International Journal of Scientific Modern Research and Technology
(Volume: 25, Issue: 01, Number: 06) Paper ID: IJSMRT - 26100106
ISSN: 2582-8150

Performance Engineering of Component Hydration Models: An Empirical Study and Optimization Based on Google PageSpeed Insights (Lighthouse) Metrics

Ankit Raitwar[1], Anurag Shrivastava[2], Dr. Puran Gour[3]

[1]M.Tech Scholar, Department of Computer Science & Engineering, NIIST, Bhopal (M.P.), India
[2]Associate Professor & Head, Department of Computer Science & Engineering, NIIST, Bhopal (M.P.), India
[3]Principal, NIIST, Bhopal (M.P.), India

Abstract: *A web page can display its main content while still being slow to respond to a click. In component-based applications, part of this gap can arise from hydration: the browser executes JavaScript to make server-rendered HTML interactive. The cost depends on how much code is delivered, which components are activated, and when their work is scheduled. This paper examines these decisions as a performance-engineering problem rather than treating hydration as a fixed framework feature. It proposes a controlled comparison of full hydration, selective hydration, and partial hydration, with client-side rendering as a separate baseline. Google PageSpeed Insights and Lighthouse provide the principal loading measurements, while browser traces and scripted interactions supply evidence that a navigation audit alone cannot provide. The study will compare Largest Contentful Paint, Total Blocking Time, Cumulative Layout Shift, JavaScript transfer size, and task duration across equivalent page workloads. Interaction latency will be evaluated separately from the Lighthouse performance score. The optimization process will reduce unnecessary client code, prioritize essential controls, and defer components only when their activation remains reliable. Repeated measurements, version records, and functional checks are built into the design so that any reported improvement can be examined and reproduced. This draft establishes the research questions, measurement protocol, and analysis framework; it does not report completed experiments. Its intended contribution is a practical method for choosing hydration boundaries without sacrificing the usability of the page.*

Keywords: *Component Hydration, Server-Side Rendering, Lighthouse, PageSpeed Insights, Core Web Vitals, JavaScript Performance*

How to cite this article: Ankit Raitwar, Anurag Shrivastava and Puran Gour (2026). Performance Engineering of Component Hydration Models: An Empirical Study and Optimization Based on Google PageSpeed Insights (Lighthouse) Metrics, International Journal of Scientific Modern Research and Technology (IJSMRT), ISSN: 2582-8150, Volume-25, Issue-01, Number 06, Oct-2026, pp. 26-32, URL:

https://www.ijsmrt.com/wp-content/uploads/2026/11/IJSMRT-26100106.pdf

Copyright © 2026 by author (s) and International Journal of Scientific Modern Research and Technology Journal. This is an Open Access article distributed under the terms of the Creative Commons Attribution License (CC BY 4.0)

(http://creativecommons.org/licenses/by/4.0/)

IJSMRT-26100106

I. INTRODUCTION

Consider a product page whose image and price appear quickly, but whose quantity selector ignores the first click. A loading screenshot would suggest success; the customer would experience a delay. This example motivates the central concern of the study: visual readiness and functional readiness should be evaluated together.

Hydration connects these two stages in many server-rendered applications. Google's rendering guidance describes its role in adding state and interactivity to existing HTML. However, a rendering architecture by itself does not establish how much work a particular page will require [1].

The research therefore asks which components need client execution, how that work should be ordered,

 ISSN: 2582-8150

and what evidence is needed before an optimization can be considered successful.

II. BACKGROUND AND COMPONENT HYDRATION MODELS

Rendering and hydration address different decisions

Rendering determines where the initial interface is produced. Hydration determines how an existing interface becomes interactive in the browser. Server-side rendering (SSR) and static generation can both produce HTML that is later hydrated. Client-side rendering (CSR), by contrast, creates the interface in the browser and is included here as a rendering baseline, not as a hydration model. These distinctions follow Google's terminology for rendering on the web [1].

The study will use operational definitions so that framework labels do not replace observable behaviour. Each implementation must document its client entry points, activation triggers, transferred scripts, and component boundaries. Streaming HTML, splitting a bundle, and changing a hydration boundary will be recorded as separate implementation decisions.

Table 1: Operational definitions of the hydration models compared in this study

Model	Operational definition	Trade-off to investigate
Full hydration	A server-rendered application root is activated on the client, including its interactive descendants.	Simple coordination, but potentially more startup work than the first interaction needs.
Selective hydration	Server-rendered boundaries can be activated in a priority order supported by the selected runtime.	Essential regions may respond sooner; eventual client work may remain substantial.
Partial hydration	Only designated interactive regions receive client activation; static regions remain HTML.	Less client scope is possible, with additional coordination between regions.
Deferred activation	An activation policy based on visibility, idle time, or interaction; it may be applied to a model above.	Startup work may fall while first-use delay increases.
CSR baseline	Client code constructs the initial component interface.	Useful reference for loading cost, but not a hydration treatment.

Why component boundaries matter

A page containing an article, navigation menu, feedback form, and chart does not necessarily need all four regions to execute client code at startup. In the proposed workload, the article can remain HTML, the menu can activate early, and the chart can activate when requested. The experiment will test whether this allocation helps under the chosen conditions; it will not assume that finer boundaries are always better.

Boundaries also introduce costs. A shared cart state, for example, may require communication between the product controls and header. The implementation record must include this coordination code. Otherwise, a smaller component bundle could conceal work that has simply moved elsewhere.

Functional equivalence

Every treatment must expose the same content and complete the same tasks with keyboard and pointer input. A button that becomes usable only after an unnoticed second click is a functional failure, even if the loading metrics improve. The comparison will therefore pair timing measurements with explicit task outcomes.

III. REVIEW OF GOOGLE TECHNICAL GUIDANCE

This review uses Google-maintained documentation as the requested source base. It is a focused technical review, not a systematic survey of peer-reviewed hydration research. A final dissertation should add independently verified academic studies and position its measured findings against them.

Architecture and loading

Google's rendering overview presents several ways to divide work between server and browser. Its relevance here is the need to compare actual delivery and execution costs rather than infer performance from the presence of server-rendered HTML [1].

The guidance on JavaScript payloads recommends splitting code so that startup receives only what it needs. The React.lazy and Suspense guide illustrates component-level splitting. These sources support testing bundle boundaries, but code splitting alone does not demonstrate partial hydration: a split component can still execute during initial load [8], [9].

 ISSN: 2582-8150

Main-thread work and interaction

Google's long-task guidance explains why script execution can delay other browser work. Its TBT documentation describes a useful lab signal for load responsiveness. INP, in turn, evaluates responsiveness across interactions over a page visit. These sources motivate two measurement tracks: loading audits and interaction scenarios [5], [6], [7].

Measurement and interpretation

PageSpeed Insights combines Lighthouse lab results with available CrUX field data. These represent different evidence: a controlled page load and aggregated experiences from real visits. Google's explanation of lab and field differences cautions that cache state, viewport, and user behaviour can change the observed experience [2], [12].

The scoring documentation explains that the performance score combines transformed metric scores. It also identifies variability between runs. Consequently, this study will retain raw measurements and report distributions rather than use one green score as its main finding [3].

IV. RESEARCH PROBLEM, QUESTIONS, AND OBJECTIVES

The specific problem is the absence of page-level evidence for deciding how much interface activation should occur during startup in the proposed application. The study will investigate that decision under repeatable workloads. It does not claim that no previous research has addressed hydration.

RQ1. How do full, selective, and partial hydration differ in loading metrics and initial client execution for equivalent pages?

RQ2. How do component count, interactive density, and test profile change those differences?

RQ3. Does deferring activation improve loading measurements while increasing first-use delay?

RQ4. Which individual optimizations preserve task success while producing repeatable improvements?

The objectives are to construct equivalent implementations, establish a traceable measurement pipeline, isolate changes through ablation, and derive workload-specific recommendations. The working hypothesis is that reducing unnecessary startup execution will help JavaScript-heavy pages. A competing possibility is that additional requests and activation coordination will offset that benefit. Both outcomes are valid research results.

V. METRICS AND MEASUREMENT FRAMEWORK

The primary endpoints will be Lighthouse LCP and TBT. CLS will be a stability guardrail. FCP, Speed Index, transfer size, and traced execution time will provide supporting evidence. Scripted task latency will be a separate usability guardrail. This priority order will be fixed before the main experiment to discourage selective reporting.

Table 2: Measures, their role in the study, and rules for interpretation

Measure	Meaning and role in this study	Interpretation
LCP	Time to the largest eligible visible image or text block. Primary loading endpoint.	Lower is better; identify the actual LCP element for each run [10].
TBT	Sum of the blocking portions of long tasks in the tool's measurement window. Primary startup endpoint.	Lower is better; does not isolate hydration by itself [5].
CLS	Unitless measure of unexpected layout movement. Stability guardrail.	Lower is better; inspect shifts during activation and later use [11].
FCP / Speed Index	First content paint and progression of visual completion in the loading audit.	Supporting signals; neither proves controls are ready [3].
JavaScript bytes	Transferred script bytes and decoded resource size, recorded separately.	Report initial-load and full-scenario totals; avoid mixing units.
Execution / task time	Script and main-thread work from recorded browser traces.	Support attribution; distinguish runtime work from other scripts [7].
Interaction latency	Input-to-next-paint and input-to-task-completion in fixed scenarios.	A lab scenario measurement, not a CrUX field percentile [6].

Thresholds and score handling

Google's good Core Web Vitals thresholds are LCP at most 2.5 seconds, INP at most 200 milliseconds, and CLS at most 0.1, assessed at the 75th percentile of page visits. A small collection of lab runs cannot establish that a site passes this field assessment [4].

 ISSN: 2582-8150

For context, the scoring page documents Lighthouse 10 weights of FCP 10%, Speed Index 10%, LCP 25%, TBT 30%, and CLS 25%. The experiment will record the actual Lighthouse version and scoring configuration returned by its tools. These documented weights will not be silently assumed for a different release [3].

TBT and INP must remain separate

TBT reflects blocking work during the measured loading period; INP concerns observed interaction responsiveness. Lower TBT can be encouraging without establishing better INP. A component deferred until the first click may avoid startup blocking but still make that click slow. The study will report this trade-off directly [5], [6].

To estimate activation cost, custom performance marks will surround application-defined activation boundaries. These intervals may contain scheduling or waiting time, so they will be labelled as wall-clock activation intervals rather than pure CPU time. Browser traces will be used to investigate what happened inside them.

VI. PROPOSED EMPIRICAL METHODOLOGY

Workloads and implementation strategy

Three small applications will provide contrasting workloads: an article page with a menu and feedback form; a product page with variants, quantity, and cart controls; and a dashboard with filtering, a data table, and a chart. Each will use a frozen dataset, identical visible content, common media assets, and the same interaction acceptance criteria.

Full and selective hydration will first be compared within one runtime where both modes are supported. Partial hydration will be evaluated using a runtime with explicit island boundaries. CSR will provide a separate reference. Comparisons across runtimes will be reported as comparisons of complete implementations, because runtime and hydration policy cannot be fully separated in that part of the study. Exact versions and configuration will be frozen after a pilot.

Test matrix and repetitions

The planned core matrix contains 3 workloads × 4 implementations × 2 profiles × 15 repetitions, giving 360 navigation measurements in the pinned local Lighthouse experiment. The profiles will be defined explicitly as mobile and desktop configurations, with recorded viewport, throttling method, CPU multiplier, and network parameters. This count is a proposed design, not a completed dataset.

Measurements will be organized into randomized blocks, each containing every implementation for one workload and profile. This reduces the risk that temporary machine or network conditions systematically favour a treatment. Runs within a cell will use the same cache policy. A separate warm-cache sensitivity study will be kept outside the cold-browser-cache core matrix.

Two complementary tool tracks

Pinned local Lighthouse will provide the controlled repeated comparison. Public HTTPS deployments will also be audited through PageSpeed Insights to check whether the same broad loading patterns appear in Google's hosted environment. PSI results will be analysed separately: its runtime versions and remote conditions are not under the researcher's full control. The PSI API supports an explicit mobile or desktop strategy and returns Lighthouse results in JSON [13], [14].

Each valid run will retain its raw report, timestamp, final URL, build identifier, tool version, and test configuration. Failed navigations, unavailable assets, and version changes will be logged. Exclusion rules will be defined before collection; a slow but technically valid run will remain in the dataset.

Interaction scenarios and evidence capture

Separate scripted sessions will open the menu, change a product quantity, add an item to the cart, and filter dashboard data. Each scenario will run with an early input at a fixed navigation-relative time and with an input after visual readiness. If the target is unavailable, the script will record that condition rather than silently wait and erase the delay.

The script will record the input timestamp, next visible feedback, final task result, and whether any event was lost or duplicated. A matching performance trace and activation marks will be collected for diagnosis. Scenario order, waiting intervals, and reset state will be published with the experiment so that first-use and repeat-use behaviour can be distinguished.

VII. OPTIMIZATION DESIGN AND IMPLEMENTATION PLAN

Optimization will begin only after the baseline reports and traces have been archived. Each change must address an observed source of work or delay. The process will preserve a baseline build, create one controlled variant, rerun the relevant workload, and check task success before accepting the change.

Reduce the scope of client execution

The first intervention will inspect which rendered regions require browser state or event handling. In the article workload, static text and metadata are candidates to remain server-rendered HTML. In the

 ISSN: 2582-8150

product workload, variant selection and cart controls still need reliable interactivity. The experiment will measure the scripts removed and any new communication cost, rather than count component boundaries as a performance gain.

Split optional code at meaningful boundaries

A chart library or an advanced filter panel will be separated from code required for the first screen. The build output and network log will verify whether the optional module actually leaves the initial request path. Shared dependencies and eager imports can undo an intended split. Total bytes after the full scenario will also be reported, since startup savings may simply postpone a download [8], [9].

Choose activation triggers around user tasks

Essential controls will activate early. A below-the-fold chart may use visibility as its trigger, while a rarely opened editor may activate on demand. These are candidate policies to test, not default prescriptions. Every deferred control must display a meaningful loading state, preserve focus, and handle early input predictably. Event replay will be used only where the selected runtime supports it and the scenario verifies its behaviour.

Shorten long tasks where traces justify it

Repeated data transformation, large synchronous initialization routines, or avoidable setup effects will be investigated independently from bundle size. Work may be reduced, cached within the agreed state model, or divided into smaller scheduled units. A worker is a candidate for suitable computation, but introduces messaging and serialization costs that must also be measured. Google's guidance motivates inspecting task duration instead of treating smaller transfers as sufficient evidence of responsiveness [7].

Protect loading and visual stability

The LCP resource should be discoverable early; delaying a hero image while optimizing scripts can undermine visible loading. Media and deferred panels will reserve space so that activation does not move surrounding content. Initial server and client output will be checked for consistency, with mismatch errors treated as implementation defects. LCP and CLS investigations will follow Google's metric-specific guidance [10], [11].

Ablation and acceptance

The evaluation will compare baseline, client-scope reduction alone, code splitting alone, scheduling alone, and the combined variant for selected workloads. An optimization will be accepted only when its timing benefit is repeatable and the same tasks remain correct. If a change improves one primary metric while worsening another, the result will be presented as a trade-off rather than compressed into a claim of overall superiority.

VIII. ANALYSIS AND REPORTING PLAN

Descriptive and comparative analysis

Each workload-profile-treatment cell will report the median, interquartile range, and observed range for raw LCP and TBT. CLS, transferred bytes, and task latency will appear alongside these endpoints. Individual runs will remain available in the supplementary dataset, including valid slow runs. Plots will show distributions rather than a single best score.

For each randomized block, the treatment value will be compared with its baseline counterpart. The main effect estimate will be the median paired difference in the original unit, with a 95% bootstrap confidence interval obtained by resampling whole blocks. Relative changes may be shown for context, using the baseline median as the denominator. When that denominator is zero, a relative percentage will not be reported.

With only fifteen blocks per cell, uncertainty may remain substantial. A pilot will inform whether more repetitions are needed before the final protocol is frozen. Repeated runs characterize measurement variability for the selected pages; they do not represent fifteen independent websites. Conclusions will therefore remain workload-specific.

Reporting template

Table 3 defines the intended results structure. It contains no experimental values. Separate rows will be produced for every workload, profile, and treatment after measurements are collected. Missing results will remain explicitly missing.

 ISSN: 2582-8150

Table 3: Intended structure of the results report

Output	Required report content
Identity	Workload, build, implementation, profile, tool version, valid / attempted runs.
Primary endpoints	Median LCP in ms and TBT in ms; IQR; paired absolute change; confidence interval.
Guardrails	CLS, successful tasks / attempts, lost or duplicate inputs, first-use delay.
Supporting evidence	Initial and full-scenario JS bytes, traced task durations, activation intervals.
External check	PSI lab results in a separate table; field-data scope and availability if present.
Interpretation	Evidence-supported explanation, regressions, uncertainty, and applicable workload.

Attribution and alternative explanations

A lower TBT value will not automatically be described as faster hydration. The trace may show that an unrelated chart initialization or analytics script changed. Similarly, LCP can improve through resource delivery without any change to activation. The analysis will connect each claim to the corresponding ablation, network record, and trace.

Expected patterns, not findings

The study expects sparse-interaction pages to offer more opportunities to remove client scope than dashboards where most regions are interactive. It also expects deferral to introduce a possible first-use penalty. These are hypotheses for testing. No performance percentage, score increase, or statistically significant difference is claimed in this draft.

IX. VALIDITY, LIMITATIONS, AND FUTURE RESEARCH

Internal and construct validity

Equivalent screenshots alone are insufficient to establish equivalent applications. Treatments must preserve data, validation rules, and state transitions. Shared assets, deployment region, cache headers, and server response behaviour will be checked before comparison. Third-party scripts will be excluded from the core matrix and introduced only in a separate sensitivity study.

The largest design limitation is the possible coupling between hydration policy and runtime. Within-runtime ablations will strengthen causal interpretation, while cross-runtime results will be described more cautiously. A runtime that ships less code may help because of several architectural choices at once. The study must not attribute all of that difference to hydration.

Lighthouse does not expose a universal, framework-independent hydration-duration metric. Custom marks and traces provide additional context, but their boundaries are researcher-defined. Startup TBT and scenario latency measure different aspects of experience and will not be merged into an invented hydration score.

External and ecological validity

Three controlled pages cannot cover every production application. Authentication, real-time updates, localization, third-party widgets, and long-lived sessions may change the trade-offs. Device emulation also cannot reproduce every physical phone. A later extension should include physical devices and a broader sample of production-like workloads.

PSI field data, when available, describes a rolling collection of real visits rather than a single experimental build. An origin-level value must not be presented as a route-specific result. A fresh prototype may have insufficient CrUX data, which is a limitation to record, not evidence of failure. Google's guidance on PSI and lab-field differences explains why these observations require separate interpretation [2], [12].

Reproducibility and extensions

The final submission should include source snapshots, lockfiles, build commands, test scripts, raw reports, exclusion logs, and the analysis procedure. Future work can study navigation after initial load, low-end physical devices, field telemetry with consent, and adaptive activation policies. Adaptive scheduling should be evaluated against a fixed policy before its added complexity is justified.

X. CONCLUSION

Component hydration should be judged by what the user can see and do, as well as by the work the browser performs. This draft proposes a way to examine those dimensions together: equivalent workloads, repeated Lighthouse measurements, separate PSI checks, traced execution, and explicit interaction scenarios.

The anticipated contribution is a reproducible decision process for choosing client boundaries and activation priorities. A useful optimization must reduce a measured cost without losing essential

behaviour or hiding latency until the first click. The experiments remain to be conducted; the final study will replace the proposed analysis with measured results and revise its recommendations accordingly.

References

Note: All sources listed below are Google-maintained technical documentation on web.dev, developer.chrome.com, or developers.google.com. Links were checked during preparation on 5 November 2026; web documentation can change, and the final study should archive the versions used. The research design, operational definitions, proposed test counts, and analysis choices are proposals developed for this study and are not presented as Google recommendations. No experimental dataset or completed empirical finding is asserted in this draft.

[1] Miller, J., & Osmani, A. (2026). Rendering on the web. Google technical documentation. web.dev/articles/rendering-on-the-web

[2] Google for Developers. (2026). About PageSpeed Insights. Google technical documentation. developers.google.com/speed/docs/insights/v5/about

[3] Chrome for Developers. (2026). Lighthouse performance scoring. Google technical documentation. developer.chrome.com/docs/lighthouse/performance/performance-scoring

[4] Walton, P. (2026). Web Vitals. Google technical documentation. web.dev/articles/vitals

[5] Walton, P., & Pollard, B. (2026). Total Blocking Time (TBT). Google technical documentation. web.dev/articles/tbt

[6] Wagner, J., & Pollard, B. (2026). Interaction to Next Paint (INP). Google technical documentation. web.dev/articles/inp

[7] Wagner, J., & Kenny, B. (2026). Optimize long tasks. Google technical documentation. web.dev/articles/optimize-long-tasks

[8] Djirdeh, H., & Posnick, J. (2026). Code splitting with React.lazy and Suspense. Google technical documentation. web.dev/articles/code-splitting-suspense

[9] Djirdeh, H., & Wagner, J. (2026). Reduce JavaScript payloads with code splitting. Google technical documentation. web.dev/articles/reduce-javascript-payloads-with-code-splitting

[10] Walton, P., & Pollard, B. (2026). Optimize Largest Contentful Paint. Google technical documentation. web.dev/articles/optimize-lcp

[11] Osmani, A., & Pollard, B. (2026). Optimize Cumulative Layout Shift. Google technical documentation. web.dev/articles/optimize-cls

[12] Walton, P. (2026). Why lab and field data can be different (and what to do about it). Google technical documentation. web.dev/articles/lab-and-field-data-differences

[13] Google for Developers. (2026). Method: pagespeedapi.runpagespeed. Google technical documentation. developers.google.com/speed/docs/insights/rest/v5/pagespeedapi/runpagespeed

[14] Chrome for Developers. (2026). Lighthouse: Optimize your website. Google technical documentation. developer.chrome.com/docs/devtools/lighthouse