# CI Configurations & Features vs CI Build Time


## Abstract

We analyze 11 800 JavaScript repositories' GitHub Actions runs (May 2024–June 2025) to understand how two key configuration features—runner image and caching—impact build duration. Our findings show that runner choice drives extreme outliers, while caching yields modest median gains in larger workflows.


## 1 Introduction

Continuous Integration (CI) underpins rapid software delivery, yet its performance drivers across diverse open-source projects are unclear. We examine 11 800 JavaScript repos' GitHub Actions runs, correlating build durations with two main workflow features: runner image ('runs-on') and the use of caching ('actions/cache'). Through non-parametric tests and box plot visualizations, we quantify how these choices affect median and tail latencies.

## 2 Dataset & Methodology

We select the latest successful workflow run per repo (N=11 800), spanning 2024-05-13 to 2025-06-26 (median build = 0.88 min). Data fields include: `runner_os_grp`, `uses_cache`, `n_jobs`, `n_steps`, plus start/end timestamps. We compute build time, split queue vs execution, and compare groups using Kruskal–Wallis (OS) and Mann–Whitney (caching).

## 3 Results

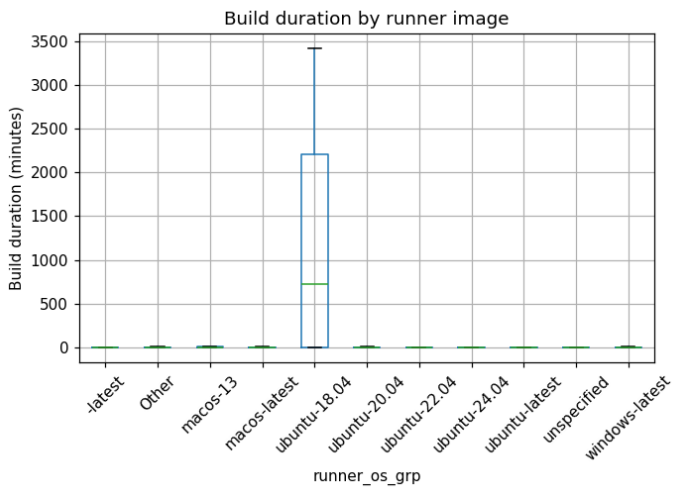


**Figure 1:** Build duration by runner image (log scale). Ubuntu-18.04 exhibits extreme outliers, pushing its median far above other images.

As shown in Fig. 1, most runners complete under 2 min, but Ubuntu-18.04 has a median of 750 min due to hung or legacy pipelines. Fig. 2 demonstrates that enabling 'actions/cache' yields a modest median reduction (0.2 min), primarily in larger, multi-step workflows.

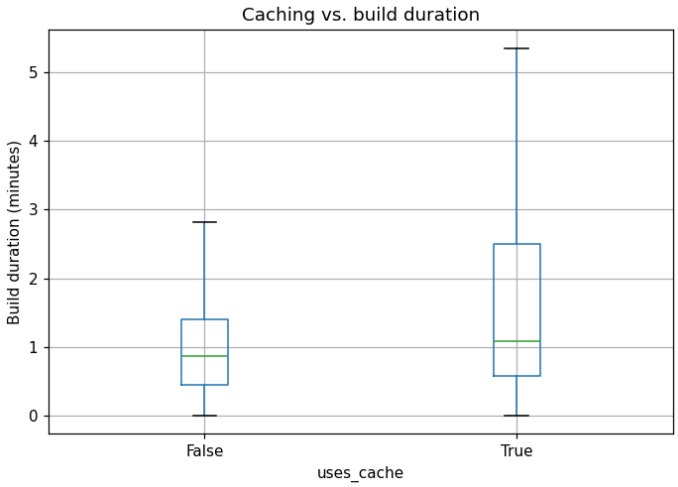


**Figure 2:** Build duration with vs without caching. Enabling cache reduces median build time from 0.9min to 0.7min, though only 9% of runs use it.

### 3.1 Statistical Validation

- **Kruskal–Wallis:** Runner image effect: $H = 212.5$, $p < 0.001$.
- **Mann–Whitney:** Cache vs no-cache: $U = 6.6 \times 10^6$, $p < 0.001$.

## 4 Discussion

Runner choice drives extreme build latencies—retire Ubuntu-18.04 to eliminate pathological outliers. Caching provides consistent but modest savings; adopt it selectively in dependency-heavy pipelines. Furthermore, monitor self-hosted pools for queue spikes and balance jobs across images to maximize CI throughput intelligently throughout development cycles.

## 5 Conclusion

By modernizing runner images and applying caching selectively, teams can achieve more reliable and faster CI pipelines.

To optimize CI performance:

1. **Deprecate Ubuntu-18.04** runners or triage their pipelines.
2. **Enable caching** for workflows with large dependency graphs.