

Laboratory Web Performance Benchmarking of React.js, SvelteKit, and Nuxt.js SPA/MPA Implementations on an Academic Conference Website

Muhammad Ariq Fadhil¹, Riska Dhenabayu², Hafid Kholidi Hadi³, Thwin Htoo Aung⁴

^{1,2,3}Digital Business Study Program, Faculty of Economics and Business, Universitas Negeri Surabaya, Surabaya, Indonesia

⁴Department of Computer Science, University of Bari Aldo Moro, Bari, Italy

Received:

June 12, 2026

Revised:

July 15, 2026

Accepted:

September 2, 2026

Published:

October 24, 2026

Corresponding Author:

Author Name*:

Muhammad Ariq Fadhil

Email*:

muhammadariq.23330@mh.s.unesa.ac.id

DOI:

10.63158/journalisi.v8i5.1757

© 2026 Journal of Information Systems and Informatics. This open access article is distributed under a (CC-BY License)



Abstract. Framework and architecture choices can determine whether academic conference systems remain discoverable, responsive, and accessible for bandwidth-diverse users. This study identifies the framework-architecture combination best suited for comparable deployments. It benchmarks laboratory Core Web Vitals-aligned metrics across six implementations of React.js, SvelteKit, and Nuxt.js, deployed as Single- and Multi-Page Application (SPA/MPA) variants reconstructed from the ICONBIT UNESA conference website. Because Interaction to Next Paint (INP) requires field data unavailable on these low-traffic subdomains, Total Blocking Time (TBT) was adopted as a laboratory proxy for interactivity. Lighthouse, PageSpeed Insights, and JMeter measurements were analyzed using Kruskal-Wallis and Dunn-Bonferroni tests, showing significant differences across all five metrics ($p < 0.001$). SvelteKit achieved the lowest Largest Contentful Paint (LCP) (mean 1.606–1.617 s) with zero TBT for its SSR-MPA variant, but SSR-based MPA deployments showed mobile cold-start penalties (mean LCP 17–18 s), while Nuxt SPA showed weak Slow 4G interactivity (TBT 554 ms). SvelteKit SSR-MPA is recommended only when this risk is mitigated through pre-warming or provisioned compute, with SvelteKit SPA as the safer static alternative otherwise. Findings are bounded by the single reconstructed website and Netlify free-tier environment evaluated.

Keywords: Core Web Vitals, SvelteKit, Single Page Application, Multi-Page Application, Serverless Cold-Start

1. INTRODUCTION

Web performance is no longer only a technical optimization issue; for academic conference information systems, it directly affects discoverability, submission access, and participation from international users. An academic conference information system is an institutional website that publishes conference-specific content (call for papers, schedule, registration, and author guidelines) to a mixed audience of prospective authors, reviewers, and attendees, typically under low, bursty traffic concentrated around submission and registration deadlines, which distinguishes it in content structure and traffic profile from general-purpose institutional websites. Google's Core Web Vitals (CWV), comprising Largest Contentful Paint (LCP), Interaction to Next Paint (INP), and Cumulative Layout Shift (CLS), constitute the industry-standard framework for measuring loading speed, interactivity, and visual stability of web-based systems [1]. Because INP requires field (real-user) data that cannot be captured in a laboratory testing environment, this study substitutes a validated laboratory proxy for interactivity, detailed in Section 2.1. CWV metrics have been integrated into Google's search ranking algorithm, with measurable ranking changes reported following Google's Page Experience Update [1]; as one ranking signal among many, they give technical web performance a documented, though not exclusive, association with the organic discoverability of institutional platforms. For academic conference information systems, which depend on search visibility to attract international participants, reviewers, and authors, this connection carries immediate operational significance.

Contemporary web application development operates under two dominant architectural paradigms: Single Page Applications (SPA) and Multi-Page Applications (MPA). SPA architectures load a single HTML document and update its content via JavaScript, avoiding full page reloads during navigation at the cost of slower initial loading, because content must be fetched and rendered on the client [2]. MPA architectures deliver server-generated HTML documents per navigation request [3], an approach that yielded faster first contentful paint and a higher SEO audit score than client-side rendering in a direct comparison [4], while requiring a full page reload on each subsequent navigation. Kowalczyk and Szandala [2], in an IEEE Access study, showed that SPAs can achieve SEO results equivalent to MPAs when the initial page load is delivered as server-generated HTML (for example, through isomorphic JavaScript) while client-side navigation is

retained, whereas a purely client-side-rendered SPA did not improve its measured SEO score despite optimization; client-side rendering alone therefore remains a meaningful SEO consideration for academically indexed platforms. Iskandar et al. [4] reported that a server-rendered page reached first contentful paint faster than its client-rendered counterpart (2.1 s vs 3.1 s), as the client-rendered version could not display content until its JavaScript had loaded. The boundary between these two paradigms has grown increasingly nuanced with the emergence of meta-frameworks such as Next.js, Nuxt.js, and SvelteKit, which support multiple rendering strategies, such as server-side rendering and static generation, within a unified codebase [5].

This study distinguishes four related but distinct concepts referenced throughout: Single Page Application (SPA), a client-rendered architecture that updates content via JavaScript within one HTML document; Multi-Page Application (MPA), an architecture that serves a distinct HTML document per navigation request, which may be generated statically at build time or dynamically per request; Server-Side Rendering (SSR), a rendering mode in which HTML is generated on the server at request time; and Static Site Generation (SSG), a rendering mode in which HTML is generated once at build time and served unchanged. Critically, MPA is a page-delivery architecture, not a rendering mode: an MPA can be implemented as statically pre-built HTML, as in this study's React MPA, or as per-request SSR, as in this study's SvelteKit MPA and Nuxt MPA. This distinction is developed further in Section 2 and is central to interpreting the mobile cold-start findings in Section 3.2.

Three JavaScript frameworks have achieved widespread adoption for building web applications across both architectural paradigms: React.js, SvelteKit, and Nuxt.js. React.js, originally developed at Facebook (now Meta), provides a component-based architecture with broad community support [6], but its virtual DOM introduces runtime overhead in memory and rendering relative to compiler-based alternatives [7]. SvelteKit builds on Svelte, which compiles components into imperative JavaScript at build time [8] and, without a virtual DOM runtime, produces smaller production bundles [7]. Nuxt.js, built on Vue.js, offers equivalent rendering flexibility and has shown competitive performance in recent framework comparisons, including the best load-test stability among four frameworks in one such study [9]. This motivates the empirical multi-condition evaluation across both dimensions (framework and rendering strategy) undertaken in this study.

Although prior studies have compared web frameworks and rendering strategies, three connected gaps remain in controlled Core Web Vitals benchmarking. Gap 1: the absence of standardized cross-architecture CWV comparisons using controlled experimental designs. This aligns with the broader standardization gap identified by Ghattas et al. [10] in a systematic review of 30 high-quality studies (screened from 6,657 initial records spanning 2010–2024), who reported a lack of standardized evaluation protocols and cross-domain benchmarks as a persistent limitation in the web performance research body. Gap 2: framework comparisons that do not isolate the SPA-versus-MPA architectural distinction within each framework; Zaidan and Suyatno [9] compared four frameworks using Lighthouse, PageSpeed Insights, and JMeter but did not isolate this distinction; Siahaan and Kenidy [11] excluded SvelteKit from their rendering performance comparison; and Putra et al. [7] evaluated bundle-level metrics without deployed CWV assessment across multiple pages and conditions. Gap 3: single-framework or single-configuration studies and limited exploration of academic information systems as an evaluation domain; Savenko and Babii [5] experimentally evaluated a combined SSR, SSG, and incremental static regeneration (ISR) optimization strategy within a single framework (Next.js), while Al Khuwaitem and Hasan [12] reported substantial variability in web-vitals-based performance scores across 485 university websites, establishing academic information systems as a meaningful yet underexplored evaluation domain. Building on these three gaps, a controlled benchmark that isolates SPA-versus-MPA architecture within each of React.js, SvelteKit, and Nuxt.js, under equivalent content and deployment conditions, on an academic information system, remains absent from the literature. To address this gap, this study conducts a controlled six-implementation CWV benchmark across React.js, SvelteKit, and Nuxt.js in both SPA and MPA architectures on a deployed academic conference information system.

The objectives of this research are:

- 1) (RQ1) To quantify CWV-aligned laboratory performance differences among six framework-architecture combinations under controlled desktop conditions.
- 2) (RQ2) To characterize performance differences across mobile emulation, Slow 4G network throttling, and repeat-visit browser caching conditions.
- 3) (RQ3) To identify the framework-architecture combination most suitable for deployment of a system of comparable scale and structure to the studied case, based on integrated CWV, load-testing, and bundle-size evidence.

This study delivers three contributions to the web performance and academic IS literature. First, to the best of our knowledge in relation to Gap 1 (standardized cross-architecture CWV comparison) and Gap 2 (SPA-versus-MPA isolation within each framework) identified above, it provides a controlled six-implementation CWV benchmark comparing React.js, SvelteKit, and Nuxt.js across both SPA and MPA architectures on a real academic website, directly addressing the standardization gap more broadly documented by Ghattas et al. [10]. Second, it documents serverless cold-start, a latency phenomenon systematically reviewed in the serverless computing literature [13], as a reproducible, quantified deployment-level performance characteristic for SSR-based MPA systems on free-tier cloud infrastructure. Third, extending the Lighthouse, PageSpeed Insights, and JMeter combination used in prior framework comparison [9] with bundle size analysis and non-parametric statistical testing, it establishes a replicable template for CWV evaluation in zero-traffic research environments where CrUX field data is unavailable. The remainder of this paper is organized as follows: Section 2 describes the research method; Section 3 presents results and discussion; Section 4 states conclusions and recommendations.

2. METHODS

This study employed a quantitative controlled benchmarking design to isolate the effects of framework and rendering architecture on CWV performance under comparable content, deployment, and measurement conditions. Six website implementations were developed as reconstructions of the ICONBIT UNESA academic conference website (iconbit.unesa.ac.id), representing three frameworks each deployed in both SPA and MPA configurations. Each implementation reproduced identical content across six pages, namely Homepage, About, Call for Papers, Schedule, Registration, and Contact, ensuring content-controlled architectural isolation. All six implementations were deployed to the Netlify production platform (Free tier, region CMH, Ohio, US East) under the project namespace `iconbit2027` and remained live throughout the entire data collection period; Netlify has also been used as a static hosting platform in prior static site generation performance research [14]. This design directly supports RQ1 through cross-implementation desktop benchmarking, RQ2 through mobile, Slow 4G, and repeat-visit comparative testing, and RQ3 through integrated CWV, load-test, and bundle-size evidence.

2.1. Research Variables

The independent variable was the framework-architecture combination, yielding six treatment levels: (1) React.js SPA, (2) React.js static-MPA, (3) SvelteKit SPA, (4) SvelteKit SSR-MPA, (5) Nuxt.js SPA, and (6) Nuxt.js SSR-MPA. The MPA treatments are not uniform in rendering mode: React static-MPA is a statically pre-built multi-entry Vite output, whereas SvelteKit SSR-MPA and Nuxt SSR-MPA execute per-request server-side rendering (SSR) on Netlify Functions. The MPA condition, as operationalized in this study, is therefore confounded with rendering mode and serverless execution model; this confound is the mechanism underlying the mobile cold-start finding reported in Section 3.2, rather than a property of MPA architecture in general. Dependent variables comprised CWV-aligned laboratory performance metrics, namely LCP, CLS, and TBT as a laboratory proxy for INP, supplementary Lighthouse metrics (FCP, TTFB, Performance score, Accessibility, Best Practices, SEO), and server response time under concurrent user load. Control variables included identical page content, visual structure, navigation paths, image assets, interactive components, testing device specification, browser/tool versions, execution scripts, cache handling, and the same Netlify production deployment platform throughout testing. Because each framework-architecture combination was implemented once, the reported differences reflect the characteristics of these specific implementations, including the developer's coding choices, rather than purely intrinsic properties of the framework or architecture in general; this is a threat to construct validity common to single-build comparative benchmarks.

Regarding INP measurement: INP requires real-user interaction data and cannot be measured in a Lighthouse laboratory environment. PageSpeed Insights returned INSUFFICIENT_DATA for INP across all 108 tested URLs due to absent Chrome User Experience Report (CrUX) traffic on the research subdomains. Total Blocking Time (TBT), a Lighthouse laboratory metric also reported in prior framework comparison [9], was therefore adopted as the replicable laboratory proxy for INP. Jain [15] treats TBT as a metric complementary to the Core Web Vitals, which then included FID, and notes that INP is being considered as FID's successor, consistent with using TBT as the available laboratory-testable interactivity indicator in this study. Accordingly, TBT was not interpreted as field INP, but as a controlled laboratory indicator of main-thread blocking and interaction readiness in the absence of CrUX-based INP data.

2.2. Implementation Development

All six implementations used Node.js v22 LTS as the shared runtime environment. React.js (v18) [6] implementations used Vite [16] as the build tool: the SPA variant employed client-side routing via React Router, while the MPA variant used a multi-entry Vite configuration generating separate HTML entry points per page to operationalize the MPA condition through separate HTML entry points and full-document navigation behavior. SvelteKit (v2) [7] SPA was configured with ``ssr: false`` and ``prerender: false``; the MPA variant enabled ``ssr: true`` for server-side rendering per request. Nuxt.js (v3) [11] SPA set ``ssr: false``; the MPA variant used the default ``ssr: true`` configuration.

All implementations reproduced equivalent visual design, navigation structure, image assets, and interactive components. A standardized particle animation component was included on the Homepage across all implementations to ensure content parity. Pre-testing was conducted to resolve routing and layout-stability defects before measurement, ensuring that observed CLS differences reflected final production behavior rather than implementation errors. All six final production builds were deployed to the Netlify production platform under unique subdomains within the iconbit2027 project namespace, and all deployments remained publicly accessible throughout the entire data collection period. Figure 1 and Figure 2 present screenshots of the Homepage across all six implementations under desktop and mobile viewport conditions respectively, confirming visual design parity across all framework-architecture combinations prior to data collection.

2.3. Testing Instruments and Configuration

Following the Lighthouse, PageSpeed Insights, and JMeter combination used in prior framework comparison [9], three measurement instruments were used, complemented by one build-level analysis. Google Lighthouse CLI (v13.3.0) [9] served as the primary CWV measurement instrument, executed on a test machine with an AMD Ryzen 5 5600H processor (6-core), 16 GB RAM, and 64-bit Windows 11, running Headless Chrome 148.0.0.0 (confirmed from the ``environment.hostUserAgent`` field recorded in each Lighthouse JSON report). All Tier 1 runs used ``--preset=desktop`` with no network throttling, representing optimal access conditions. Measurements were automated via PowerShell scripts with standardized inter-run delays to allow server warm-up, executed in a fixed sequential order by implementation and page rather than randomized; this is

acknowledged in Section 4 as a threat to internal validity, since time-of-day and CDN edge-node effects could in principle correlate with implementation identity rather than reflecting a true performance difference. Output was saved as structured JSON files for statistical processing in Python using SciPy v1.17.1 and NumPy v2.4.4.

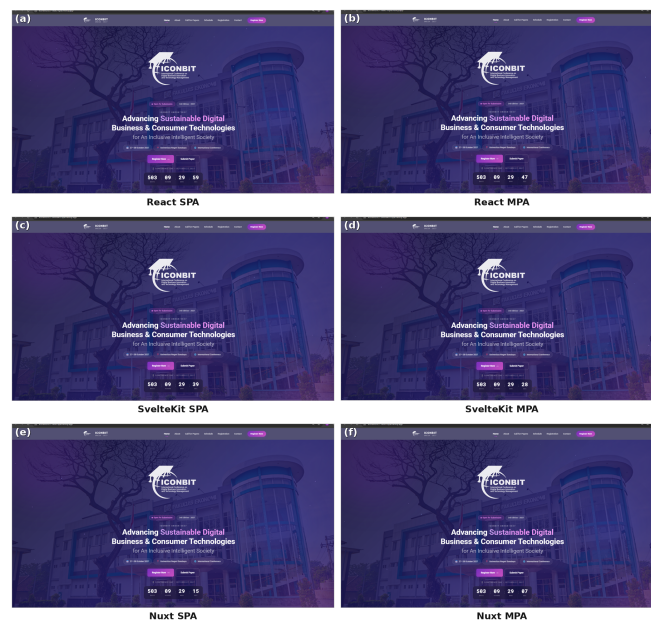


Figure 1. Homepage of six desktop

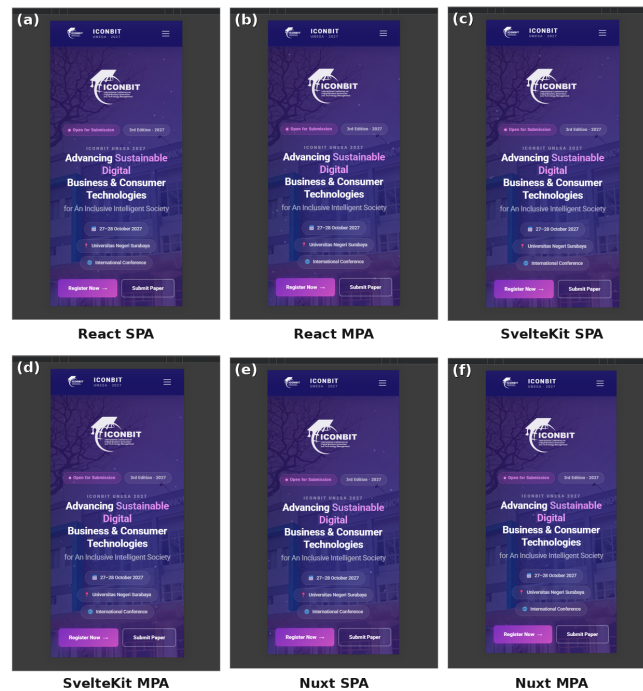


Figure 2. Homepage of six s under mobile

PageSpeed Insights API (v5) (strategy=mobile) served as an independent external measurement instrument using Google's distributed measurement infrastructure. Three independent runs per URL, spaced across separate collection passes to avoid the API's short-lived server-side result caching, were conducted across all implementations and pages, yielding 108 independent JSON files (6 implementations x 6 pages x 3 runs).

Apache JMeter (v5.6.3) with Java 17 LTS executed concurrent load tests targeting the Homepage URL of each implementation. The test plan covered five user scenarios (10, 25, 50, 75, and 100 concurrent users) with a 10-second ramp-up period and a single request loop per user. JMeter load test scenarios were constrained to a maximum of 100 concurrent users due to Netlify free-tier rate limiting policies; load levels of 50 and 100 concurrent users have also been applied in prior JMeter evaluations of web information systems, including an academic conference website [17], [18]. Results were recorded as JMeter Test Log (JTL) files and analyzed for average response time, minimum, maximum, 90th percentile, throughput, and error rate. Bundle sizes were extracted from each implementation's production build output directory immediately after the final deployment build. Because load testing targeted only Homepage requests, JMeter results were interpreted as complementary response-time evidence rather than full transactional scalability evidence.

Table 1. Testing Instruments and Data Collection Summary

Instrument	Data Volume	Condition	Metrics Collected	Validity
Lighthouse CLI v13.3.0 (Tier 1 Desktop)	720 JSON (6x6x20 runs)	Desktop, fast network, first visit (warm)	LCP, TBT, CLS, FCP, TTFB, Performance, Accessibility, Best Practices, SEO	720/720 valid 0 cold-start runs
Lighthouse CLI v13.3.0 (Tier 2 Comparative)	180 JSON (6x1x10x3)	Mobile Desktop Slow 4G Desktop Repeat Visit	LCP, TBT, CLS, Performance score (Homepage only)	180/180 valid

Instrument	Data Volume	Condition	Metrics Collected	Validity
PageSpeed Insights API v5	108 JSON (6x6x3 runs)	Mobile strategy	LCP, TBT, CLS, FCP, TTFB, INP (CrUX Field data)	108/108 valid INP: INSUFFICIENT_DATA (all 108 URLs)
Apache JMeter v5.6.3	30 JTL (6x5 scenarios)	10/25/50/75/100 concurrent users	Avg/Min/Max RT, P90, throughput, error rate	0% error rate all 30 scenarios
Bundle Size Analysis	6 measurements	Production build output	JS total (KB), CSS total (KB), combined total (KB)	Verified from build logs

RT = response time. INP field data unavailable due to absent CrUX traffic on research subdomains.

2.4. Data Collection Procedure

Data collection proceeded in three sequential tiers. Tier 1 comprised 720 Lighthouse CLI runs (6 implementations x 6 pages x 20 runs) under desktop fast-network conditions. A warm-run protocol was applied: any run producing LCP ≥ 8.0 seconds was classified as a cold-start run and excluded from analysis, as such values reflect serverless function initialization (cold-start) latency [13] rather than representative rendering performance. All 720 Tier 1 runs produced LCP < 8.0 s, confirming 100% data integrity. The exclusion criterion was used only to define the Tier 1 warm-run desktop benchmark; cold-start behavior was not discarded from the study but analyzed separately under comparative mobile/serverless conditions. Tier 2 comprised 180 Lighthouse CLI runs (6 implementations x 1 page x 10 runs x 3 conditions) targeting the Homepage under three real-world usage scenarios: (1) mobile device emulation using the default Lighthouse mobile preset with CPU throttling; (2) desktop with Slow 4G network throttling (150 ms RTT, 1.6 Mbps download bandwidth); and (3) desktop with browser cache retained (`--disable-storage-reset` flag) simulating repeat visits. Tier 3 consisted of 108 PSI API calls, 30 JMeter load-test scenarios, and six bundle-size records collected from production build outputs.

2.5. Statistical Analysis

Automated web-performance measurements exhibit run-to-run variability and occasional non-representative outlier readings even when the measured website is unchanged [19]. Accordingly, non-parametric statistical tests were selected for all analyses. Macunluoglu and Ocakoglu [20] showed through simulation that the Kruskal-Wallis test preserves Type-I error across normal and log-normal data distributions when group variances are homogeneous, although its performance deteriorates when variances are heterogeneous. For Tier 1, the Kruskal-Wallis H-test was applied independently to each metric ($N = 120$ per implementation: 20 runs $\times$ 6 pages). Effect size was quantified using epsilon-squared (ϵ^2), interpreted using the eta-squared convention presented by Fiel Peres [21]: negligible ($\epsilon^2 < 0.01$), small (0.01 to < 0.06), medium (0.06 to < 0.14), or large (≥ 0.14). For all metrics with a significant Kruskal-Wallis result ($p < 0.05$), Dunn's post-hoc test with Bonferroni correction was applied to control family-wise error across the 15 pairwise comparisons per metric. For Tier 2, the Mann-Whitney U test with effect size r was applied to compare each condition against the Tier 1 desktop baseline. All computations were performed in Python 3 using SciPy v1.17.1 and NumPy v2.4.4. To further decompose the Tier 1 result into framework and architecture main effects and their interaction, a 3 (framework) $\times$ 2 (architecture) Aligned Rank Transform (ART) ANOVA was additionally applied to each metric, and the Kruskal-Wallis test was re-run independently within each of the six pages as a sensitivity check (Section 3.1). The integrated ranking presented in Table 10 (Section 3.6) is an evidence-based narrative synthesis across four measurement dimensions (CWV, mobile/network stress conditions, load testing, and bundle size), not a formal multi-criteria decision-making model; no explicit weighting, AHP, or TOPSIS procedure was applied.

2.6. Research Design Flow

Fig. 3 presents the five-phase research design flow: (1) Literature Review and Website Analysis, (2) Implementation Development and Deployment, (3) Pre-testing Validation, (4) Multi-instrument Data Collection, and (5) Statistical Analysis and Reporting. The complete dataset, including the raw Lighthouse Tier 1 and Tier 2 JSON files, PageSpeed Insights JSON files, Apache JMeter JTL files, bundle-size records, Python analysis scripts, and the compiled Excel workbook, is archived and will be deposited in a public data repository upon acceptance; the dataset is also available to reviewers upon request during the review process.

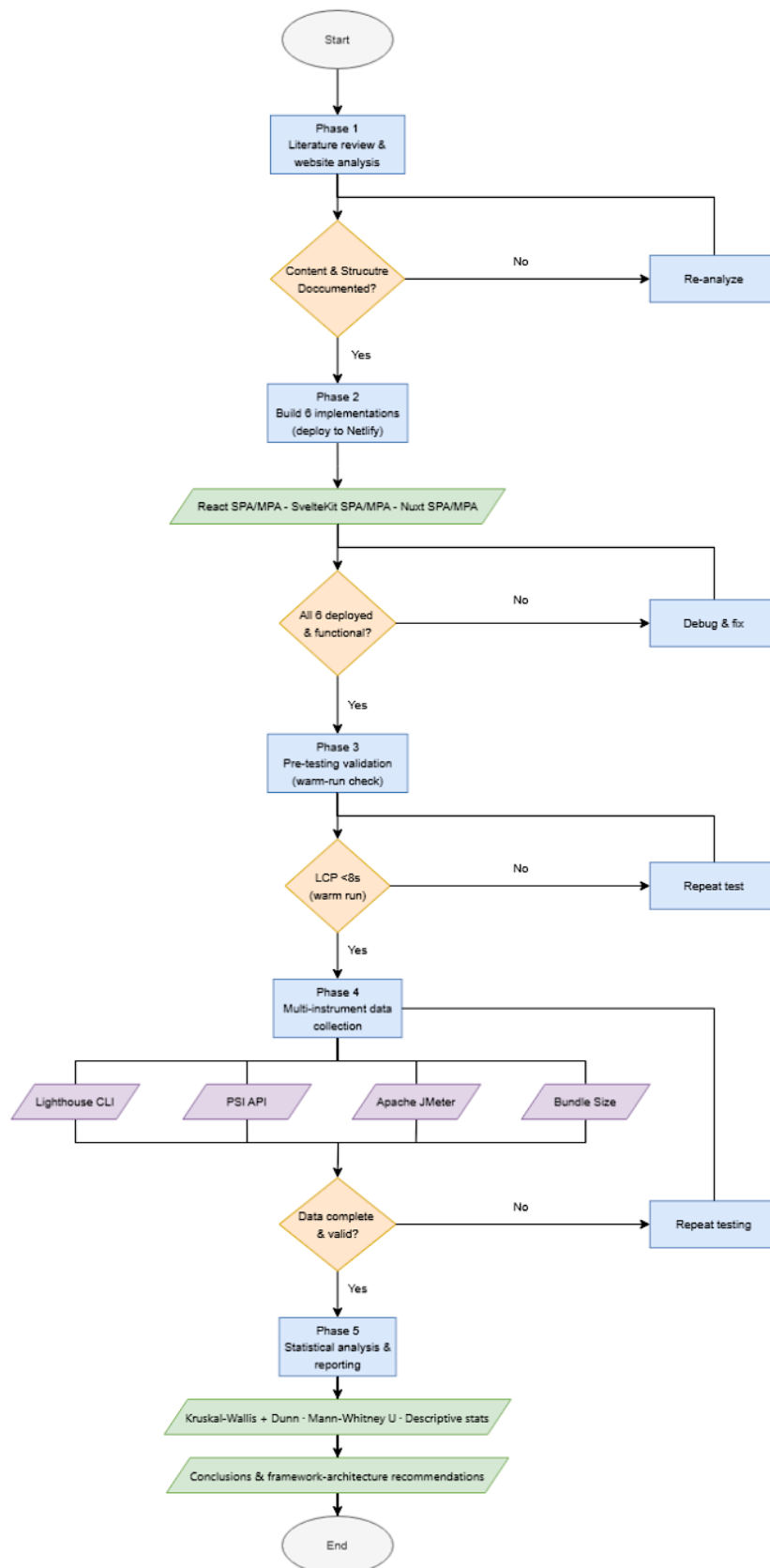

Figure 3. Research Design Flow

Table 2. Research Design Summary

Phase	Activity	Output	Validation Criterion
1	Literature review and website analysis	Content inventory: 6 pages, navigation, assets of ICONBIT UNESA	Content parity confirmed across all 6 implementations
2	Implementation development	6 functional implementations deployed to Netlify (React/SvelteKit/Nuxt SPA/MPA)	All 6 live on Netlify; 0 routing errors; CLS defects resolved pre-collection
3	Pre-testing validation	Production URLs active; warm-run threshold (LCP < 8 s) established	Manual warm-run verification on all 6 implementations prior to batch runs
4	Multi-instrument data collection (Tiers 1, 2, PSI, JMeter, Bundle)	1,044 total data points: 720 + 180 + 108 JSON + 30 JTL + 6 bundle records	100% file validity; 0% JMeter error rate; 0 Tier 1 cold-start exclusions
5	Statistical analysis and reporting	Kruskal-Wallis + Dunn Bonferroni (Tier 1); Mann-Whitney U (Tier 2); descriptive statistics (JMeter)	All p-values Bonferroni-corrected; effect sizes (e2 and r) reported for all significant results

3. RESULTS AND DISCUSSION

This section presents 1,008 Lighthouse/PSI measurements from three instruments, out of 1,044 total records collected across all measurement instruments (Table 1), alongside 30 JMeter load-test scenarios and 6 bundle-size records collected through three measurement instruments and one build-level analysis and discusses their significance in relation to the three research objectives, the existing literature, and the practical context of academic information systems deployment. All six implementations are

referenced as: React SPA, React static-MPA, SvelteKit SPA, SvelteKit SSR-MPA, Nuxt SPA, and Nuxt SSR-MPA. These results answer RQ1 by confirming that framework-architecture selection produces statistically significant differences under controlled desktop conditions, although the magnitude varies across metrics.

3.1. Core Web Vitals: Desktop Fast Network

Table 3 presents Kruskal-Wallis test results across all five Tier 1 metrics (N = 120 per implementation: 20 runs x 6 pages). All five metrics yielded statistically significant differences ($p < 0.001$), with effect sizes ranging from small for TTFB ($e2 = 0.040$) to large for CLS ($e2 = 0.499$), with three of five metrics classified as large effect and FCP as medium, confirming that framework-architecture choice has a meaningful impact on CWV performance across all measured dimensions.

To decompose these pooled differences into framework and architecture main effects and their interaction, a 3 (framework) $\times$ 2 (architecture) Aligned Rank Transform (ART) ANOVA was applied to each metric (Table 4). LCP differences are attributable almost entirely to framework choice (partial $\eta^2 = 0.181$, $p < 0.001$) rather than to SPA-versus-MPA architecture (partial $\eta^2 = 0.005$, $p = 0.067$, not significant) or their interaction (not significant), confirming that SvelteKit's LCP advantage operates independently of rendering strategy. TBT is jointly driven by framework (partial $\eta^2 = 0.219$), architecture (partial $\eta^2 = 0.415$), and a substantial interaction between them (partial $\eta^2 = 0.282$, all $p < 0.001$). CLS is predominantly framework-driven (partial $\eta^2 = 0.381$), while FCP and TTFB are, by contrast, more architecture-driven than framework-driven (partial $\eta^2 = 0.106$ and 0.102 respectively). A page-stratified sensitivity check (Table 5), re-running the Kruskal-Wallis test independently within each of the six pages, confirms that the pooled result for LCP, TBT, and CLS holds in all six pages individually ($p < 0.05$ in every page); FCP is significant in five of six pages (not significant on the Schedule page); and TTFB, already the smallest pooled effect, is significant in only two of six pages, indicating its effect is both small and page-inconsistent.

Table 3. Kruskal-Wallis Test Results, Tier 1 Desktop Fast Network (N = 120 per implementation)

Metric	Threshold	H-statistic	p-value	e2	Effect Size	Best Performer	Worst Performer
LCP (s)	<=2.5 s	147.55	<0.001	0.200	Large	SvelteKit SSR-MPA (median 1.118 s)	React SPA (median 2.665 s)
TBT (ms)	<=200 ms	218.70	<0.001	0.299	Large	SvelteKit SSR-MPA (0.0 ms)	SvelteKit SPA (45.3 ms homepage)
CLS	<=0.10	360.99	<0.001	0.499	Large	React SPA (0.0000)	Nuxt SPA (0.0236)
FCP (s)	<=1.8 s	103.13	<0.001	0.137	Medium	React SPA (0.885 s)	Nuxt SSR-MPA (0.963 s)
TTFB (ms)	<=800 ms	33.19	<0.001	0.040	Small	All implementations ~20 ms	

**Largest effect size across all metrics. All implementations achieved Google Good threshold for every metric under Tier 1 conditions.*

Table 4. Aligned Rank Transform (ART) Factorial ANOVA Results, Framework × Architecture, Tier 1 (N = 720)

Metric	Effect	F	df	p-value	Partial η^2
LCP	Framework	78.880	2, 714	<0.001	0.181 (large)
LCP	Architecture	3.365	1, 714	0.067 (ns)	0.005 (negligible)
LCP	Framework × Architecture	1.553	2, 714	0.212 (ns)	0.004 (negligible)
TBT	Framework	100.367	2, 714	<0.001	0.219 (large)
TBT	Architecture	505.715	1, 714	<0.001	0.415 (large)

Metric	Effect	F	df	p-value	Partial η^2
TBT	Framework × Architecture	139.835	2, 714	<0.001	0.282 (large)
CLS	Framework	219.802	2, 714	<0.001	0.381 (large)
CLS	Architecture	43.052	1, 714	<0.001	0.057 (medium)
CLS	Framework × Architecture	31.544	2, 714	<0.001	0.081 (medium)
FCP	Framework	9.061	2, 714	<0.001	0.025 (small)
FCP	Architecture	84.807	1, 714	<0.001	0.106 (medium)
FCP	Framework × Architecture	11.419	2, 714	<0.001	0.031 (small)
TTFB	Framework	5.367	2, 714	0.005	0.015 (small)
TTFB	Architecture	81.157	1, 714	<0.001	0.102 (medium)
TTFB	Framework × Architecture	11.719	2, 714	<0.001	0.032 (small)

Table 5. Page-Stratified Kruskal-Wallis Sensitivity Check (n = 20 per implementation per page)

Metric	Pages significant (p < 0.05)	Pages not significant
LCP	6 / 6	N/A
TBT	6 / 6	N/A
CLS	6 / 6	N/A
FCP	5 / 6	Schedule (p = 0.498)
TTFB	2 / 6	Contact, About, Registration, Homepage (p = 0.07–0.25)

Table 6 presents mean LCP values per implementation and page. SvelteKit SPA (mean 1.606 s) and SvelteKit SSR-MPA (mean 1.617 s) consistently achieved the lowest LCP across all six pages. Dunn post-hoc testing with Bonferroni correction confirmed both SvelteKit implementations as significantly superior to all others ($p < 0.001$), while no significant difference was found between them ($p = 1.000$). This finding directly addresses RQ1 and demonstrates that Svelte's compile-time component model, which eliminates the runtime

virtual DOM library and produces bundles approximately 40% smaller than React.js, appears to be the strongest observed determinant of LCP under the controlled desktop condition, operating independently of the SPA-versus-MPA architectural choice. This is broadly consistent with the framework-level performance differentiation reported by Zaidan and Suyatno [9], who found Astro JS to outperform Next JS, Nuxt JS, and SvelteKit across most Google Lighthouse and PageSpeed Insight Web Vitals metrics, with SvelteKit excelling only on Cumulative Layout Shift; the present findings similarly indicate that framework identity, rather than rendering strategy, is the dominant determinant of LCP, though the two studies differ in which framework proves fastest. Putra et al. [7] attributed Svelte's bundle compactness to compile-time optimization; the deployed CWV measurements here corroborate their bundle-level findings with user-observable LCP metrics across 120 observations per implementation.

Table 6. Mean LCP (seconds) per Implementation and Page, Tier 1 Desktop Fast Network

Implementation	Home	About	CFP	Schedule	Reg.	Contact	Mean	IQR (25th–75th, pooled N=120)
React SPA	2.476	2.420	2.482	1.463	2.486	1.928	2.209	1.292–2.757
React static-MPA	2.738	1.838	1.382	1.893	2.508	1.943	2.050	1.051–2.595
SvelteKit SPA	2.012	1.340	1.344	1.963	1.881	1.095	1.606	0.903–2.486
SvelteKit SSR-MPA	2.422	1.428	1.427	1.815	1.649	0.963	1.617	0.899–2.512
Nuxt SPA	2.639	2.191	2.501	2.685	2.510	2.261	2.465	2.509–2.748
Nuxt SSR-MPA	2.862	1.998	2.310	2.839	2.485	1.442	2.323	1.536–2.804

Table 3 reports median (top/bottom performers only); Table 6 reports mean and IQR for all six implementations. Different purposes, not conflicting.

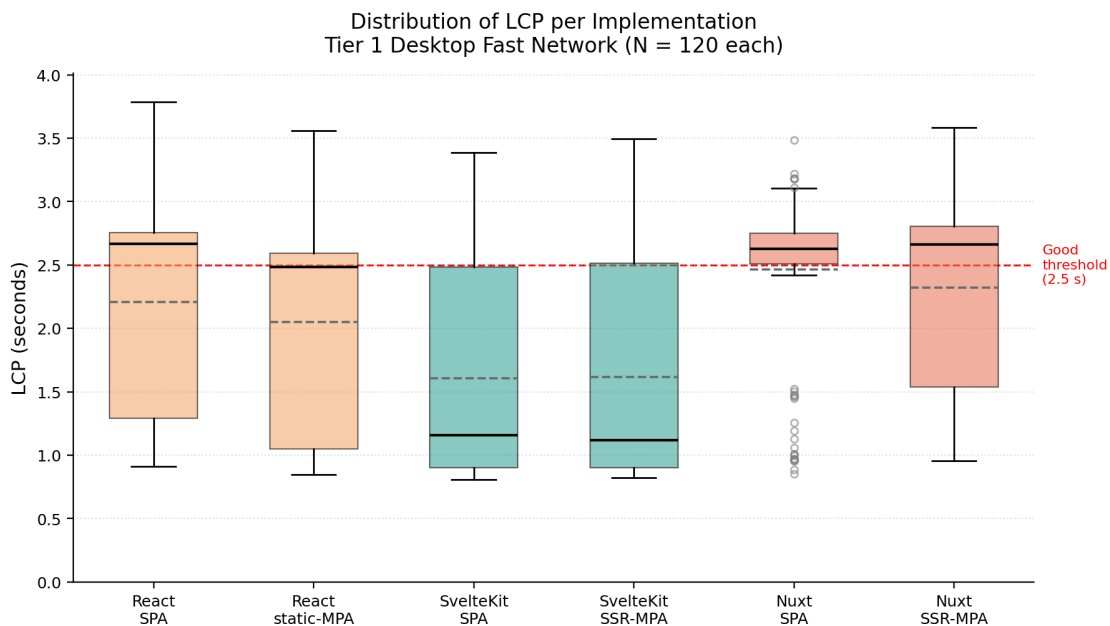


Figure 4. Distribution of LCP per Implementation, Tier 1 Desktop Fast Network (N = 120 each). Boxes show the interquartile range with the median line; the dashed red line marks the 2.5 s Good LCP threshold

Within the React.js framework, the benefit of delivering pre-rendered HTML through MPA was statistically confirmed (React static-MPA vs React SPA, $p = 0.007$), consistent with Iskandar et al. [4], who reported faster first contentful paint for a server-rendered page than for its client-rendered counterpart. No significant LCP difference was observed between Nuxt SPA and Nuxt SSR-MPA ($p = 1.000$), suggesting that for Nuxt.js, the Vue.js runtime overhead is sufficiently dominant to neutralize the rendering-strategy benefit. This suggests that framework-intrinsic characteristics can moderate the impact of rendering strategy, as observed here in the specific Nuxt.js case. For TBT, SvelteKit SSR-MPA recorded 0.00 ms across all six pages, the absolute best result, while SvelteKit SPA's homepage TBT elevation (45.3 ms) is attributable to a standardized particle animation component, not an architectural characteristic. SvelteKit SSR-MPA's zero TBT is consistent with reduced runtime JavaScript execution and the absence of virtual DOM diffing overhead on the main thread, a structural consequence of Svelte's compile-time approach that produces no virtual DOM diffing overhead during user interaction [7]. All six implementations remained well within the Good TBT threshold (< 200 ms). CLS, FCP, and TTFB were all within good thresholds for every implementation, with no meaningful SPA-versus-MPA TTFB difference under warm-server CDN conditions. Accessibility scores

ranked 0.89 to 0.96, and Best Practices and SEO scores were 1.000 universally, indicating consistent implementation quality; automated accessibility evaluation tools are an established means of auditing educational websites against WCAG [22].

3.2. Deployment-Stress Conditions: Mobile, Slow 4G, and Repeat Visit

To address RQ2, Table 7 presents Homepage LCP across three deployment-stress conditions and Mann-Whitney U test results for the repeat-visit comparison. The purpose of this tier was not to repeat desktop benchmarking, but to test whether the same ranking remained stable under access conditions typical of mobile and bandwidth-diverse users.

Table 7. Tier 2 Comparative Conditions: Homepage LCP (seconds) and Repeat-Visit Statistics

Implementation	Desktop Baseline	Mobile LCP (s)	Mobile Warm/10	Slow 4G LCP (s)	Slow 4G TBT (ms)	Repeat LCP (s)	Cache p-value
React SPA	2.476	7.615	8/10	7.653	44.9	1.971	0.495 ns
React static-MPA	2.738	7.183	6/10	7.702	68.2	2.005	0.262 ns
SvelteKit SPA	2.012	6.855	8/10	6.985	48.6	1.685	0.524 ns
SvelteKit SSR-MPA	2.422	17.701	0/10	7.048	0.0	1.795	0.013 *
Nuxt SPA	2.639	6.680	2/10	16.262	554.2	3.006	0.056 ns
Nuxt SSR-MPA	2.862	17.995	0/10	7.408	59.6	1.593	0.001 **

*Bold red = critical/degraded values. Bold green = best value. Yellow = SSR cold-start affected. ns = not significant; * $p < 0.05$; ** $p < 0.01$. Nuxt SPA Slow 4G TBT = Needs Improvement (> 200 ms threshold).*

Under mobile emulation, SvelteKit SSR-MPA and Nuxt SSR-MPA produced 0 of 10 warm runs, with all runs returning cold-start LCP values of 17.7 s and 18.0 s respectively, a complete failure of the warm-run protocol applied in Tier 1. This finding reveals a critical deployment-level characteristic: SSR-based MPA implementations on Netlify's serverless infrastructure experience function initialization latency that compounds with mobile CPU throttling. Golec et al. [13] documented this cold-start phenomenon systematically in a review of serverless computing, identifying on-demand function initialization as a well-established source of latency in serverless deployment. This has direct implications for academic IS practitioners: institutions deploying SSR-MPA systems on free-tier cloud infrastructure must implement pre-warming strategies or provision dedicated compute instances to avoid severe mobile user experience degradation. SPA implementations, serving static CDN assets, were unaffected by cold-start behavior, confirming that this is an architecture-deployment interaction effect rather than a framework weakness. Therefore, mobile cold-start should be interpreted as an interaction between SSR deployment mode and serverless infrastructure, not as a standalone framework-level performance property.

Under Slow 4G throttling, Nuxt SPA recorded TBT of 554.2 ms, the only Needs Improvement result across all Tier 2 conditions. This is plausibly associated with Nuxt SPA's larger JavaScript payload: its 257.62 KB JS bundle is 88.17 KB larger than SvelteKit SPA's 169.45 KB, a difference corresponding to a theoretical bandwidth-based estimate of approximately 450 ms additional download time on a 1.6 Mbps connection ($88.17 \text{ KB} \div 195.3 \text{ KB/s}$), consistent with, though not a direct measurement of, the prolonged main-thread blocking observed during parsing. Hydration cost (Nuxt SPA hydrates the full Vue.js application client-side, unlike SvelteKit's compiled output) and the shared homepage particle-animation component are additional plausible contributors; because the Nuxt SPA effect is specific to the Slow 4G condition while the animation is present under every network condition, bundle size and hydration cost are considered the primary candidate explanations, with the animation as a secondary, page-specific contributor. This finding is consistent with prior observations that large JavaScript payloads particularly penalize users on slower network connections [2], [7], and that sending less JavaScript at initialization improves interactivity [2]; the present results show that this disadvantage is quantifiably measurable in deployed CWV assessments. For

academic conferences targeting participants from bandwidth-diverse regions, this constitutes a direct accessibility barrier.

Under repeat-visit conditions, Nuxt SSR-MPA demonstrated the largest cache benefit: LCP declined from 2.862 s to 1.593 s (-51%, $p = 0.001$, $r = 0.60$, Large effect). SvelteKit SSR-MPA also showed significant improvement ($p = 0.013$, $r = 0.46$, Medium effect). These findings are consistent with Malavolta et al. [23], who empirically demonstrated that populated browser cache produces significantly lower page load times in progressive web apps, with a medium effect size. Nuxt SPA was the only implementation without a significant cache effect ($p = 0.056$, ns), likely due to absent aggressive cache-control header configuration on its CDN-delivered bundle assets.

3.3. Load Testing: Apache JMeter

Table 8 presents average server response times across five concurrent user scenarios. All 30 test scenarios recorded a 0% error rate, confirming zero-error Homepage response stability under the tested 10–100 concurrent-user scenarios and meeting the zero-error criterion applied in prior academic-website load testing [24].

Table 8. Apache JMeter Response Time and Throughput per Scenario

Implementation	Users	Avg (ms)	P90 (ms)	Max (ms)	Throughput (req/s)	Error Rate
React SPA	10	115.2	124.3	397	1.12	0%
React SPA	25	86.8	106.2	268	2.62	0%
React SPA	50	85.6	94.0	272	5.14	0%
React SPA	75	81.5	85.6	305	7.68	0%
React SPA	100	108.3	176.0	1078	10.18	0%
React static-MPA	10	92.8	96.7	274	1.12	0%
React static-MPA	25	88.9	94.6	277	2.62	0%
React static-MPA	50	92.9	118.6	291	5.14	0%
React static-MPA	75	83.6	86.6	290	7.68	0%

Implementation	Users	Avg (ms)	P90 (ms)	Max (ms)	Throughput (req/s)	Error Rate
React static-MPA	100	79.6	83.1	289	10.18	0%
SvelteKit SPA	10	108.1	125.0	359	1.12	0%
SvelteKit SPA	25	149.9	325.6	512	2.62	0%
SvelteKit SPA	50	86.8	92.2	305	5.14	0%
SvelteKit SPA	75	96.9	128.6	574	7.68	0%
SvelteKit SPA	100	78.6	82.2	283	10.18	0%
SvelteKit SSR- MPA	10	121.5	130.0	292	1.12	0%
SvelteKit SSR- MPA	25	106.9	122.6	282	2.62	0%
SvelteKit SSR- MPA	50	113.7	119.2	504	5.14	0%
SvelteKit SSR- MPA	75	107.2	118.2	311	7.67	0%
SvelteKit SSR- MPA	100	101.3	105.0	314	10.17	0%
Nuxt SPA	10	102.0	137.8	280	1.12	0%
Nuxt SPA	25	83.4	82.2	281	2.62	0%
Nuxt SPA	50	87.8	87.5	484	5.14	0%
Nuxt SPA	75	81.9	86.6	282	7.68	0%
Nuxt SPA	100	88.1	137.2	312	10.18	0%
Nuxt SSR-MPA	10	159.6	183.2	338	1.12	0%
Nuxt SSR-MPA	25	153.5	170.8	355	2.62	0%
Nuxt SSR-MPA	50	143.1	145.6	354	5.14	0%
Nuxt SSR-MPA	75	153.2	170.6	360	7.68	0%

Implementation	Users	Avg (ms)	P90 (ms)	Max (ms)	Throughput (req/s)	Error Rate
Nuxt SSR-MPA	100	163.5	199.4	620	10.17	0%

Computed From the 30 archived JTL Files. Throughput reflects the fixed test schedule rather than server-side differences; P90 and Max reveal spikes (e.g. React SPA, 1,078 ms at 100 users) not visible from the average alone.

SvelteKit SPA achieved the fastest response at 100 concurrent users (79 ms), followed by React static-MPA (80 ms). Nuxt SSR-MPA was consistently the slowest across all five scenarios (range 143 to 164 ms), a pattern plausibly associated with per-request SSR rendering overhead under concurrent requests, coherent with the TBT and LCP findings in Sections 3.1 and 3.2. This is consistent with Savenko and Babii [5], who note that SSR renders each page on the server at every request whereas statically pre-generated pages minimize response times and server load, and who observed the lowest latency for static site generation in their load-tested Next.js prototype. SvelteKit SPA's isolated elevated reading at 25 concurrent users (150 ms) was not reproduced at adjacent load levels (87 ms at 50u, 97 ms at 75u), consistent with evidence from repeated Lighthouse audits that single measurements can produce non-representative outlier readings driven by transient network, server, and client-side variability [19]. The 0% error rate across all 30 scenarios confirms infrastructure stability under the tested load range, consistent with the zero-error operation reported up to 200 concurrent users for another academic conference website [18]. These load-test results provide complementary evidence toward RQ3 by characterizing server-side responsiveness under concurrent load.

3.4. Bundle Size Analysis

Table 9 presents production JavaScript and CSS bundle sizes extracted immediately after the final deployment build. Bundle size provides a plausible mechanistic link between framework choice and the LCP and TBT outcomes observed in Sections 3.1 and 3.2.

Table 9. Production Build Bundle Size per Implementation

Implementation	Framework	Arch	JS (KB)	CSS (KB)	Total (KB)	Rank	vs. React SPA
SvelteKit SPA	SvelteKit	SPA	169.45	74.09	243.54	1st	-39%

Implementation	Framework	Arch	JS (KB)	CSS (KB)	Total (KB)	Rank	vs. React SPA
Nuxt SSR-MPA	Nuxt.js	MPA	261.65	5.73	267.38	2nd	-33%
SvelteKit SSR-MPA	SvelteKit	MPA	159.23	126.50	285.73	3rd	-29%
Nuxt SPA	Nuxt.js	SPA	257.62	77.83	335.45	4th	-16%
React SPA	React.js	SPA	326.41	73.26	399.67	5th	baseline
React static-MPA	React.js	MPA	288.13	113.67	401.80	6th	+0.5%

Green shading: SvelteKit group (smallest bundles). Orange shading: React group (largest bundles). Baseline: React SPA total bundle.

SvelteKit SPA produced the smallest total bundle (243.54 KB, -39% versus React SPA), and SvelteKit SSR-MPA the third smallest (285.73 KB, -29%). Both are substantially smaller than React implementations at approximately 400 KB each. This differential is a structural consequence of Svelte's compile-time component model, which compiles components to efficient imperative JavaScript without shipping a runtime library, a framework-intrinsic characteristic confirmed by Putra et al. [7]. This bundle-size gap is associated with, though not a direct measurement of, the Slow 4G TBT finding for Nuxt SPA discussed in Section 3.2: on a 1.6 Mbps connection, the 160 KB total bundle size difference between React SPA (399.67 KB) and SvelteKit SPA (243.54 KB) corresponds to a theoretical bandwidth-based estimate of approximately 800 ms additional download time ($160 \text{ KB} \div 195.3 \text{ KB/s}$). The link from bundle size to download, parse, and execution time, and thereby to delayed interactivity, is consistent with the front-end performance framework described by Ekpobimi et al. [25]; the further step to elevated TBT is inferred from the present measurements. Nuxt SSR-MPA's compact CSS (5.73 KB) is attributable to SSR extracting only page-specific CSS per request, rather than bundling all styles together, a server-side rendering optimization not available to CSR-based SPA architectures. This bundle-size evidence supports RQ1 and RQ3 by offering a mechanistic explanation for the LCP and TBT differences reported in Sections 3.1–3.2.

3.5. PageSpeed Insights Mobile Consistency Analysis

PSI API mobile measurements (108 JSON files, 3 independent runs per URL, re-collected using a pass-based protocol to avoid the API's short-lived server-side result caching) using Google's independent measurement infrastructure provided an external consistency check for the key mobile findings. SvelteKit SSR-MPA Homepage recorded a mean PSI mobile LCP of 17.143 s, consistently slow across all three runs (17.14–17.15 s, SD = 0.008), matching its Tier 2 cold-start value of 17.701 s (0 of 10 warm runs, Table 7). Nuxt SSR-MPA Homepage recorded a mean PSI mobile LCP of 14.077 s, driven by an intermittent pattern across the three runs (6.9 s, 17.6 s, 17.7 s, SD = 5.074), consistent with its Tier 2 cold-start value of 17.995 s (0 of 10 warm runs, Table 7). Both results confirm that the serverless SSR cold-start behavior identified in Tier 2 is reproducible across an independent measurement infrastructure, and is attributable to the serverless execution model rather than an intrinsic property of the MPA architecture or the underlying framework. React SPA Homepage also recorded an elevated mean PSI mobile LCP (17.247 s, consistent across all three runs, SD = 0.070) despite having no serverless component; this is not attributable to serverless cold-start, is consistent with an intermittent slow-loading pattern also present in a minority of its Tier 2 runs (2 of 10 runs at approximately 18 s, Table 7), and is reported as an unresolved limitation in the Conclusion rather than attributed to a specific cause. INP returned INSUFFICIENT_DATA for all 108 URLs due to the CrUX minimum traffic threshold, confirming that TBT remains the only available interactivity metric for this deployment; because PageSpeed Insights combines Lighthouse data with CrUX field data [15], its output without CrUX data remains synthetic/laboratory evidence rather than real-user field evidence. The relative ranking of implementations across PSI and Lighthouse CLI instruments remained consistent: SvelteKit implementations maintained the lowest LCP values and React/Nuxt implementations the highest in both instruments. The consistent rankings observed here across independent measurement infrastructure indicate that Tier 1 findings are robust, in line with the more consistent evaluation practices called for by Ghattas et al. [10].

3.6. Integrated Ranking and Recommendation

Table 10 integrates findings across all four measurement dimensions to produce an evidence-based ranking and directly addresses RQ3.

Table 10. Conditional Implementation Ranking Based on Controlled Performance and Deployment Risk

Rank	Implementation	LCP Mean (s)	TBT Mean (ms)	JMeter 100u (ms)	Bundle (KB)	Mobile Warm/10	Key Characteristic
1	SvelteKit SSR-MPA	1.617	0.00	101	285.73	0/10	Best median LCP + zero TBT. Cold-start on serverless mobile; addressable via pre-warming strategies.
2	SvelteKit SPA	1.606	19.81	79	243.54	8/10	Smallest bundle (-39%) + fastest JMeter. No cold-start. Homepage TBT from animation, not architecture.
3	React static-MPA	2.051	0.44	80	401.80	6/10	SSR LCP benefit confirmed ($p = 0.007$ vs React SPA). No cold-start. Best for React ecosystem teams.
4	Nuxt SSR-MPA	2.323	1.14	164	267.38	0/10	Best repeat-visit cache benefit (LCP -51%, $p = 0.001$). Consistently slowest JMeter. Mobile cold-start.
5	React SPA	2.209	0.17	108	399.67	8/10	Adequate on all metrics; no standout advantage. Large bundle constrains low-bandwidth robustness.
6	Nuxt SPA	2.465	8.66	88	335.45	2/10	TBT 554 ms Slow 4G (Needs Improvement). No cache benefit. Lowest overall robustness for global deployment.

Narrative synthesis, not a Formal MCDA model. Based on single implementations/runs per combination; indicative, not confidence-interval estimates; read alongside Table 3 and Table 7's significance tests.

Integrating all measurement dimensions, and noting explicitly that SvelteKit SSR-MPA recorded 0 of 10 warm mobile runs (Section 3.2), SvelteKit SSR-MPA is recommended as the primary choice only when this SSR cold-start risk can be mitigated through deployment configuration, for a system of comparable scale and structure to the studied case (RQ3). It achieves the best median LCP (1.118 s), zero TBT across all pages, competitive load-test performance (101 ms at 100 concurrent users), and the third-smallest bundle. Cold-start latency can be mitigated through keep-warm and pre-warming strategies [13];

in the present results, cold-start susceptibility under mobile serverless conditions therefore appears as a deployment-layer limitation rather than an intrinsic framework deficiency. This deployment-layer distinction is critical to the ranking rationale: SvelteKit SSR-MPA's 0/10 mobile warm-run rate in Table 7 reflects infrastructure configuration rather than framework capability, as its Tier 1 desktop performance: best median LCP (1.118 s), zero TBT, effect size $e2 = 0.299$ Large, remains unaffected by the serverless cold-start phenomenon [13]. SvelteKit SSR-MPA's controlled desktop advantage holds across both SPA and MPA configurations and withstands multi-condition stress testing, reinforcing its recommendation for deployment contexts prioritizing raw rendering performance.

SvelteKit SPA is recommended as the primary alternative for deployment contexts where SSR infrastructure management is not feasible, offering the smallest bundle and fastest load-test response without cold-start exposure. For institutions prioritizing low operational complexity, static deployment, and cold-start avoidance, SvelteKit SPA may be the more conservative production choice despite SvelteKit SSR-MPA's superior controlled desktop metrics. React static-MPA is the recommended choice for teams with existing React.js investment, providing statistically confirmed LCP improvement over React SPA ($p = 0.007$) without mobile cold-start risk, since its pages are served as static assets rather than through serverless functions. Nuxt SPA is explicitly not recommended for globally accessible academic platforms: its Needs Improvement TBT under Slow 4G and absent repeat-visit cache effectiveness constitute critical accessibility deficiencies for international, bandwidth-diverse audiences. This conclusion reflects the present study's Slow 4G results and is consistent with prior observations that large JavaScript payloads particularly penalize users on slower network connections [2], [7]. Developers committed to the Nuxt.js ecosystem who must serve bandwidth-constrained audiences should consider the MPA configuration instead, which demonstrated a 51% LCP improvement on repeat visits ($p = 0.001$) and benefits from server-side CSS extraction reducing total bundle size to 267.38 KB. The findings collectively respond to the standardized-evaluation-protocol gap noted by Ghattas et al. [10] by providing a controlled, cross-architecture CWV evidence base for academic IS framework selection.

For institutions deploying comparable academic conference systems, these findings suggest: (a) preferring static hosting where per-request personalization is not required,

avoiding serverless cold-start exposure entirely; (b) where SSR is required, using scheduled pre-warming pings or provisioned (non-serverless) compute; (c) setting explicit CDN cache-control headers on static bundle assets, particularly for implementations such as Nuxt SPA that showed no significant repeat-visit benefit; (d) applying bundle/code-splitting for JavaScript-heavy SPA builds to reduce Slow 4G download latency; and (e) deferring non-essential animation components, such as this study's homepage particle animation, until after first interaction.

3.7 Discussion

The RQ1 finding that framework choice, more than the SPA-versus-MPA distinction, drives the observed CWV differences is consistent with Zaidan and Suyatno [9], who found that framework identity (Astro JS leading most Lighthouse and PageSpeed Web Vitals metrics, Nuxt JS leading JMeter stability metrics) accounted for most of the observed performance variation, without decomposing rendering strategy as a separate factor. This study extends that account in two respects. First, the architectural effect itself proved framework-dependent rather than a fixed, generalizable property: the SPA-versus-MPA distinction reached significance only within React.js, a nuance not addressed by single-condition benchmarks such as Ollila et al. [8] and Putra et al. [7]. Second, by varying framework while holding architecture constant, and varying architecture while holding framework constant, this design isolates the two factors more directly than prior single-configuration studies such as Ollila et al. [8].

These results also close a specific gap in the comparative literature. Siahaan and Kenidy [11] compared React, Vue, Next.js, and Nuxt.js but excluded Svelte from their four-way rendering comparison; including SvelteKit here shows it outperforming both Nuxt.js configurations evaluated in the present study. Putra et al.'s [7] compile-time-architecture explanation, developed at the build and bundle level, is corroborated here with deployed, user-observable CWV rather than static build metrics alone. The absence of a significant SPA-versus-MPA difference within Nuxt.js specifically suggests, in turn, that framework-intrinsic characteristics can moderate the effect of rendering strategy: Vue.js's own runtime overhead appears sufficient to absorb an effect that is significant within React.js.

The mobile cold-start penalty for SvelteKit SSR-MPA and Nuxt SSR-MPA reproduces, in a web-performance context, the latency degradation the serverless-computing literature

attributes to cold-start invocation overhead. Golec et al. [13] characterize cold-start primarily as a platform-level phenomenon tied to function initialization on first or infrequent invocation, although its magnitude is also shaped by application factors such as deployment package size, dependencies, and runtime language; the present results are consistent with the platform-level component of this account, since the same rendering logic showed no equivalent penalty when served as a static asset rather than through a serverless function. This indicates that the cold-start literature's infrastructure-level explanation transfers directly to the web-performance domain rather than requiring a separate, framework-specific account, and that the practical mitigation is a deployment-configuration matter, such as pre-warming or provisioned compute, rather than a reason to avoid the SSR architecture itself. Because CWV function as one of several search-ranking signals rather than a dominant one, as already noted in the Introduction, this penalty is most consequential for exactly the kind of low-traffic, bursty deployment profile modeled here, where incidental request volume is unlikely to keep a serverless function warm.

A further interpretive point concerns how MPA itself was operationalized in this design: React static-MPA was implemented as a static build, while SvelteKit SSR-MPA and Nuxt SSR-MPA relied on server-side rendering per request. This heterogeneity means the cold-start penalty above is properly attributable to the SSR execution model, not to the MPA category as such, a distinction that prior comparative work treating MPA as a single, uniform architectural category [2] does not draw out. Future comparative designs would benefit from separating page-delivery architecture, SPA versus MPA, from rendering strategy, static versus SSR, as independent factors, which also motivates the static-site-generation extension proposed in the Conclusion.

Taken together, these findings address the three gaps identified in the Introduction. Gap 1, the absence of standardized cross-architecture benchmarking, is addressed by evaluating six framework-architecture combinations under one shared measurement protocol, directly responding to the standardized-protocol gap Ghattas et al. [10] documented across website performance evaluation more broadly. Gap 2, framework comparisons that conflate rendering mode with architecture, is addressed by the confound analysis above. Gap 3, limited exploration of academic information systems as an evaluation domain, is addressed by conducting the benchmark on a deployed academic

conference website: the resulting variability across framework-architecture combinations, observed while holding content, platform, and infrastructure constant, suggests that at least part of the cross-institutional performance variability Al Khuwaitem and Hasan [12] documented across 485 university websites may originate from underlying technology choices rather than operational differences alone. Separately, the Nuxt SPA weaknesses reported across the Tier 2 network, bundle-size, and caching evidence, its Slow 4G interactivity result, comparatively large bundle, and absent repeat-visit cache benefit, converge on a single architectural explanation rather than three unrelated findings: that client-side-rendered architectures shipping larger JavaScript payloads are disadvantaged under bandwidth-constrained conditions, a premise noted in prior work [2], [7] and supported here by deployment-specific, multi-measurement evidence for Nuxt.js.

4. CONCLUSION

This study provides a laboratory benchmark of six framework-architecture implementations of React.js, SvelteKit, and Nuxt.js on a single reconstructed academic conference website. RQ1 is answered, as framework-architecture choice produces statistically significant differences across all five laboratory performance metrics under controlled desktop conditions, with SvelteKit's compile-time architecture achieving the strongest LCP results in both SPA and MPA form. RQ2 is answered, as these differences shift substantially under mobile, Slow 4G, and repeat-visit conditions, most notably through the serverless cold-start penalty observed for SSR-based MPA implementations and the Slow 4G interactivity weakness observed for Nuxt SPA. RQ3 is answered through an evidence-based narrative synthesis, not a formal multi-criteria decision model, in that SvelteKit's architecture is associated with the strongest laboratory performance in this case, in both SPA and MPA form, though the SPA-versus-MPA distinction itself was performance-significant only within React.js under desktop conditions. Based on this integrated evidence, SvelteKit SSR-MPA is a conditional recommendation, applicable only when serverless cold-start risk is mitigated through pre-warming or provisioned compute; SvelteKit SPA is the safer static-deployment alternative where such mitigation is not feasible; React static-MPA suits teams with existing React.js investment; and Nuxt SPA warrants caution for bandwidth-diverse, globally accessed academic platforms given its Slow 4G interactivity result and absent repeat-visit cache benefit. These rankings are

conditional on the deployment context described and are not intended as a universal ordering of the three frameworks, nor as a claim that applies to academic conference information systems in general beyond the single reconstructed case studied here. These findings are further bounded by the Netlify Free-tier (CMH, Ohio, US East) deployment environment, which may not generalize to provisioned or edge-warmed infrastructure; the laboratory CWV setting, which does not capture real-user field variability; the TBT-based INP proxy, adopted because CrUX field data were unavailable; the fixed, non-randomized execution order of Tier 1 runs; the single implementation built per framework-architecture combination, meaning the reported differences reflect these specific implementations rather than intrinsic, implementation-independent framework properties; and an intermittent slow-loading pattern observed on select pages, notably React SPA Homepage, that is not attributable to serverless cold-start and whose source was not established within the scope of this study. Future research should evaluate edge-deployment configurations with provisioned or pre-warmed serverless instances to test whether the mobile cold-start penalty can be eliminated, extend the comparison to a static-site-generation (SSG) MPA variant for SvelteKit and Nuxt.js to isolate rendering mode from architecture more fully, incorporate real-user monitoring and CrUX-eligible traffic once the deployed systems accumulate sufficient visits, replicate the design across additional academic website types and additional frameworks such as Astro and Next.js, and evaluate multiple independent implementations per framework-architecture combination to separate framework effects from implementation-specific coding choices.

REFERENCES

- [1] R. R. Gaddam, "Optimizing Core Web Vitals: A comprehensive framework for enhanced digital performance," *Sarcouncil J. Eng. Comput. Sci.*, vol. 4, no. 7, pp. 704–711, 2025, doi: 10.5281/zenodo.15969506.
- [2] K. Kowalczyk and T. Szandala, "Enhancing SEO in single-page web applications in contrast with multi-page applications," *IEEE Access*, vol. 12, pp. 11597–11614, 2024, doi: 10.1109/ACCESS.2024.3355740.
- [3] D. Klochkov and J. Mulawka, "Improving Ruby on Rails-based web application performance," *Information*, vol. 12, no. 8, p. 319, Aug. 2021, doi: 10.3390/inFo12080319.

- [4] T. F. Iskandar, M. Lubis, T. F. Kusumasari, and A. R. Lubis, "Comparison between client-side and server-side rendering in the web development," in *IOP Conf. Ser.: Mater. Sci. Eng.*, vol. 801, Institute of Physics Publishing, Jun. 2020, p. 012136. doi: 10.1088/1757-899X/801/1/012136.
- [5] M. Savenko and K. Babii, "Performance optimization strategies for large-scale web applications using Next.js," *IEEE Access*, vol. 13, pp. 217376–217386, Dec. 2025, doi: 10.1109/ACCESS.2025.3647563.
- [6] V. Komperla, P. Deenadhayalan, P. Ghuli, and R. Pattar, "React: A detailed survey," *Indones. J. Electr. Eng. Comput. Sci.*, vol. 26, no. 3, pp. 1710–1717, Jun. 2022, doi: 10.11591/ijeecs.v26.i3.pp1710-1717.
- [7] F. P. E. Putra, Hasbullah, F. Muslim, and R. Paradina, "Technical performance comparison of modern frontend frameworks study on Svelte, React, and Vue," *Brilliance: Res. Artif. Intell.*, vol. 5, no. 1, pp. 355–364, Jul. 2025, doi: 10.47709/brilliance.v5i1.6133.
- [8] R. Ollila, N. Mäkitalo, and T. Mikkonen, "Modern web frameworks: A comparison of rendering performance," *J. Web Eng.*, vol. 21, no. 3, pp. 789–814, 2022, doi: 10.13052/jwe1540-9589.21311.
- [9] A. J. Zaidan and D. F. Suyatno, "Rendering performance analysis of Astro JS, Next JS, Nuxt JS, and SvelteKit frameworks using Google Lighthouse, PageSpeed Insight, and JMeter," *Journal of Emerging Information System and Business Intelligence*, vol. 6, no. 1, pp. 1–13, 2025, doi: 10.26740/jeisbi.v6i1.64283.
- [10] M. Ghattas, S. Odeh, and A. M. Mora, "Predicting website performance: A systematic review of metrics, methods, and research gaps (2010–2024)," *Computers*, vol. 14, no. 10, p. 446, Oct. 2025, doi: 10.3390/computers14100446.
- [11] M. Siahaan and R. Kenidy, "Rendering performance comparison of React, Vue, Next, and Nuxt," *Jurnal Mantik*, vol. 7, no. 3, pp. 1851–1860, 2023, doi: 10.35335/mantik.v7i3.4242.
- [12] M. Al Khuwaitem and L. Hasan, "Clustering university websites based on usability metrics: Insights from web vitals analysis," *Soc. Sci. Humanit. Open*, vol. 13, p. 102608, Jun. 2026, doi: 10.1016/j.ssaho.2026.102608.
- [13] M. Golec, G. K. Walia, M. Kumar, F. Cuadrado, S. S. Gill, and S. Uhlig, "Cold start latency in serverless computing: A systematic review, taxonomy, and future directions," *ACM Comput. Surv.*, vol. 57, no. 3, p. 65, Nov. 2024, doi: 10.1145/3700875.

- [14] J. Vepsäläinen, A. Hellas, and P. Vuorimaa, "Implications of edge computing for static site generation," in *Int. Conf. Web Inf. Syst. Technol. (WEBIST)*, Science and Technology Publications, Lda, 2023, pp. 223–231. doi: 10.5220/0012173900003584.
- [15] V. Jain, "Web Vitals and core metrics for web performance optimization," *Int. J. Core Eng. Manag.*, vol. 7, no. 6, pp. 198–205, 2023, doi: 10.5281/zenodo.14915137.
- [16] P. Podhorský, V. Kunštnár, and K. Jelemenská, "Architectural possibilities of application development in the environment of web technologies," in *2025 5th Int. Conf. Electr., Comput., Commun. Mechatronics Eng.*, 2025, pp. 1–6. doi: 10.1109/ICECCME64568.2025.11277664.
- [17] Indrianto, "Performance testing on web information system using Apache JMeter and Blazemeter," *Jurnal Ilmiah Ilmu Terapan Universitas Jambi*, vol. 7, no. 2, pp. 138–149, Dec. 2023, doi: 10.22437/jiituj.v7i2.28440.
- [18] A. R. Annisa, M. Ardiana, P. N. Rahayu, T. Brian, and M. A. Jami'in, "Performance evaluation of web applications using JMeter load testing for server capacity and response efficiency," *J. Appl. Informat. Comput.*, vol. 10, no. 1, pp. 585–590, 2026, doi: 10.30871/jaic.v10i1.11840.
- [19] T. Heričko, B. Šumak, and S. Brdnik, "Towards representative web performance measurements with Google Lighthouse," in *Proc. 2021 7th Student Comput. Sci. Res. Conf. (StuCoSReC)*, University of Maribor Press, Sep. 2021, pp. 39–42. doi: 10.18690/978-961-286-516-0.9.
- [20] A. C. Macunluoğlu and G. Ocakoğlu, "Comparison of the performances of non-parametric k-sample test procedures as an alternative to one-way analysis of variance," *Eur. Res. J.*, vol. 9, no. 4, pp. 687–696, Jul. 2023, doi: 10.18621/eurj.1037546.
- [21] F. Fiel Peres, "Effect sizes for nonparametric tests," *Biochem. Med. (Zagreb)*, vol. 36, no. 1, p. 010101, Feb. 2026, doi: 10.11613/BM.2026.010101.
- [22] U. Singh, J. D. Venkatesh, A. Muraleedharan, K. S. Saluja, J. H. Anamika, and P. Biswas, "Accessibility analysis of educational websites using WCAG 2.0," *Digit. Gov.: Res. Pract.*, vol. 5, no. 3, p. 32, Oct. 2024, doi: 10.1145/3696318.
- [23] I. Malavolta, K. Chinnappan, L. Jasmontas, S. Gupta, and K. A. K. Soltany, "Evaluating the impact of caching on the energy consumption and performance of progressive web apps," in *2020 IEEE/ACM 7th Int. Conf. Mobile Softw. Eng. Syst.*, Association for Computing Machinery, Inc, Jul. 2020, pp. 109–119. doi: 10.1145/3387905.3388593.

- [24] S. S. Raweyai and I. R. Widiyari, "Performance testing of academic website using load testing method supported by Apache JMeterTM at XYZ University," *Jurnal Teknik Informatika (Jutif)*, vol. 5, no. 3, pp. 721–730, May 2024, doi: 10.52436/1.jutif.2024.5.3.1796.
- [25] H. O. Ekpobimi, R. C. Kandeke, and A. A. Fasanmade, "Conceptual framework for enhancing front-end web performance: Strategies and best practices," *Glob. J. Adv. Res. Rev.*, vol. 2, no. 1, pp. 099–107, Sep. 2024, doi: 10.58175/gjarr.2024.2.1.0032.