs with job and step details | 29,383,424 |
| Total number of job runs | 122,736,464 |
| Total number of step runs | 1,362,348,471 |
| Average and median jobs per run | Avg: 4.20 Med: 1 |
| Average and median steps per job | Avg: 11.10 Med: 9 |
| Observation period (days) | 146 (2026-02-01→ 2026-06-27) |

* Active repositories are those that had at least one workflow run during the observation period.

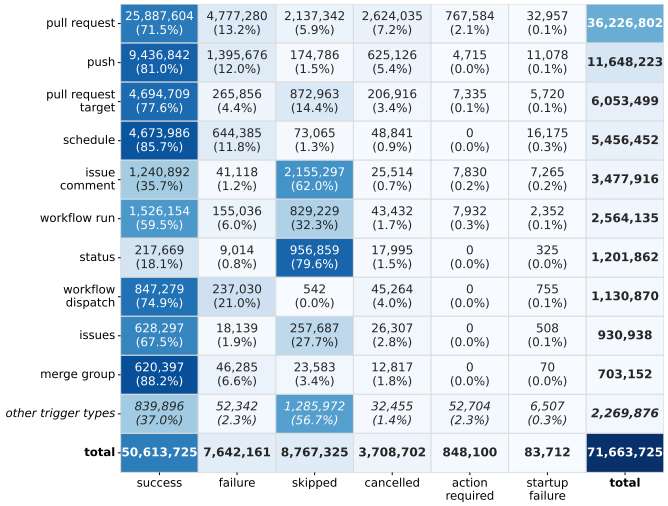


Fig. 1. Per-event conclusion breakdown for the ten most frequent workflow triggers. A cell's color intensity reflects to the percentage of runs with the given conclusion among all runs of that trigger type.

Details about the job and step runs triggered by pull requests (events pull_request and pull_request_target) were not collected. This was an intentional scoping decision, since pull_request events correspond to the execution of workflows on code that is not (yet) part of the repository, and thus are less directly related to the execution of workflows in "official" branches. By excluding these data, we significantly reduce the amount of API queries needed, enabling us to retrieve the workflow execution metadata from a large number of repositories faster than they are produced. However, we still collect the metadata of the workflow executions triggered by the merge of each pull request. This approach allows us to track pull request outcomes without capturing all intermediate "work in progress" changes made in pull requests.

Table I lists the main attributes collected at each level: workflow runs, jobs executed during workflow runs, and steps executed as part of jobs during workflow runs. For example, at each level we retrieved information about the execution status (e.g., queued, in_progress, completed), the run conclusion (i.e., success, failure, skipped, cancelled, startup failure, and action required), and timing information for the workflow runs (run_started_at and updated_at), as well as for job runs and step runs (started_at and completed_at). Table II summarizes the main characteristics of GHARuns [12], quantifying the number of workflow, job and step runs and their associated metadata.

## V. EXPLORATORY EMPIRICAL ANALYSIS

This section presents exploratory analyses of GHARuns, illustrating the kind of insights that can be obtained from it.

To understand the effect of the trigger event type on the overall conclusion of a workflow run, we analyze the relationship between the metadata fields event and conclusion. Different trigger event types[3] correspond to different workflow usage scenarios, such as routine integration (e.g., push and pull_request), scheduled maintenance (schedule), manual invocations (workflow_dispatch), or conditional automation (e.g., issue_comment, pull_request_review), and may therefore show different execution outcome patterns.

The heatmap of Figure 1 shows the distribution of the 71.7M+ workflow run conclusions for the ten most frequent trigger event types. The remaining 20 less frequent event types are aggregated in the "other trigger types" row (2,269,876 runs, 3.2% of all runs). The vast majority of workflows are triggered by code integration activities (e.g., pull request, push, and pull request target). The most common trigger event is pull_request (50.55% of all runs) where 71.50% of runs succeeded, 13.20% failed, 5.90% were skipped, and 7.20% were cancelled. For most event types, the majority of runs succeeded, whereas event types status and issue_comment show a high percentage of skipped runs (79.60% and 62% respectively). This is likely because these event types offer little to no trigger-level filtering. For instance status events cannot be filtered in the workflow trigger definition at all, and issue_comment events cannot be filtered on the comment contents. Workflows triggered by these event types typically rely on job-level conditions (if) that are evaluated only after the run has been created. Therefore when the condition does not hold, the run is recorded with conclusion skipped.

GHARuns enables researchers to study performance-related and efficiency-related aspects of workflows based on the timing information of runs at different levels of granularity. Figure 2 shows the distribution of the durations of workflow runs, job runs and step runs that conclude with either success or failure. Workflow run duration was computed as the period between the start time (run_started_at) of the workflow run and the completion time (completed_at) of the last finished job that had a conclusion of either success or failure. To compute this figure we restricted the analysis to the subset of 29.4M+ of workflow runs for which detailed job run metadata was available, and we further restricted to only those 22.4M+ workflow runs that had executed at least one job with a

[3]https://docs.github.com/en/actions/reference/workflows-and-actions/events-that-trigger-workflows[last access on 2026-07-09

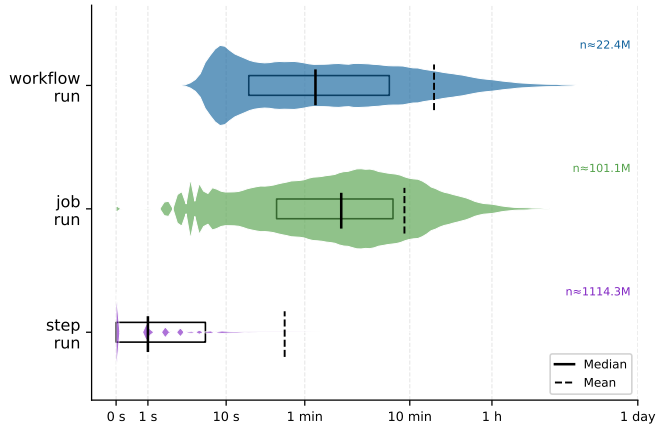


Fig. 2. Distribution (in log scale) of workflow, job and step run duration, all capped at the 99.9th percentile.

TABLE III
FAILURE RATES OF WORKFLOW RUNS ACROSS SIZE BUCKETS BASED ON THE NUMBER OF WORKFLOW JOBS AND STEPS. RUNS THAT DID NOT EXECUTE ANY JOB OR STEP ARE EXCLUDED FROM THE ANALYSIS.

| | bucket | # runs | # failures | % failure rate |
|---|---|---|---|---|
| *by jobs* | 1-2 jobs | 21,503,714 | 1,277,868 | 5.94% |
| | 3-5 jobs | 3,458,808 | 381,048 | 11.02% |
| | 6-10 jobs | 1,827,491 | 253,430 | 13.87% |
| | 11-20 jobs | 1,126,106 | 215,068 | 19.10% |
| | 21+ jobs | 998,727 | 270,911 | 27.13% |
| | total | 28,914,846 | 2,398,325 | 8.29% |
| *by steps* | 1-5 steps | 5,200,710 | 214,057 | 4.12% |
| | 6-10 steps | 6,206,016 | 453,809 | 7.31% |
| | 11-20 steps | 4,244,469 | 510,260 | 12.02% |
| | 21-50 steps | 2,830,123 | 394,773 | 13.95% |
| | 51+ steps | 3,583,754 | 699,816 | 19.53% |
| | total | 22,065,072 | 2,272,715 | 10.30% |

success or failure outcome.

All three distributions in Figure 2 are highly skewed, with a long tail of workflow, job, and step runs taking a long time to complete. The median duration is 77 seconds for workflow runs, 134 seconds for job runs, and 1 second for step runs. The mean duration is considerably higher than the median for all cases due to the skewness fo the distributions.

These results provide initial clues about the performance of GHA workflows. Combined with the actual workflow contents, this could be used to identify potential bottlenecks in workflow execution. For example, it could be used to analyse performance differences between executing *reusable Actions* and *custom shell commands*, or to analyse whether failed runs take longer to execute than successful runs.

We also explored the relationship between workflow size and failure rates by analyzing the distribution of workflow run conclusions across different workflow sizes, using the number of jobs and steps as proxies for workflow size as in [19]. Table III shows the failure rates of workflow runs across different job and step count buckets. We observe an increasing failure rate as workflow size increases. For example, runs of workflows with 1 or 2 jobs have a failure rate of 5.94%. For 3 to 5 jobs, the workflow failure rate increases to 11.02%, and the failure rate continues to increase for buckets of workflows with more jobs. The last bucket of heavy workflow runs with 21+ jobs experience a very high failure rate of 27.13%, nearly quadrupling the rate for workflows with 1 or 2 jobs.

We observe a similar pattern when considering the number of workflow steps. Runs of workflows with 1 to 5 steps have a low failure rate of 4.12%, and the failure rate continues to increase as the number of workflow steps increases. The last bucket of heavy workflow runs with 51+ steps has a high failure rate of 19.53%, nearly quintupling the rate for workflows with 1 to 5 steps.

These results provide preliminary evidence that larger workflows are more prone to failures, possibly due to increased complexity and potential for errors in larger workflows. Future research should investigate the specific factors that contribute to higher failure rates in large workflows, such as the use of more complex building blocks, increased likelihood of configuration errors, or longer execution times leading to timeouts.

## VI. USEFULNESS OF GHARUNS

GHARuns provides execution metadata for a large number of GHA workflows in a large number of GitHub repositories, alongside the continuous extraction tool necessary to maintain it. This enables researchers to analyze the dynamic execution behavior of CI/CD pipelines at scale and over time.

Among others, GHARuns can be utilized to address research questions regarding workflow reliability and performance. For instance, one could investigate workflow failure rate, how this rate changes over time, identify the common root causes of failures at the job and step levels and detect error-prone shell commands in steps. Furthermore, the temporal dimension of the metadata allows researchers to analyse the time it takes to fix failing workflows, and identify project characteristics (e.g., programming language, project size, activity level) that influence time-to-recovery. The dataset can also be used to find opportunities to improve workflow execution speed, by identifying bottlenecks during the execution of jobs and steps.

In addition to carrying out the aforementioned performance-related analyses, researchers can combine GHARuns with a complementary dataset of workflow file histories [18] to analyze the relationship between the configuration of GHA workflows and their execution. This could be used, for instance, to analyse whether certain workflow configuration characteristics are more likely to lead to workflow failures and flakiness [20], and whether the reliability of workflows is related to their size, complexity or other maintainability aspects [19].

One could also take into account the evolution of workflow file changes over time by relying on the GAWD workflow differencing tool [21], in order to study which kind of fine-grained workflow changes relate to workflow failure patterns. This could provide insights in the types of changes that are

more likely to lead to workflow failures, or the types of changes that require more time to fix failures. Similarly, the dataset could be used to reduce the resource waste due to rerunning failed workflows [22].

## VII. Limitations

Due to GitHub's API rate limits it was not practically feasible to extract fine-grained metadata at job and step level for *all* workflow runs. We have therefore restricted the fine-grained extraction to workflow runs that were not triggered by pull requests. As a consequence, GHARuns is less suitable for researchers seeking to analyse pull-request-related workflow execution data specifically.

Although our workflow run extraction tool includes several retry mechanisms to fetch complete job and step metadata, for 883 workflow runs (0.001% of all runs) we were only able to gather their workflow level metadata. We could not retrieve job or step level run metadata for them. More precisely, for 729 of them, all GraphQL and REST queries consistently failed; for the remaining 154 workflow runs only a partial set of job runs was retrieved, all within workflows belonging to 46 repositories with unusually large CI matrix strategies (e.g., *huggingface/transformers*, *NVIDIA/cccl*). These partial records were removed from GHARuns to avoid bias in job-count distributions.

## VIII. Conclusion

Obtaining and maintaining execution data of CI/CD workflows from GitHub at a large scale over extended periods of time is technically challenging. To address this challenge, we created GHARuns, a large dataset of GitHub Actions workflow run metadata in 45K+ active GitHub repositories.

Version 2026-07-08 of GHARuns contains execution metadata over a period of 146 days from 2026-02-01 to 2026-06-27, obtained using the GitHub REST API for 71.7M+ workflow runs of 197K+ workflows. For 29.4M+ workflow runs (excluding all those triggered by pull requests), GHARuns contains metadata at the fine-grained level of job runs and step runs, obtained using the GraphQL API.

GHARuns can be complemented easily with metadata about the evolution workflow configuration files, as provided in [18]. The dataset enables researchers to analyze the dynamic execution behavior of GitHub workflows at scale and over time, enabling longitudinal empirical studies related to workflow reliability and performance, and how this relates to other aspects such as size, complexity and maintainability.

## IX. Data Availability

The GHARuns dataset is publicly available on Zenodo [12]. The repository hosting the source code and technical extraction details is available at https://github.com/sgl-umons/gharuns-collector.

## X. Acknowledgment

This work is supported by F.R.S.-FNRS under grant numbers J.0147.24 and F.4515.23.

## References

[1] M. Golzadeh, A. Decan, and T. Mens, "On the rise and fall of CI services in GitHub," in *Int'l Conf. Software Analysis, Evolution and Reengineering*. IEEE, 2022.

[2] S. G. Saroar and M. Nayebi, "Developers' perception of GitHub Actions: A survey analysis," in *Int'l Conf. Evaluation and Assessment in Software Engineering*, 2023.

[3] H. Onsori Delicheh, G. Cardoen, A. Decan, and T. Mens, "Automation and reuse practices in GitHub Actions workflows: A practitioner's perspective," *ACM Trans. Softw. Eng. Methodol.*, 2026.

[4] T. Ghaleb, O. Abduljalil, and S. Hassan, "CI/CD configuration practices in open-source Android apps: An empirical study," *ACM Trans. Softw. Eng. Methodol.*, May 2025.

[5] L. Zheng, S. Li, X. Huang, J. Huang, B. Lin, J. Chen, and J. Xuan, "Why do GitHub Actions workflows fail? An empirical study," *ACM Trans. Softw. Eng. Methodol.*, Jul. 2025.

[6] A. Khatami, C. Willekens, and A. Zaidman, "Catching smells in the act: A GitHub Actions workflow investigation," in *Int'l Conf. Source Code Analysis and Manipulation*. IEEE, 2024.

[7] G. Cardoen, T. Mens, and A. Decan, "A dataset of GitHub Actions workflow histories," in *Int'l Conf. Mining Software Repositories*. ACM, 2024.

[8] P. Rostami Mazrae, A. Decan, T. Mens, and M. Wessel, "An empirical study of the evolution of GitHub Actions workflows," *Journal of Systems and Software*, vol. 236, 2026.

[9] I. Bouzenia and M. Pradel, "Resource usage and optimization opportunities in workflows of GitHub Actions," in *Int'l Conf. Software Engineering*. ACM, 2024.

[10] A. Khatami, C. Brandt, and A. Zaidman, "Beyond the YAML file: Understanding real-world GitHub Actions workflow adoption," in *Int'l Conf. Evaluation and Assessment in Software Engineering*. IEEE, 2026.

[11] F. Moriconi, T. Durieux, J.-R. Falleri, R. Troncy, and A. Francillon, "GHALogs: Large-scale dataset of GitHub Actions runs," in *Int'l Conf. Mining Software Repositories*, 2025.

[12] A. Talebzadeh Bardsiri, T. Mens, and A. Decan. (2026) GHARuns: A large dataset of GitHub Actions workflow execution metadata [dataset]. Zenodo [Version 2026-07-08]. [Online]. Available: https://doi.org/10.5281/zenodo.20427454

[13] P. Valenzuela-Toledo and A. Bergel, "Evolution of GitHub Action workflows," in *Int'l Conf. Software Analysis, Evolution and Reengineering*. IEEE, 2022.

[14] G. Cardoen, T. Mens, and A. Decan, "An empirical analysis of code clones in GitHub Actions workflows," in *Int'l Conf. Software Analysis, Evolution and Reengineering*. IEEE, 2026.

[15] S. Muralee, I. Koishybayev, A. Nahapetyan, G. Tystahl, B. Reaves, A. Bianchi, W. Enck, A. Kapravelos, and A. Machiry, "ARGUS: a framework for staged static taint analysis of GitHub workflows and actions," in *USENIX Security Symposium*, 2023.

[16] G. Benedetti, L. Verderame, and A. Merlo, "Automatic security assessment of GitHub Actions workflows," in *Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses*. ACM, 2022.

[17] F. Moriconi, T. Durieux, J.-R. Falleri, R. Troncy, and A. Francillon. (2025) GHALogs: Large-scale dataset of GitHub Actions runs [dataset]. Zenodo (Version 1.0.1). [Online]. Available: https://doi.org/10.5281/zenodo.14796970

[18] G. Cardoen. (2026) A dataset of GitHub Actions workflow histories [dataset]. Zenodo [Version 2026-05-22]. [Online]. Available: https://doi.org/10.5281/zenodo.20340547

[19] A. T. Bardsiri, A. Decan, and T. Mens, "On the GitHub Actions language: Usage, evolution, and workflow reliability," in *Int'l Conf. Software Maintenance and Evolution*. IEEE, 2026.

[20] W. Ge and C. Zhang, "Understanding and detecting flaky builds in GitHub Actions," 2026, arXiv preprint. [Online]. Available: https://doi.org/10.48550/arXiv.2602.02307

[21] P. Rostami Mazrae, A. Decan, and T. Mens, "gawd: A differencing tool for GitHub Actions workflows," in *Int'l Conf. Mining Software Repositories*. ACM, 2024.

[22] J. Huang and B. Lin, "On the reruns of GitHub Actions workflows," *ACM Trans. Softw. Eng. Methodol.*, Feb. 2026.