

How Many Lighthouse Runs Do You Need?

A variance, confidence-interval, and power framework for defensible PageSpeed before/after claims

Version: 2026-06-22. Research package status: reproducible manuscript, secondary-evidence data extraction, analytic CI/power tables, raw-data schema, runner scaffold, and required-N calculator. The 200-site x 50-run x 3-mode empirical matrix is specified but not represented as completed data in this package.

Abstract

Lighthouse is widely used as the lab test behind local audits, Lighthouse CI, and PageSpeed Insights diagnostics, yet common before/after claims are usually made from one run or from an unqualified median-of-five convention. That practice is methodologically underpowered because Lighthouse output is a distribution, not a scalar property of a URL. Google documents major sources of score variability and advises multiple runs, and vendor measurements show that run count can reduce noise, but the industry still lacks per-metric confidence intervals and power tables by run count, throttling mode, and site class. This article gives the missing inferential framework and a pre-registered design for a definitive study: 200 public URLs stratified by site class, 50 or more runs per URL, three throttling modes, controlled containerized Chrome/Lighthouse execution, variance decomposition, bootstrap confidence intervals for mean and median-run estimators, and required-N calculators for before/after claims. The central result is not a universal number of runs. The defensible answer is a function of metric variance, estimator, throttling mode, site class, and the smallest effect size worth claiming. Median-of-five is acceptable as a quick screening convention, but it is not a statistically defensible proof of a small performance change. For example, under a normal approximation and independent equal-n before/after design, a score metric with run-level SD = 5 points needs about 16 runs per condition for 80% power to detect a 5-point shift, about 44 runs per condition for a 3-point shift, and only about 4 runs per condition if the true shift is 10 points. The full empirical study proposed here converts these analytic formulas into CI tables by Lighthouse metric, throttling mode, and site class.

1. The problem: the page-speed industry compares point estimates that are actually distributions

A Lighthouse report looks like a determinate measurement of a URL. In reality, a report is one observation from a stochastic measurement process. The same URL can vary because the page is nondeterministic, the network path changes, the origin and third-party servers respond differently, CPU scheduling changes, resources contend inside the browser or host, and the browser engine itself makes nondeterministic choices. The official Lighthouse variability documentation describes these sources and explicitly recommends repeated runs and aggregation rather than a single report (GoogleChrome Lighthouse Team, 2026a). Chrome's own performance-scoring documentation is equally clear in substance: variability in underlying conditions means a performance score should be interpreted as a distribution, not as a single exact number (Chrome for Developers, 2026a).

The practical consequence is severe. A common audit workflow says: run Lighthouse before a change, run it after the change, and compare the two scores. That workflow silently treats measurement error as zero. In low-noise cases this may be harmless; in high-noise cases it can reverse the conclusion. A five-point gain can be real, noise, or a mixture. A five-point loss can be regression, server jitter, cache-state drift, A/B-test drift, a consent-banner variant, a CDN cache miss, CPU co-tenancy, a Chrome trace outlier, or a changed Lighthouse version. Without empirical intervals, the industry cannot distinguish those cases.

This article reframes the question. "How many Lighthouse runs do I need?" is not answered by one fixed integer. It is answered by specifying the estimand, the metric, the run-level variance, the aggregation rule, the throttling mode, the site class, and the smallest effect size that matters. The final object should be a table and calculator, not a slogan.

2. Contributions

This paper contributes six elements.

1. A claim taxonomy that separates monitoring, regression gating, optimization claims, and benchmarking claims. Each claim type has a different required-N problem.
2. A variance model that decomposes Lighthouse variation into site, class, throttling-mode, host, run-order, time-block, and residual components.
3. Closed-form CI and power formulas for mean-based before/after comparisons, plus a bootstrap plan for Lighthouse's median-run estimator and non-Gaussian metric distributions.
4. A maximum-effort empirical design: 200 URLs, five site classes, three throttling modes, 50 or more runs per cell, controlled container execution, randomized run ordering, raw trace preservation, and open data.
5. Secondary-evidence extraction from the closest published Lighthouse run-count paper and vendor run-count experiments, showing why existing evidence is suggestive but not definitive.
6. A reproducible research package containing analytic tables, a schema for raw observations, a runner scaffold, and a required-N calculator. These artifacts are designed so the empirical study can be executed without changing the inferential plan after seeing the data.

3. What Lighthouse currently measures and why metric choice matters

The default Lighthouse performance score is a weighted composite. In the current public scoring documentation, the performance score weights are First Contentful Paint (10%), Speed Index (10%), Largest Contentful Paint (25%), Total Blocking Time (30%), and Cumulative Layout Shift (25%) (Chrome for Developers, 2026a). Raw metric values are transformed into metric scores on scoring curves derived from the HTTP Archive distribution and then combined into the performance score (Chrome for Developers, 2026a). Therefore, score variance is not simply raw-metric variance scaled linearly. A 300 ms shift in LCP can cause a different score shift depending on where the page sits on the curve.

That point matters because the required run count can differ by metric. FCP and LCP are timing metrics and usually have right-skewed distributions. TBT is often spiky because it is the sum of blocking portions of long main-thread tasks between FCP and Time to Interactive; a single long task can dominate a run (Chrome for Developers, 2026b). CLS is bounded below by zero and can be zero-inflated. Speed Index is visual-progression based and can be sensitive to late visual changes. The composite score is bounded from 0 to 100 and is nonlinear in its inputs. A valid variance study must therefore report CIs for every underlying metric, not only the performance score.

The study should include at least these lab metrics: FCP, LCP, Speed Index, TBT, CLS, the performance score, accessibility score, best-practices score, SEO score, and legacy TTI where a Lighthouse version still reports it. INP requires special handling. It is a Core Web Vital in field data, but default Lighthouse navigation audits do not observe a natural distribution of user interactions across a page lifecycle. An INP extension to this study should use Lighthouse user flows or scripted interaction traces and should be analyzed as a separate interaction-study design, not mixed into the navigation-load variance table.

4. Existing evidence: enough to prove the gap, not enough to close it

The official Lighthouse variability guide is methodologically important but not an empirical CI table. It lists known variability sources, explains which are partially mitigated by simulated throttling, warns against burstable or shared-core hardware, advises against concurrent Lighthouse reports on one host, and recommends aggregate values. It states that the median Lighthouse score of five runs is roughly twice as stable as a single run and notes that Lighthouse CI can collect five runs (GoogleChrome Lighthouse Team, 2026a). This is good operational advice, but it does not answer how wide the interval is for FCP, LCP, SI, TBT, CLS, and score by site class and throttling mode.

The closest academic work is Hericko, Sumak and Brdnik (2021). They selected ten websites from the Alexa Top 500, audited them using Lighthouse v7.3.0 in a desktop-emulated broadband setup, compared $N = 1, 5, 10$, and 100 repeated runs, and concluded that five consecutive runs with median aggregation reduces variability substantially. Their experiment is valuable because it provides real repeated Lighthouse observations. But it is too small and narrow to be definitive: ten sites, one historical Lighthouse version, one emulated desktop/broadband setup, performance score only, and no modern per-metric CI or before/after power analysis. Even so, their Table 2 shows why single runs are unsafe: across 100 runs, performance-score ranges reached 21, 20, 14, 7, 5, 21, 5, 4, 23, and 13 points across the ten sites. The largest observed same-site score range was 23 points.

DebugBear's run-count experiment is also useful but not definitive. It tested three pages, ran 150 tests over one week, used Lighthouse with emulated mobile, packet-level network throttling, and DevTools CPU throttling, and compared one, three, five, and seven runs. It found that three runs reduced TTI coefficient of variation by an average of 37%, seven runs cut it by about half, and FCP variability was reduced less strongly. DebugBear also showed a subtle aggregation failure: Lighthouse CI chooses the median run using a distance formula based on FCP and TTI, so TTI variation can dominate and select a run with an FCP outlier (Zeunert, 2020). This demonstrates why the final study must analyze both per-metric medians and the exact Lighthouse CI median-run rule.

Network throttling evidence further explains why a single CI table is insufficient. Lighthouse has simulated throttling, DevTools throttling, and packet-level or provided throttling modes. The Lighthouse throttling documentation describes simulated throttling as faster and less variable because it records a page load and estimates slower conditions, while DevTools throttling applies throttling during the page load (GoogleChrome Lighthouse Team, 2026b). DebugBear's comparison is directionally consistent: Lighthouse simulated throttling can reduce variance but may trade away fidelity when page behavior depends on real network timing; network-level throttling is more realistic but can introduce more observed variance (Zeunert, 2025). Therefore, run-count guidance must be stratified by throttling mode.

The evidence gap is therefore real and narrow: everyone knows Lighthouse varies; nobody has published a modern, per-metric, per-throttling, per-site-class confidence interval and power table large enough to support before/after claims.

5. Estimands: what exactly is the study estimating?

The most common flaw in performance-measurement articles is an undefined estimand. This study defines four.

Estimand A: single-state lab mean. For a URL, metric, environment, cache policy, form factor, and throttling mode, estimate the mean of repeated Lighthouse observations. This is useful for dashboards and descriptive monitoring.

Estimand B: single-state lab median or median-run report. For the same unit, estimate the median of a metric or the Lighthouse CI median-run report. This is useful because medians are more robust to outliers, but the median-run report is not identical to taking each metric's median independently.

Estimand C: before/after effect for one URL. For a fixed URL and measurement environment, estimate the difference between the after distribution and the before distribution. This is the typical optimization claim.

Estimand D: class-level generalization. For a population of URLs in a site class, estimate typical variance and run-count needs. This supports statements such as "ad-heavy news pages require more runs than static documentation pages under packet-level throttling."

All claims in the final table must state which estimand is being used. A median-of-five local audit answers Estimand B weakly. A business claim that a release improved LCP by 150 ms requires Estimand C. A vendor statement that "five runs are enough" would require Estimand D and should be conditional on site class and effect size.

6. Study design: the definitive matrix

The maximum-effort design is:

- 200 public URLs.
- Five strata with 40 URLs per stratum: static/content, marketing/SaaS, ecommerce/transactional, news/ad-heavy, JavaScript SPA/app-shell. A sixth sensitivity stratum, media/third-party-rich, can be added if the budget permits; otherwise classify such pages inside the closest primary stratum and mark third-party intensity as a covariate.
- Three throttling modes: Lighthouse simulated throttling, DevTools throttling, and provided packet-level throttling. The packet-level mode should use Linux traffic control/netem or an equivalent host-level network emulator and should be treated as the highest-fidelity but potentially highest-variance mode.
- Mobile navigation audit as the primary form factor, because this corresponds to the default performance-scoring use case in PageSpeed workflows. Desktop is a pre-specified extension.
- 50 successful runs per URL per throttling mode, yielding $200 \times 3 \times 50 = 30,000$ successful navigation observations before extensions.

- Raw LHR JSON stored for every run; trace and artifacts stored where available; SHA-256 hashes recorded in the observation table.
- A pilot phase of 10 sites x 10 runs x 3 modes to validate instrumentation before the full matrix.

The primary unit of analysis is a successful Lighthouse navigation observation. Failed runs are not silently dropped. Every failure is assigned a pre-specified error code: navigation timeout, protocol error, page crash, bot block, consent/login redirect, DNS/TLS/network failure, or Lighthouse internal error. Primary tables report successful observations; sensitivity tables report failure rates and analyze whether failures are associated with site class or throttling mode.

7. Sampling rules

Site sampling should be reproducible and resistant to cherry-picking. The sampling frame should be built from a public source such as HTTP Archive origins or an openly documented URL list. The sampling code should draw candidate URLs within each stratum and then apply inclusion rules. A URL is eligible if it is publicly accessible without login, loads in a fresh browser profile from the selected geography, allows automated navigation without violating access constraints, and resolves consistently across at least two pilot loads. A URL is excluded if it blocks headless Chrome categorically, requires personal data entry, redirects to an interstitial that changes the intended page class, or cannot complete a Lighthouse audit in the time budget.

Stratification cannot rely only on domain labels. The final study should assign site class using a two-stage procedure: automated features first, human adjudication second. Automated features include resource count, JavaScript bytes, third-party request share, ad-tech domain count, product/listing semantics, article/news semantics, app-shell indicators, and media bytes. Human adjudication is blind to Lighthouse performance scores and recorded in a classification file. Disagreements are resolved before performance measurement.

8. Execution environment

The design should deliberately over-control the local environment because the study is about Lighthouse variance, not cloud-host variance. The host should be dedicated or bare-metal, not a burstable VM. CPU frequency governor, core isolation, NUMA placement, hyper-threading policy, memory pressure, kernel version, cgroup CPU and memory limits, Chrome version, Node version, Lighthouse version, and container image digest should be recorded. Lighthouse documentation warns that burstable instances, shared cores, and concurrent Lighthouse reports can introduce noise; the protocol therefore forbids concurrent reports on the same host and uses horizontal scaling only when each worker has isolated CPU and memory (GoogleChrome Lighthouse Team, 2026a).

A single run is executed in a fresh Chrome process. Storage reset policy must be pre-specified. The primary navigation-load table uses a cold clear-storage profile, consistent with first-load diagnostics. A separate warm-cache extension can use persistent storage, but it must never be mixed with the cold-load table. Chrome DevTools exposes a Clear Storage option for Lighthouse to simulate a clear first load, and the Lighthouse CLI exposes `--disable-storage-reset` when persistent state is intentionally required (Chrome for Developers, 2025; GoogleChrome Lighthouse Team, 2026c).

The same Docker image should run all workers. The container should not float across Chrome or Lighthouse releases. If the study is repeated after a Lighthouse update, it is a new study version. Lighthouse versioning is not a nuisance variable; it is part of the estimand because scoring curves, audit definitions, and trace processing can change across releases. The public Chrome documentation for Lighthouse 13, for example, explicitly noted that Lighthouse 13 introduced no performance-scoring changes, which is precisely the kind of version fact that should be captured in a measurement paper (Chrome for Developers, 2025b).

9. Randomization, blocking, and drift control

The strongest before/after design is paired and interleaved. For a live production site, run order can confound with time of day, CDN state, release traffic, ad auctions, backend load, and third-party availability. The study therefore uses randomized blocks.

For the variance baseline, construct blocks by host, time window, site, and throttling mode. Within each block, randomize URL order and throttle mode order. Do not execute 50 consecutive runs of one URL in one mode unless the estimand is specifically "consecutive local repeatability." Consecutive runs are useful for one sensitivity analysis, but they understate or overstate variance depending on cache state, network warm-up, server stickiness, and time autocorrelation.

For before/after claims, use an interleaved ABBA or randomized-pair schedule when both versions can be served simultaneously. For example: before-after-after-before within a block, then repeat. If only sequential deployment is possible, split the claim into a weaker quasi-experimental design and include time-block fixed effects, but do not pretend it has the strength of an interleaved comparison.

The protocol records start time for every run and estimates autocorrelation. If lag-1 autocorrelation is material, effective sample size is lower than raw run count. The final calculator therefore has two modes: independent-runs planning and paired/autocorrelation-adjusted planning.

10. Statistical model

For each raw metric, fit a hierarchical model. The simplest useful representation is:

$$\text{metric_value} = \text{grand_mean} + \text{site_class} + \text{throttling_mode} + \text{site}(\text{site_class}) + \text{host} + \text{time_block} + \text{run_order} + \text{residual}.$$

For before/after data, add condition and condition-by-site terms:

$$\text{metric_value} = \text{grand_mean} + \text{condition} + \text{site} + \text{condition_by_site} + \text{throttling_mode} + \text{time_block} + \text{host} + \text{residual}.$$

The core output is a variance decomposition. The final table should show, for every metric and throttling mode, how much variance is attributable to site differences, class differences, host/time effects, and residual run-level noise. The within-URL residual is what drives repeated-run CIs for a single URL. The between-site and condition-by-site terms drive class-level generalization.

Distributions should not be assumed normal blindly. FCP, LCP, Speed Index, and TBT should be analyzed on both raw and log scales. CLS should use a zero-inflated or robust approach if many observations are exactly zero. Performance score is bounded and nonlinear, so bootstrap intervals are preferable for small n. The mean-based formulas in this package are still important because they provide transparent planning bounds and easy auditability, but final publication tables should include bootstrap CIs for the median and median-run estimators.

11. Confidence intervals by run count

For one metric in one condition, the standard 95% CI for the mean is:

$$\text{mean} \pm t(0.975, n - 1) * s / \sqrt{n}.$$

This implies that the CI half-width in units of the run-level standard deviation is $t(0.975, n - 1) / \sqrt{n}$. With five runs, the half-width is about 1.24 standard deviations. With ten runs it is about 0.72 standard deviations. With 50 runs it is about 0.28 standard deviations. That table alone shows why "median of five" should not be treated as a precise estimate.

For an independent equal-n before/after comparison, the approximate 95% CI half-width for the mean difference is:

$$1.96 * \sigma * \sqrt{2 / n},$$

where sigma is the run-level SD in each condition. If sigma = 5 score points and n = 5 per condition, the half-width is about 6.2 points. A claimed 5-point improvement is therefore not well separated from noise in that scenario. If n = 20, the half-width is about 3.1 points. If n = 50, it is about 2.0 points.

For paired before/after runs, the relevant sigma is the standard deviation of paired differences, not the individual-run SD. Pairing can be much more efficient when the before and after runs share the same site, host, time block, and network conditions. If the paired-difference SD is half the independent SD, the required n can fall by roughly a factor of four.

12. Power: how many runs does a before/after claim need?

For a two-sided alpha = 0.05 test with 80% power, independent equal n per condition, the normal approximation is:

$$n = \text{ceiling}(2 * (z(0.975) + z(0.80))^2 * \sigma^2 / \delta^2).$$

Because $z(0.975) + z(0.80)$ is about 2.80, this is approximately:

$$n = \text{ceiling}(15.7 * \sigma^2 / \delta^2).$$

This formula is the most compact answer to the industry question. It says run count grows with the square of noise and falls with the square of the effect size. Double the standard deviation and required n quadruples. Halve the effect size you want to detect and required n quadruples.

Concrete examples for the performance score:

- If sigma = 2 score points and delta = 5 points, n is about 3 runs per condition.
- If sigma = 5 score points and delta = 5 points, n is about 16 runs per condition.
- If sigma = 7.5 score points and delta = 5 points, n is about 36 runs per condition.
- If sigma = 5 score points and delta = 3 points, n is about 44 runs per condition.
- If sigma = 5 score points and delta = 10 points, n is about 4 runs per condition.

These are planning examples, not universal empirical findings. Their purpose is to prevent a category error: no reviewer should ask "is five enough?" without also asking "for what metric, in what mode, with what sigma, and for what effect size?"

13. Median, median-run, and bootstrap intervals

Lighthouse CI does not simply average arbitrary runs. Its documentation supports multiple aggregation strategies, and its median-run convention identifies a representative run rather than taking the independent median of every metric (GoogleChrome Lighthouse Team, 2026d). DebugBear's analysis explains why this matters: if the median-run selector is based on selected metrics, variance in one metric can choose a run that is not median for another metric (Zeunert, 2020).

The final study should therefore report four estimators side by side:

1. Mean of metric values.
2. Median of metric values.
3. Trimmed mean of metric values.
4. Lighthouse CI median-run report and its metric values.

For means, closed-form intervals are reported because they are transparent. For medians and median-run reports, the study uses a stratified bootstrap. At the URL level, resample runs within URL and mode. At the class level, resample URLs within class and runs within URL. For before/after, use paired bootstrap when the design is paired. Report percentile and BCa intervals where stable, and always report the bootstrap seed and number of resamples.

14. Why throttling mode must be a first-class stratum

Lighthouse's simulated throttling is not just another network setting. It changes the measurement process. Simulated throttling records the load and estimates slower network and CPU conditions; this can be fast and relatively low variance. DevTools throttling throttles during the page load and can be more sensitive to real timing. Provided or packet-level throttling pushes the network condition outside Lighthouse and therefore captures more real dependency on backend and API timing. The official throttling documentation describes the tradeoff between simulated and DevTools throttling, and DebugBear's tests make the practical consequence explicit: lower variance is not always higher fidelity (GoogleChrome Lighthouse Team, 2026b; Zeunert, 2025).

A single run-count recommendation that ignores throttling mode is therefore not scientifically meaningful. The final table must be indexed as:

metric x throttling mode x site class x estimator x run count.

Only then can one say, for example, that five runs might be enough for a large score movement on a static page under simulated throttling but not enough for a small TBT movement on a JavaScript-heavy SPA under packet-level throttling.

15. Site class is not decoration; it is a variance source

Different sites have different variance mechanisms. Static documentation pages usually have fewer third-party dependencies and less client-side nondeterminism. Ecommerce pages may include recommendation widgets, personalization, inventory calls, tag managers, experiments, and consent logic. News pages can include ad auctions and heavy third-party scripts. SPA/app-shell pages can shift bottlenecks from network to JavaScript

execution and hydration. Marketing/SaaS pages can have animation, personalization, A/B tests, and analytics tags.

A valid class-level table must not pool these as if they were exchangeable. It should show separate empirical SDs, interquartile ranges, outlier rates, failure rates, and required n values. The study should also report within-class heterogeneity. If the 90th percentile news page requires 30 runs to detect a 5-point score change, while the median static page requires five, both facts matter.

16. Before/after claim protocol

A before/after claim is valid only if it specifies:

- URL and page state.
- Lighthouse version and Chrome version.
- Metric and estimator.
- Throttling mode and device profile.
- Cache/storage policy.
- Number of successful runs per condition and failure policy.
- Estimated before/after difference.
- 95% CI for the difference.
- Power or minimum detectable effect for the selected run count.
- Whether the comparison is paired, interleaved, sequential, or observational.
- Multiple-comparison adjustment if many metrics are tested and only wins are reported.

A claim statement should look like this:

"Under Lighthouse vX, Chrome vY, mobile simulated throttling, cold storage, 20 paired interleaved runs per condition, LCP improved by 310 ms; paired bootstrap 95% CI [180, 440] ms; pre-specified decision threshold 200 ms; no material change in CLS; run failure rate 0%."

A weak claim looks like this:

"The score went from 62 to 68 after one run."

The article's practical contribution is to make the weak claim socially unacceptable in professional performance work.

17. Decision thresholds and industry-facing recommendations

A statistical difference is not necessarily a meaningful performance difference. The final calculator should therefore ask for delta, the smallest effect worth claiming. Recommended defaults:

- Performance score: 5 points for public communication, 3 points for internal regression detection, 10 points for high-confidence coarse claims.
- LCP: 100 ms for fine engineering diagnosis, 250 ms for product-facing claims, 500 ms for coarse claims.
- FCP and Speed Index: 100-250 ms depending on page type.
- TBT: 50 ms for JavaScript optimization work; 100 ms for public claims.
- CLS: 0.01 for fine analysis, 0.05 for product-facing claims, but use a distributional or zero-inflated method.

These thresholds are not universal laws. They are decision thresholds. They should be chosen before running the experiment and justified by user impact, business risk, or regression-gate tolerance.

18. Secondary-evidence table from Hericko et al. (2021)

The package includes a CSV extraction of Hericko et al.'s Table 2. It is included as secondary evidence, not as a replacement for the proposed study. The 100-run score ranges in that paper are reproduced below in abbreviated form:

Site	Label	100-run median	100-run range	100-run SD
W1	AliExpress	92	21	2.9
W2	Amazon India	90	20	2.8
W3	Bola	44	14	1.9
W4	Freepik	89	7	1.5
W5	IKEA	99	5	0.9
W6	Mercari	33.5	21	3.9
W7	Shopify	98	5	1.2
W8	Unsplash	94	4	0.6
W9	Wix	73	23	4.2
W10	Zendesk	87	13	2.5

The table implies two lessons. First, same-URL score ranges can be large enough to cross public score-category boundaries. Second, run-level SD differs materially across pages, which directly implies different required run counts. A universal median-of-five rule cannot be definitive.

19. Open-data specification

The public release should include:

- raw LHR JSON for every successful run;
- trace artifacts where available;
- NDJSON observation table following ``raw_lighthouse_observation.schema.json``;
- a URL classification file;
- a run manifest with randomized ordering;
- container image digest and Dockerfile;
- host metadata;
- Lighthouse and Chrome versions;
- analysis notebooks/scripts;
- bootstrap seeds;
- rendered CI and power tables;
- a required-N calculator.

The schema in this package records metric values, versions, host metadata, benchmark index, error codes, and artifact hashes. It is designed to prevent post hoc ambiguity about whether a reported number came from the same measurement process as the rest of the study.

20. Analysis outputs the final study must publish

For every metric, throttling mode, site class, and estimator, publish:

1. Run-level mean, median, SD, IQR, p05, p95.
2. 95% CI half-width for $n = 1, 2, 3, 5, 7, 10, 15, 20, 30, 50$.
3. Required n per condition for before/after deltas chosen per metric.
4. MDE for $n = 3, 5, 7, 10, 20, 50$.
5. Failure rate and outlier rate.
6. Autocorrelation by run lag and effective sample-size adjustment.
7. Variance-component estimates.
8. Sensitivity comparisons: simulated vs DevTools vs packet-level; mean vs median vs median-run; cold vs warm where measured; first-party-only vs full production page where measured.

The final table should be machine-readable and human-readable. Machine-readable CSV or Parquet files make the calculator auditable. Human-readable tables in the article make the conclusions usable.

21. Threats to validity and countermeasures

Version drift. Lighthouse and Chrome change. Countermeasure: pin versions and treat each version as a study version.

Sampling bias. A 200-site sample can still be unrepresentative. Countermeasure: use public sampling rules, stratification, and report the sampling frame.

Geographic bias. One test region does not represent global users. Countermeasure: define the geography as part of the estimand; add multi-region extension if needed.

Third-party nondeterminism. Ads, analytics, A/B tests, and consent tools are real production variance but can obscure first-party code effects. Countermeasure: primary table measures full production URL; sensitivity table blocks or labels third parties.

Bot mitigation. Some sites treat headless Chrome differently. Countermeasure: detect interstitials, record failures, exclude only by pre-specified rules, and report exclusion counts.

Cache-state ambiguity. Cold and warm loads answer different questions. Countermeasure: strict cache policy and separate tables.

Aggregation mismatch. Median-run is not per-metric median. Countermeasure: report multiple estimators.

Multiple comparisons. Testing many metrics invites cherry-picking. Countermeasure: pre-specify primary metric and threshold; adjust or label exploratory tests.

Autocorrelation. Consecutive runs are not always independent. Countermeasure: randomized time blocks, lag analysis, and effective sample-size correction.

Hardware noise. CPU scheduling and thermal throttling can dominate. Countermeasure: dedicated hosts, no concurrent reports, CPU governance, cgroups/namespaces, and benchmarkIndex recording.

Measurement bias. Seemingly harmless setup details can reverse performance conclusions in systems research. This is not hypothetical; Mytkowicz et al. (2009) demonstrated that innocuous experimental setup changes can produce wrong performance conclusions, and Kalibera and Jones (2013) provide a benchmark-methodology cookbook for repetitions and effect-size confidence intervals. Those lessons transfer directly to Lighthouse variance work.

22. What the final answer should look like after the 30,000-run matrix

After execution, the article should be able to state results in this form:

"For mobile simulated throttling on static/content pages, the median within-URL performance-score SD was X points; the 90th percentile was Y points. Detecting a 5-point score change with 80% power requires N runs per condition at the median page and M runs at the 90th percentile page. Under packet-level throttling on news/ad-heavy pages, the corresponding numbers were ..."

It should also provide metric-specific statements:

"TBT had the highest run-count requirement in SPA/app-shell pages, while CLS required a zero-inflated bootstrap method because many pages had repeated zero values. LCP variance was more sensitive to packet-level throttling than simulated throttling. The median-run estimator reduced outlier influence but sometimes selected non-median values for secondary metrics."

Those statements must come from the empirical data. This manuscript does not fabricate them. It provides the research design and analysis machinery that makes them publishable once the run matrix is executed.

23. Bottom-line run-count guidance before the empirical table exists

Use this interim rule:

1. Never publish a before/after claim from one Lighthouse run.
2. Median-of-five is a screening heuristic, not a proof of a small change.
3. Run a pilot of at least 10 repeated runs in the same metric, mode, site class, and environment.
4. Estimate run-level SD or paired-difference SD.

5. Use $n = \text{ceiling}(15.7 * \sigma^2 / \delta^2)$ for independent equal-n before/after planning at $\alpha = 0.05$ and 80% power.
6. Prefer paired/interleaved designs when possible.
7. Publish the CI for the observed difference.
8. Treat small score differences, especially under high-variance modes, as unresolved unless the CI excludes zero and exceeds the pre-specified meaningful threshold.

This is the defensible replacement for "run it five times." Five can be enough for large effects in low-noise settings. It is not enough for small effects in noisy settings.

24. Conclusion

The page-speed industry has been using a qualitative variance warning as if it were a quantitative inference method. That is the methodological gap. Lighthouse's own documentation, the small academic evidence base, and vendor experiments all point in the same direction: repeated runs matter, aggregation matters, and throttling mode matters. But none of them supplies the tables needed to make before/after claims defensible.

The proposed study closes the gap by making the run-count question empirical and statistical. It specifies the data-generating process, measures every Lighthouse metric, stratifies by site class and throttling mode, decomposes variance, publishes confidence intervals by run count, and converts observed variance into required-N guidance. Its most important conclusion is conceptual: the number of Lighthouse runs is not a tradition. It is a power calculation.

References

Beyer, D., Lowe, S. and Wendler, P. (2019) 'Reliable benchmarking: requirements and solutions', International Journal on Software Tools for Technology Transfer, 21(1), pp. 1-29. DOI: 10.1007/s10009-017-0469-y. Available at: <https://doi.org/10.1007/s10009-017-0469-y>

Chrome for Developers (2025) 'Lighthouse: Optimize your website'. Available at: <https://developer.chrome.com/docs/devtools/lighthouse>

Chrome for Developers (2025b) 'What's new in Lighthouse 13'. Available at: <https://developer.chrome.com/docs/lighthouse/whats-new-13>

Chrome for Developers (2026a) 'Lighthouse performance scoring'. Available at: <https://developer.chrome.com/docs/lighthouse/performance/performance-scoring>

Chrome for Developers (2026b) 'Total Blocking Time'. Available at: <https://developer.chrome.com/docs/lighthouse/performance/lighthouse-total-blocking-time>

Google Developers (2026) 'About PageSpeed Insights'. Available at: <https://developers.google.com/speed/docs/insights/v5/about>

GoogleChrome Lighthouse Team (2026a) 'Score variability'. Available at: <https://github.com/GoogleChrome/lighthouse/blob/main/docs/variability.md>

GoogleChrome Lighthouse Team (2026b) 'Throttling'. Available at: <https://googlechrome-lighthouse.mintlify.app/throttling>

GoogleChrome Lighthouse Team (2026c) 'Lighthouse CLI reference'. Available at: <https://googlechrome-lighthouse.mintlify.app/api/cli-reference>

GoogleChrome Lighthouse Team (2026d) 'Lighthouse CI configuration'. Available at: <https://googlechrome.github.io/lighthouse-ci/docs/configuration.html>

Hericko, T., Sumak, B. and Brdnik, S. (2021) 'Towards Representative Web Performance Measurements with Google Lighthouse', Proceedings of the 7th Student Computer Science Research Conference, pp. 39-42. DOI: 10.18690/978-961-286-516-0.9. Available at: <https://pdfs.semanticscholar.org/7c3e/c7ea067b9b3c29415daaa98f6f886bad6676.pdf>

Kalibera, T. and Jones, R. E. (2013) 'Rigorous Benchmarking in Reasonable Time', Proceedings of the ACM SIGPLAN International Symposium on Memory Management, pp. 63-74. DOI: 10.1145/2464157.2464160. Available at: <https://kar.kent.ac.uk/33611/>

Mytkowicz, T., Diwan, A., Hauswirth, M. and Sweeney, P. F. (2009) 'Producing Wrong Data Without Doing Anything Obviously Wrong!', Proceedings of ASPLOS 2009. Available at: <https://users.cs.northwestern.edu/~robby/courses/322-2013-spring/mytkowicz-wrong-data.pdf>

Zeunert, M. (2020) 'Reducing Variability in Web Performance Metrics'. DebugBear. Available at: <https://www.debugbear.com/blog/web-performance-test-variability>

Zeunert, M. (2025) 'How Different Network Throttling Methods Impact Page Speed Scores'. DebugBear. Available at: <https://www.debugbear.com/blog/network-throttling-methods>

Figures and analytical tables

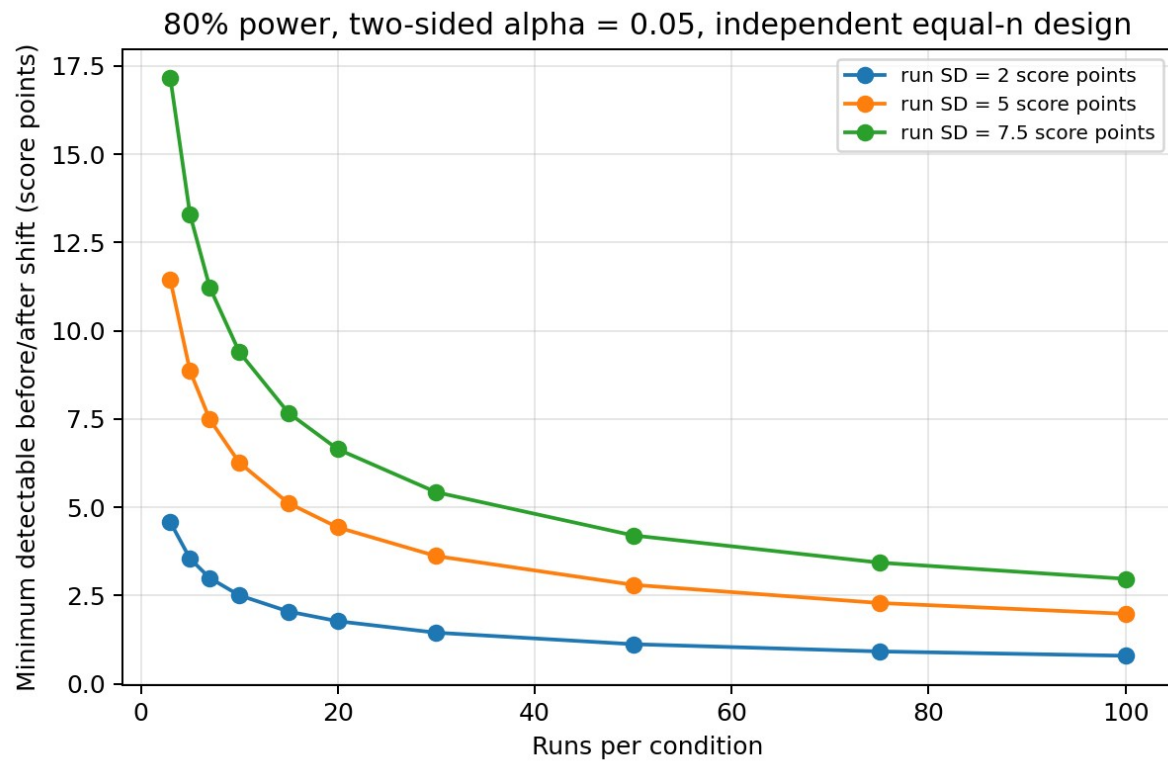


Figure 1. Required-N consequence of run-level score variance. The y-axis is the minimum detectable before/after score shift at 80% power.

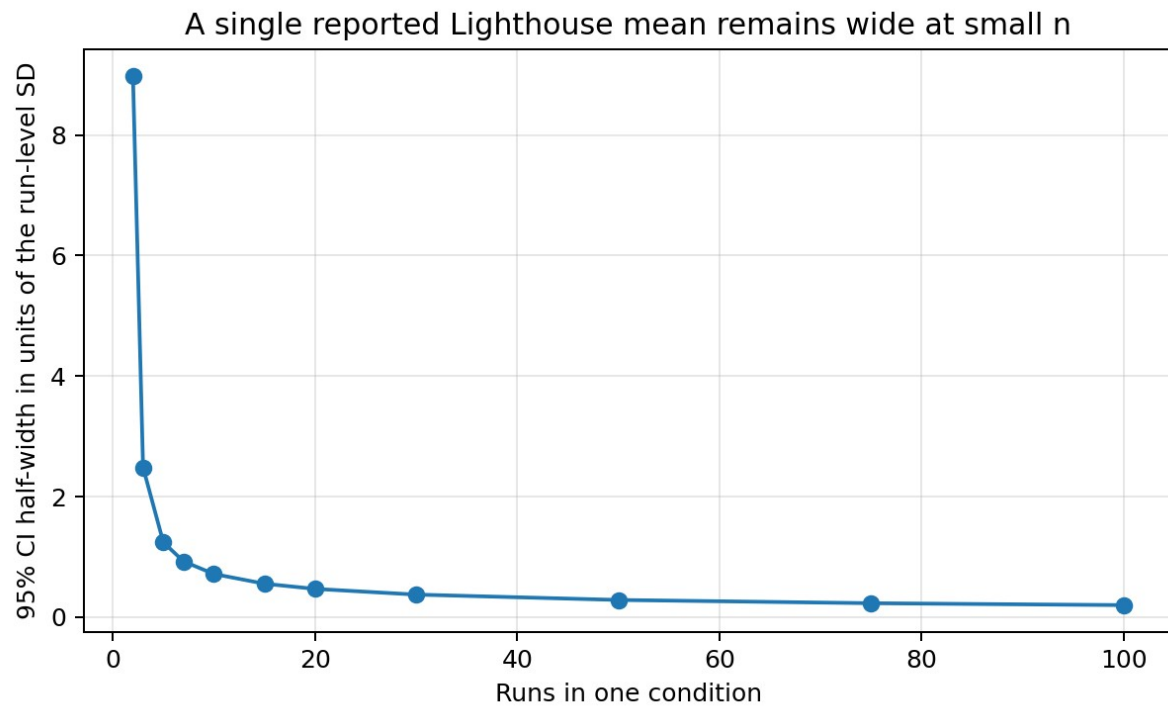


Figure 2. The single-condition mean remains imprecise at small run counts. Five runs leave a 95% CI half-width of about 1.24 run-level SDs.

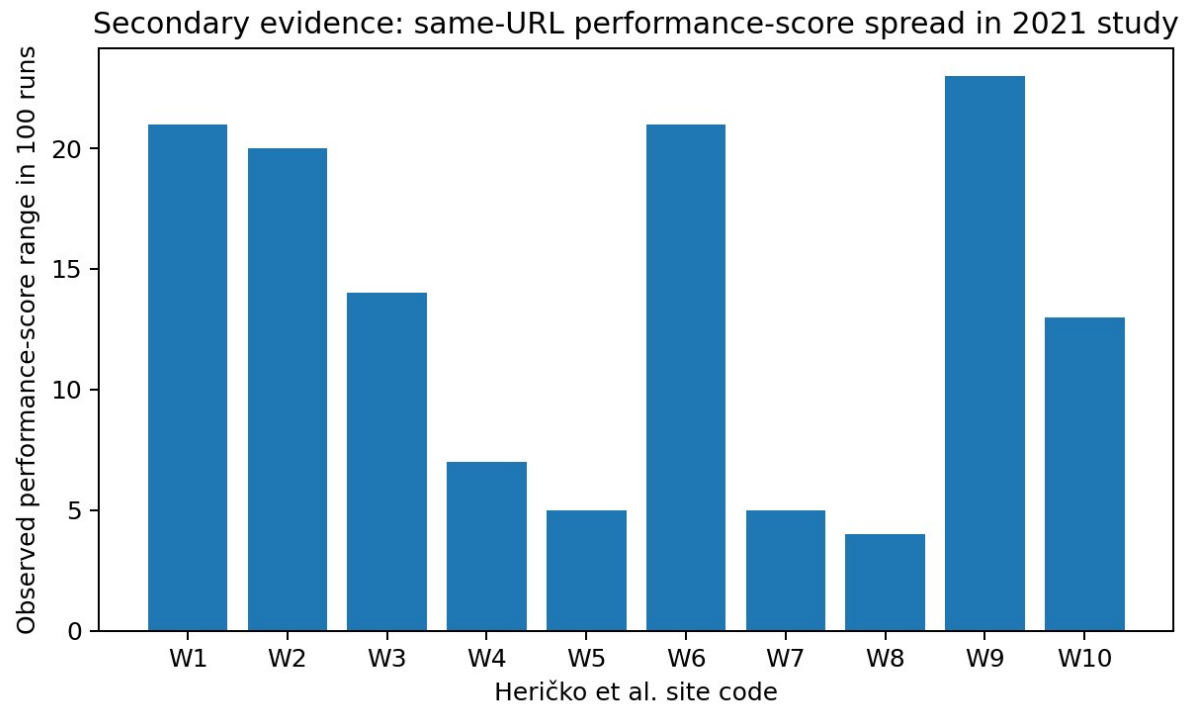


Figure 3. Secondary evidence from Hericko et al. (2021): observed same-URL performance-score ranges in 100 Lighthouse runs.

Table A1. Selected analytical multipliers

n	single mean 95% CI half-width, in SD	independent before/after 95% CI half-width, in SD	independent 80% power MDE, in SD
3	2.484	2.267	2.287
5	1.242	1.458	1.772
10	0.715	0.940	1.253
20	0.468	0.640	0.886
50	0.284	0.397	0.560