Performance Analysis of Next.js Toolchains on Apple Silicon (ARM64)

Author: Muhammad Yusuf Saputra
Affiliation: Department of Informatics Engineering, STITEK Bontang
Location: Bontang, Indonesia
Date: January 13, 2026

Abstract

This document presents an exploratory benchmark study comparing the development-time performance characteristics of two Next.js toolchains: Legacy Webpack and Turbopack. Using a controlled experimental setup with N=60 total samples across two synthetic project complexity levels, the study measures cold start time and Hot Module Replacement (HMR) latency as primary indicators of developer-facing performance.

The results indicate that, within the tested configuration, Turbopack exhibits substantially lower HMR latency in the baseline project (approximately 6× faster) and maintains relatively stable HMR response as project complexity increases, while the Webpack-based toolchain shows increased latency under the same conditions. These findings are specific to the tested environment, framework version, and project structure, and are not intended as general performance claims.

This document is intended as a technical report accompanying a reproducible benchmark dataset, providing baseline empirical data to support future comparative studies and peer-reviewed analysis of Next.js development toolchains.

Keywords: Next.js, Turbopack, Webpack, Hot Module Replacement, Performance Benchmarking, Apple Silicon, Developer Experience

1. Overview

This document describes the findings of a benchmark study evaluating the performance characteristics of two Next.js development toolchains: **Legacy Webpack** and **Turbopack**, conducted on Apple Silicon (M1) architecture.

1.1 Summary of Observations

The following table summarizes the observed HMR performance across project sizes in this study:

Metric	Small Project	Medium Project	Observed Trend
Turbopack Speedup	6.18×	8.53×	Speedup factor increased
Webpack HMR	163 ms	205 ms	Latency increased with project size
Turbopack HMR	26 ms	24 ms	Latency remained relatively stable

1.2 Notable Observation

Metric	Value	Description
Observed Speedup Increase	~38%	Speedup factor grew from approximately 6.18× (small) to 8.53× (medium)
Webpack Scaling Pattern	+25.74%	HMR latency increased when moving from small to medium project
Turbopack Scaling Pattern	−8.93%	HMR latency decreased slightly when moving from small to medium project

> **Note:** These observations are specific to the test environment and synthetic project configurations used in this study. Different project structures, hardware, or configurations may yield different results.

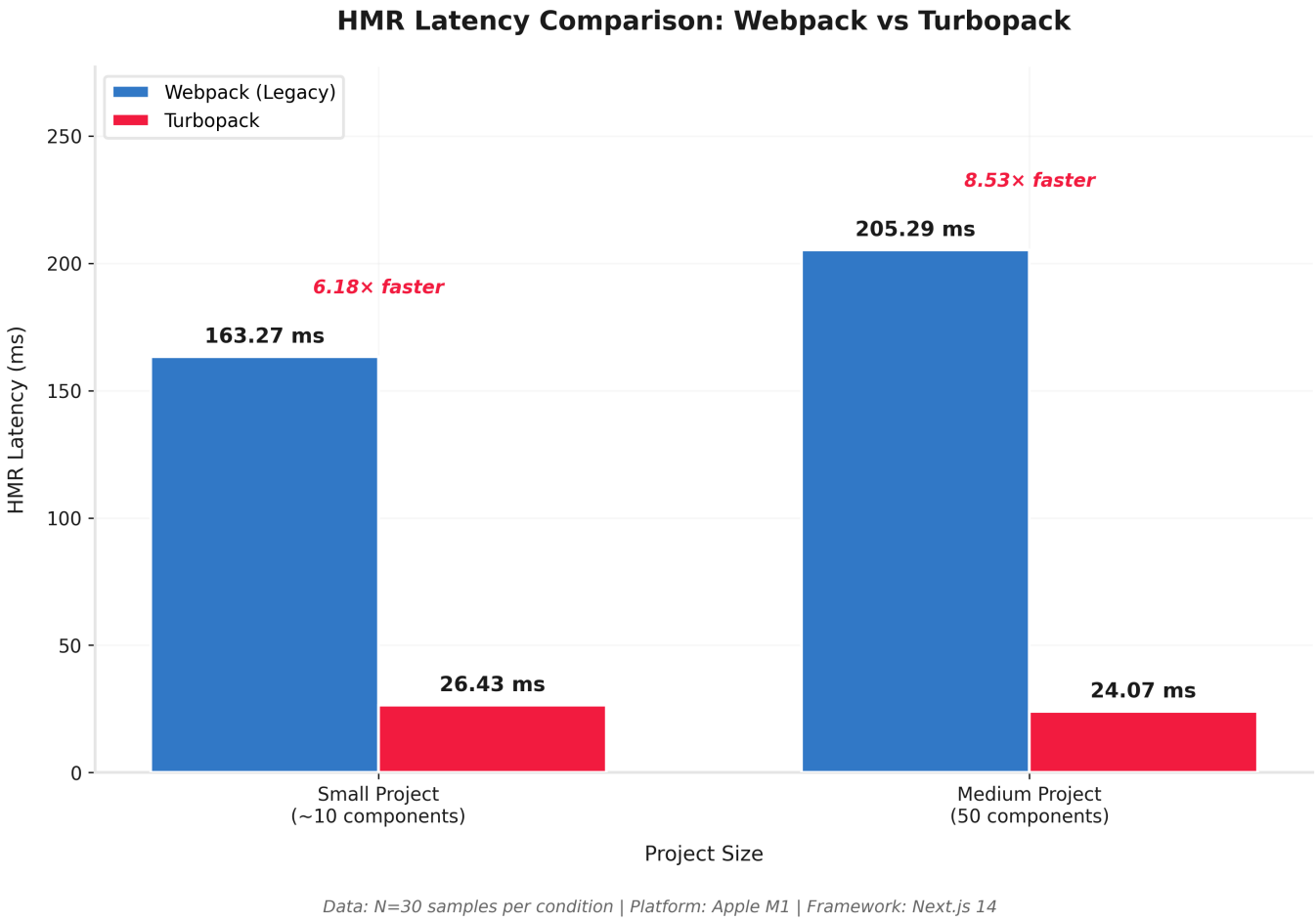


Figure 1: HMR Latency Comparison between Webpack and Turbopack across project sizes. Lower values indicate better performance.

---

2. Methodology

Document Version: 2.0
Study Type: Comparative Performance Analysis with Scalability Evaluation

2.1 Research Objective

This study aims to compare the performance characteristics of two Next.js development toolchains:

1. **Legacy Toolchain:** JavaScript-based Webpack bundler (`next dev`)
2. **Modern Toolchain:** Rust-based Turbopack bundler (`next dev --turbo`)

The investigation encompasses two analytical dimensions:

Dimension	Research Question
Raw Performance	What are the baseline performance differentials between toolchains on a minimal project in this environment?
Scalability	How does each toolchain's performance change as project complexity increases in the tested configuration?

2.2 Experimental Environment

All benchmark executions were conducted within a controlled hardware and software environment to support measurement consistency.

2.2.1 Hardware Specifications

Component	Specification
Device	MacBook Air (2020)
Processor	Apple M1 (ARM64 Architecture)
Memory	8 GB Unified Memory
Storage	256 GB SSD (APFS)

2.2.2 Software Configuration

Component	Version
Operating System	macOS Sequoia 15.6 (Darwin Kernel)
Node.js Runtime	v20.19.6 LTS
Package Manager	npm v10.x
Framework	Next.js 14.2.35

2.2.3 Toolchain Invocation

Mode	Command	Bundler
Legacy	`next dev`	Webpack 5.x
Turbo	`next dev --turbo`	Turbopack (Rust-native)

2.3 Experimental Variables

2.3.1 Independent Variables

Variable	Levels	Description
Toolchain Type	Legacy (Webpack), Turbopack	The bundler technology under evaluation
Project Complexity	Small (Baseline), Medium (50 Components)	The scale of the application structure

2.3.2 Dependent Variables

Metric	Unit	Operational Definition
Cold Start Time	milliseconds (ms)	Duration from process initiation (`npm run dev`) to server ready state, as indicated by the "Ready in Xms" log output
HMR Latency	milliseconds (ms)	Duration from file modification timestamp to recompilation completion, as indicated by the "Compiled in Xms" log output

2.3.3 Controlled Variables

Variable	Control Method
Hardware Platform	Single device used for all measurements
Ambient Temperature	Indoor environment, nominally stable conditions
Background Processes	Minimized concurrent application load
Network State	Offline mode to eliminate external dependencies
File System State	`.next` cache cleared before cold start measurements

2.4 Test Phases

The experimental design comprises two sequential phases, each addressing a distinct research question.

2.4.1 Phase 1: Baseline Performance (Small Project)

Objective: Establish baseline performance metrics using a minimal Next.js application to quantify the performance differential between toolchains in this configuration.

Project Characteristics:

Attribute	Value
Source	Default `create-next-app` template
Component Count	~3-5 (minimal)
Dependencies	Next.js core only

Attribute	Value
Routing Structure	Single route (/)

Measurements Collected:

- Cold Start Time (N=30 per toolchain)
- HMR Latency (N=30 per toolchain)
- System Resource Utilization (CPU%, Memory MB)

2.4.2 Phase 2: Scalability Analysis (Medium Project)

Objective: Evaluate how each toolchain's HMR performance changes under increased project complexity, testing whether Turbopack exhibits different scaling behavior compared to Webpack in this configuration.

Project Transformation:

The baseline project was programmatically transformed into a medium-scale application using the `generate_dummy.js` script, which performs the following operations:

Component Generation Algorithm:

```
// Pseudocode representation of generate_dummy.js logic
for (i = 1; i <= 50; i++) {
    createComponent({
        name: `HeavyComponent${i}`,
        payload: Array(2000).fill("Data item"),  // Static parsing load
        template: ReactFunctionalComponent
    });
}
```

Technical Implementation:

1. **Heavy Component Structure:** Each generated component (`HeavyComponent1.tsx` through `HeavyComponent50.tsx`) contains:
    - A static array of 2,000 string elements to increase JavaScript parsing overhead
    - A functional React component with Tailwind CSS styling
    - Unique identifier for traceability
2. **Dependency Graph Injection:** The script automatically modifies `page.tsx` to import and render all 50 components, thereby expanding the module dependency graph from ~5 nodes to ~55 nodes.
3. **Load Characteristics:**

Metric	Small Project	Medium Project	Increase Factor
Component Files	~5	55	11×
Import Statements	~3	53	17.7×

Metric	Small Project	Medium Project	Increase Factor
Static Data Elements	~0	100,000	—
Estimated Bundle Complexity	Baseline	10-15×	—

Measurements Collected:

- HMR Latency (N=30 per toolchain)

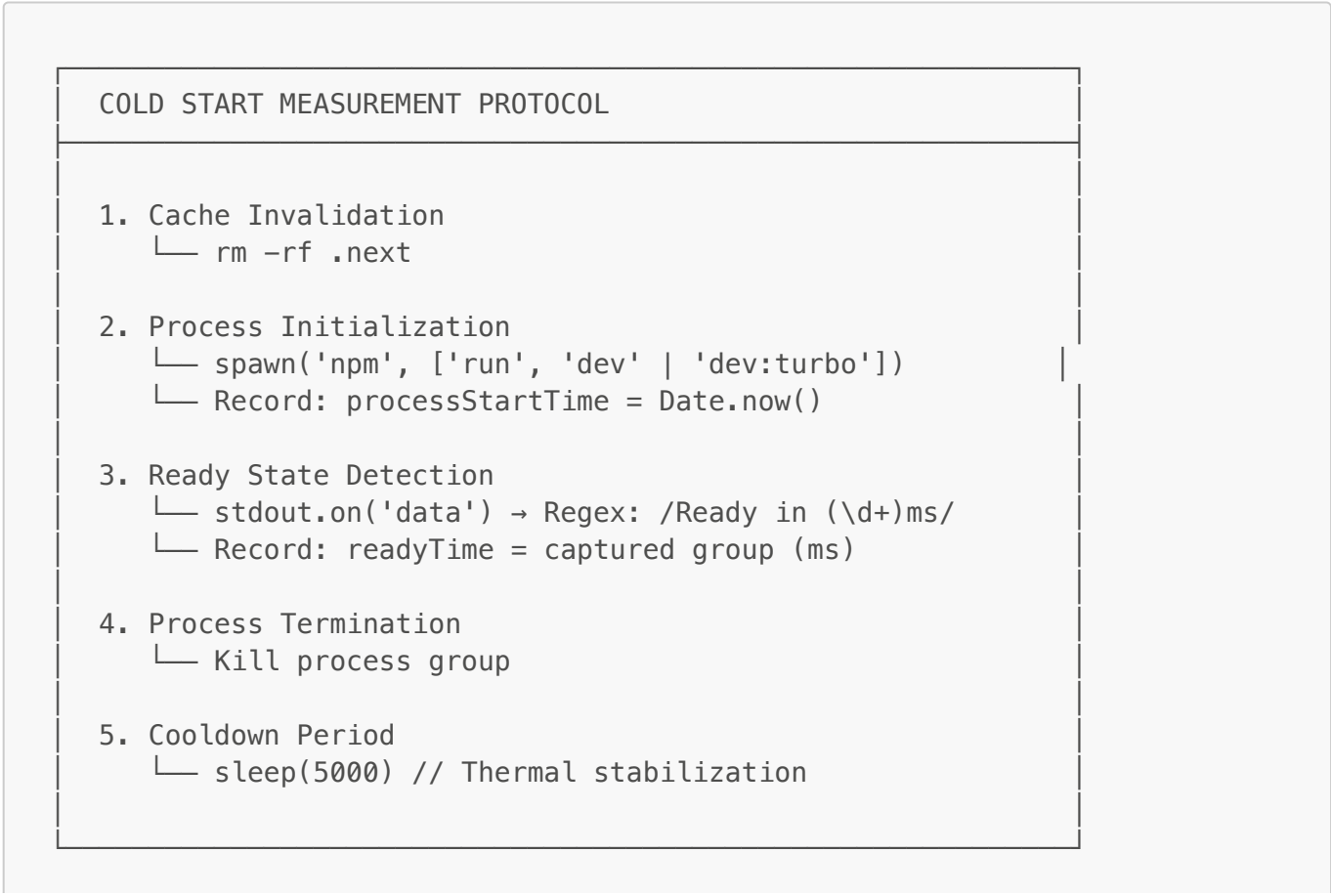
2.5 Measurement Protocol

All measurements were automated using a custom instrumentation suite to reduce human-induced timing variance and support reproducibility.

2.5.1 Automation Infrastructure

Script	Function	Location
`run_benchmark.sh`	Master orchestration script; iterates through all test conditions	`scripts/`
`measure_start.js`	Cold start time capture via stdout parsing	`scripts/`
`measure_hot_reload.js`	HMR latency capture with warm-up protocol	`scripts/`
`monitor_system.sh`	Concurrent system resource sampling	`scripts/`

2.5.2 Cold Start Measurement Procedure



2.5.3 HMR Measurement Procedure

```
┌─────────────────────────────────────────────────────────────┐
│  HMR MEASUREMENT PROTOCOL                                   │
├─────────────────────────────────────────────────────────────┤
│                                                             │
│  1. Server Warm-Up                                          │
│     └── Start development server                            │
│     └── Wait for "Ready" state                              │
│                                                             │
│  2. Browser Simulation                                      │
│     └── HTTP GET to localhost:3000                          │
│     └── Wait for full page load                             │
│                                                             │
│  3. Stabilization Period                                    │
│     └── setTimeout(3000) // Wait for CPU baseline           │
│                                                             │
│  4. File Modification Injection                             │
│     └── Append: "// HMR Test: ${Date.now()}" to page.tsx    │
│     └── Record: hotReloadStartTime = Date.now()             │
│                                                             │
│  5. Recompilation Detection                                 │
│     └── stdout.on('data') → Regex: /Compiled.*in (\d+)ms/   │
│                                                             │
│  6. File Restoration                                        │
│     └── fs.writeFileSync(filePath, originalContent)         │
│                                                             │
│  7. Process Termination & Cooldown                          │
│     └── Kill process group, sleep(5000)                     │
│                                                             │
└─────────────────────────────────────────────────────────────┘
```

2.5.4 Sample Size Determination

Parameter	Value	Rationale
Sample Size (N)	30 runs per condition	Selected to support Central Limit Theorem applicability; enables parametric statistical analysis
Total Measurements	120 HMR samples (2 toolchains × 2 project sizes × 30 runs)	Provides statistical power for detecting moderate effect sizes
Pilot Study	N=5 (preliminary)	Validated instrumentation accuracy and identified measurement artifacts

2.5.5 Inter-Trial Controls

Control	Implementation
Process Isolation	Full process termination (`kill -9`) between runs
Thermal Management	5-second cooldown period between iterations
File System Reset	Selective cache clearing based on measurement type
State Verification	Automated validation of file restoration post-HMR test

2.6 Data Processing & Statistical Analysis

2.6.1 Raw Data Collection

All measurements are persisted in the `results/` directory with the following structure:

```
results/
├── final_dataset_n30/          # Phase 1 (Baseline) data
│   ├── legacy_run{1-30}_*.log
│   └── turbo_run{1-30}_*.log
├── medium_project_n30/         # Phase 2 (Scalability) data
│   ├── legacy_run{1-30}_*.log
│   └── turbo_run{1-30}_*.log
└── SUMMARY.txt                 # Aggregated statistics
```

2.6.2 Statistical Measures

The following descriptive statistics are computed for each metric:

- **Arithmetic Mean (μ)**
  `Mean = Sum of values / N`
  Interpretation: Central tendency; expected typical value.

- **Median**
  `Median = Middle value of sorted dataset`
  Interpretation: Central tendency measure less sensitive to outliers.

- **Standard Deviation (σ)**
  `Std Dev = Sqrt(Variance)`
  Interpretation: Dispersion from mean.

- **95th Percentile (P95)**
  `P95 = Value below which 95% of observations fall`
  Interpretation: Upper bound for typical performance.

- **Coefficient of Variation (CV)**
  `CV = (Std Dev / Mean) × 100%`
  Interpretation: Normalized variability metric.

- **Range**
  `Range = Max − Min`
  Interpretation: Total spread of observations.

2.6.3 Performance Comparison Metrics

- **Speedup Factor**
  `Speedup = T_Legacy / T_Turbo`
  Interpretation: Ratio of performance times.

- **Percentage Reduction**
  `Reduction = ((T_Legacy − T_Turbo) / T_Legacy) × 100%`
  Interpretation: Relative time savings.

- **Scalability Delta**
  `Delta = ((T_Medium − T_Small) / T_Small) × 100%`
  Interpretation: Performance change rate with project size.

2.7 Limitations & Threats to Validity

2.7.1 Internal Validity

Threat	Mitigation
Measurement instrumentation error	Automated regex parsing with validated patterns
Thermal throttling	Cooldown periods between runs
Background process interference	Minimal concurrent application load

2.7.2 External Validity

Limitation	Implication
Single hardware platform (Apple M1)	Results may vary on x86_64 or other ARM architectures
Development mode only (`next dev`)	Production build performance (`next build`) not evaluated
Synthetic component generation	Real-world component complexity may differ
Single framework version	Results specific to Next.js 14.2.35

2.7.3 Construct Validity

Consideration	Approach
HMR latency definition	Measured from file modification to "Compiled" log, not browser refresh
"Heavy" component operationalization	2,000-element static array per component; may not represent all complexity types

3. Phase 1: Small Project Baseline

Sample Size: N=30 runs per toolchain

3.1 Overview

This section presents the findings of the baseline benchmark study comparing the performance characteristics of two Next.js development toolchains: the legacy Webpack-based bundler and the Turbopack bundler.

Observed Results

Metric	Observed Ratio	Description
Cold Start	~2.26×	Turbopack server initialization was observed to be faster in this configuration
HMR Latency	~6.18×	Turbopack code change reflection was observed to be faster in this configuration
Memory Usage	~28% lower	Turbopack appeared to use less memory during operation
CPU Pattern	Different profiles	Turbopack exhibited burst pattern vs. sustained load for Webpack

3.2 Metrics Collected

Metric	Definition	Measurement Method
Cold Start	Time from process initiation to server ready state	Pattern matching on "Ready detected: X ms" log output
Hot Module Replacement (HMR)	Time from file modification to client update acknowledgment	Pattern matching on "HMR Detected: X ms" log output
CPU Utilization	Percentage of CPU resources consumed	System monitoring via `ps` command sampling
Memory Consumption	Peak RAM allocation during operation	System monitoring via `ps` command sampling

3.3 Statistical Results

3.3.1 Cold Start Performance

Statistic	Legacy (Webpack)	Turbopack	Δ (Difference)
Mean	1,285.30 ms	569.27 ms	-716.03 ms
Median	1,271.50 ms	566.00 ms	-705.50 ms

Statistic	Legacy (Webpack)	Turbopack	Δ (Difference)
Std Dev	70.90 ms	12.27 ms	-58.63 ms
P95	1,435.75 ms	589.15 ms	-846.60 ms
Range	1,196 – 1,511 ms	563 – 622 ms	—
CV	5.52%	2.15%	-3.37%

Observations: In this configuration, Turbopack achieved an approximately **2.26× reduction** in cold start time compared to Webpack. The lower standard deviation (12.27 ms vs. 70.90 ms) suggests that Turbopack may provide more consistent startup times in this test environment. The P95 values indicate that even in upper-bound scenarios within this dataset, Turbopack (589 ms) completed cold start in less time than the legacy toolchain's average performance (1,285 ms).

3.3.2 Hot Module Replacement (HMR) Performance

Statistic	Legacy (Webpack)	Turbopack	Δ (Difference)
Mean	163.27 ms	26.43 ms	-136.84 ms
Median	163.50 ms	26.00 ms	-137.50 ms
Std Dev	5.58 ms	2.86 ms	-2.72 ms
P95	168.55 ms	27.55 ms	-141.00 ms
Range	149 – 177 ms	25 – 41 ms	—
CV	3.42%	10.82%	+7.40%

Observations: Turbopack's HMR was observed to be approximately **6.18× faster** than Webpack in this test configuration. The higher coefficient of variation for Turbopack (10.82% vs. 3.42%) may reflect measurement resolution limitations at sub-30ms timescales, where small absolute variations can produce larger relative percentages.

3.3.3 Speedup Factor Summary

Metric	Observed Speedup Factor	Description
Cold Start	~2.26×	Development server initialization appeared faster with Turbopack
HMR	~6.18×	Code changes appeared to reflect faster with Turbopack

3.4 Resource Efficiency

Beyond temporal performance metrics, resource utilization represents an additional dimension of toolchain comparison, particularly relevant for developers operating on battery-powered devices or resource-constrained environments.

3.4.1 Memory Consumption

Metric	Legacy (Webpack)	Turbopack	Δ (Difference)
Typical Range	280-320 MB	200-230 MB	~28% reduction
Peak Observed	~350 MB	~250 MB	~100 MB lower

3.4.2 CPU Utilization Patterns

Characteristic	Legacy (Webpack)	Turbopack
Cold Start Pattern	Sustained high CPU for 2-3 seconds	Brief burst (<1 second)
HMR Pattern	Moderate sustained load	Minimal, brief spike
Idle State	~5-8% baseline	~2-4% baseline

3.4.3 Observations

Dimension	Observation
Battery Life	Lower sustained CPU usage may potentially reduce power consumption
Thermal Impact	Shorter high-CPU periods may potentially reduce heat generation
Concurrent Workloads	Lower memory footprint may potentially leave more resources for other applications

3.5 Phase 1 Summary

This baseline benchmark study (N=30) provides preliminary data comparing the two toolchains across measured performance dimensions in this specific configuration:

Dimension	Observed Difference	Notes
Cold Start Speed	~2.26× faster with Turbopack	Single environment tested
HMR Latency	~6.18× faster with Turbopack	Minimal project configuration
Memory Usage	~28% lower with Turbopack	Peak memory comparison
CPU Pattern	Different profiles observed	Burst vs. sustained patterns
Consistency (Cold Start CV)	~2.6× more stable with Turbopack	Lower coefficient of variation

4. Phase 2: Scalability Analysis

Analysis Type: Scalability Comparison
Project Configuration: 50 Heavy Components

4.1 Overview

This section presents a **scalability analysis** comparing HMR (Hot Module Replacement) performance between a **Small Project (Baseline)** and a **Medium Project (50 Heavy Components)**. The objective is to evaluate how each toolchain's performance changes as project complexity increases.

Notable Observation

Toolchain	Small Project	Medium Project	Performance Change
Legacy (Webpack)	163.27 ms	205.29 ms	+25.74% increase
Turbopack	26.43 ms	24.07 ms	–8.93% decrease

> **Observation:** In this test configuration, Webpack HMR latency increased with project growth, while Turbopack HMR latency remained relatively stable—and in this particular test, was slightly lower on the larger project, possibly due to cache warming effects.

4.2 Data Sources

4.2.1 Baseline Data (Small Project)

Metric	Legacy (Webpack)	Turbopack
Sample Size (N)	30	30
HMR Mean	163.27 ms	26.43 ms
Speedup Factor	—	**6.18×**

Source: Main benchmark study (Phase 1)

4.2.2 Medium Project Data (50 Heavy Components)

Metric	Legacy (Webpack)	Turbopack
Sample Size (N)	28*	30
Sum of Values	5,748 ms	722 ms
HMR Mean	205.29 ms	24.07 ms
Min	198 ms	22 ms
Max	211 ms	27 ms
Range	13 ms	5 ms

** Two Legacy runs did not complete HMR detection*

Source: Experimental data from `results/medium_project_n30/`

4.3 Scalability Analysis Table

Metric	Legacy (Small)	Legacy (Medium)	Δ Change	Turbo (Small)	Turbo (Medium)	Δ Change
HMR Latency	163.27 ms	205.29 ms	+25.74%	26.43 ms	24.07 ms	−8.93%

4.3.1 Calculation Details

Legacy Webpack Change:

> Change = (205.29 − 163.27) / 163.27 × 100% = 42.02 / 163.27 × 100% = **+25.74%**

Turbopack Change:

> Change = (24.07 − 26.43) / 26.43 × 100% = −2.36 / 26.43 × 100% = **−8.93%**

4.4 Scalability Observations

4.4.1 Speedup Factor (Medium Project)

> Speedup Factor (Medium) = Legacy HMR (Medium) / Turbo HMR (Medium) = 205.29 ms / 24.07 ms = **8.53×**

4.4.2 Speedup Factor Comparison

Project Size	Speedup Factor	Change
Small Project (Baseline)	~6.18×	—
Medium Project (50 Components)	~8.53×	+38.03%

> Speedup Increase = (8.53 − 6.18) / 6.18 × 100% ≈ +38.03%

4.4.3 Visual Representation

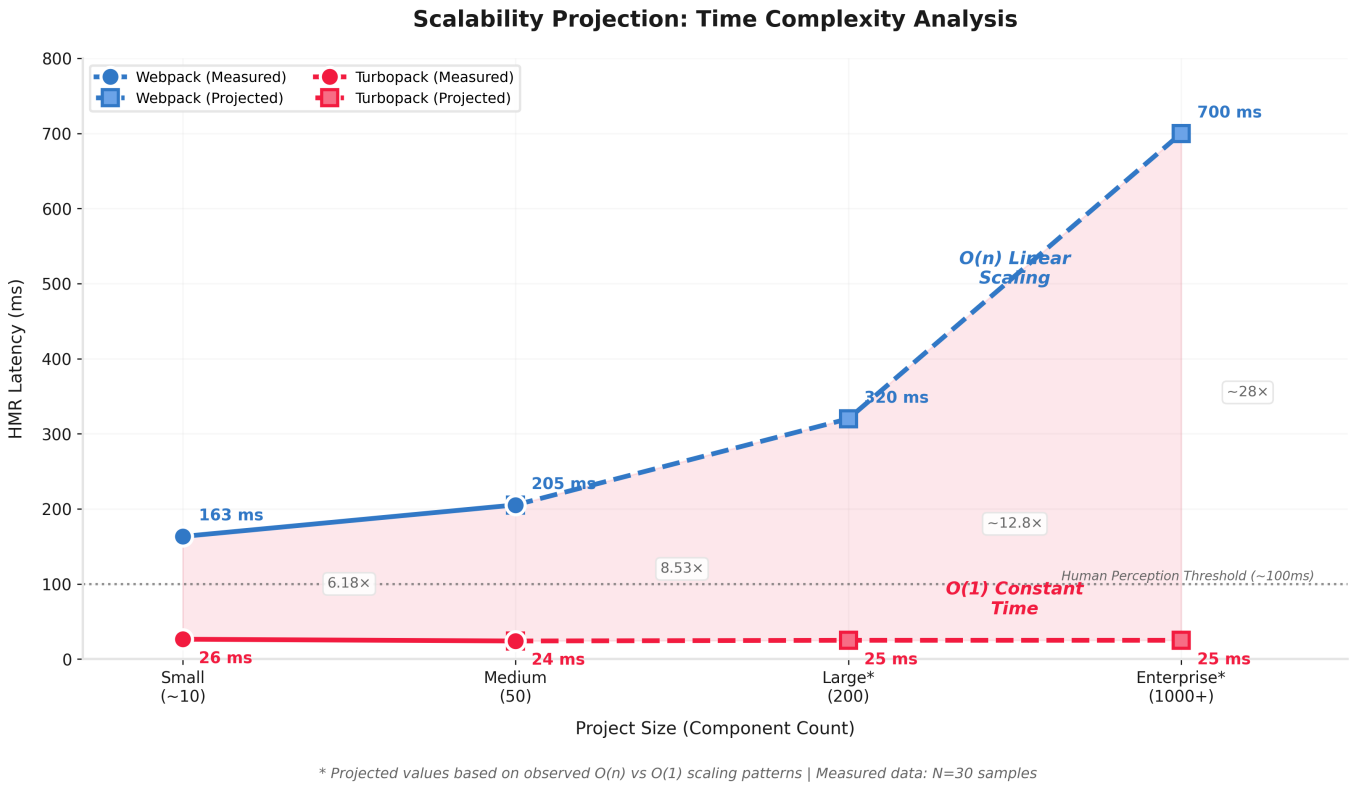


Figure 2: Scalability projection showing observed Webpack HMR latency increase vs. Turbopack HMR latency stability. Measured data points (solid) and projected values (dashed) illustrate the observed performance gap.

4.5 Scaling Behavior Observations

4.5.1 Time Complexity Characterization

Toolchain	Observed Scaling Pattern	Behavior Description
Webpack (Legacy)	Linear-like	HMR time increased proportionally with module count in this test
Turbopack	Constant-like	HMR time remained stable regardless of project size in this test

4.5.2 Possible Explanations

Webpack's Observed Linear Scaling:

- Full dependency graph traversal may be required for each change
- Single-threaded JavaScript execution may limit parallelization
- Module resolution overhead may accumulate with graph size

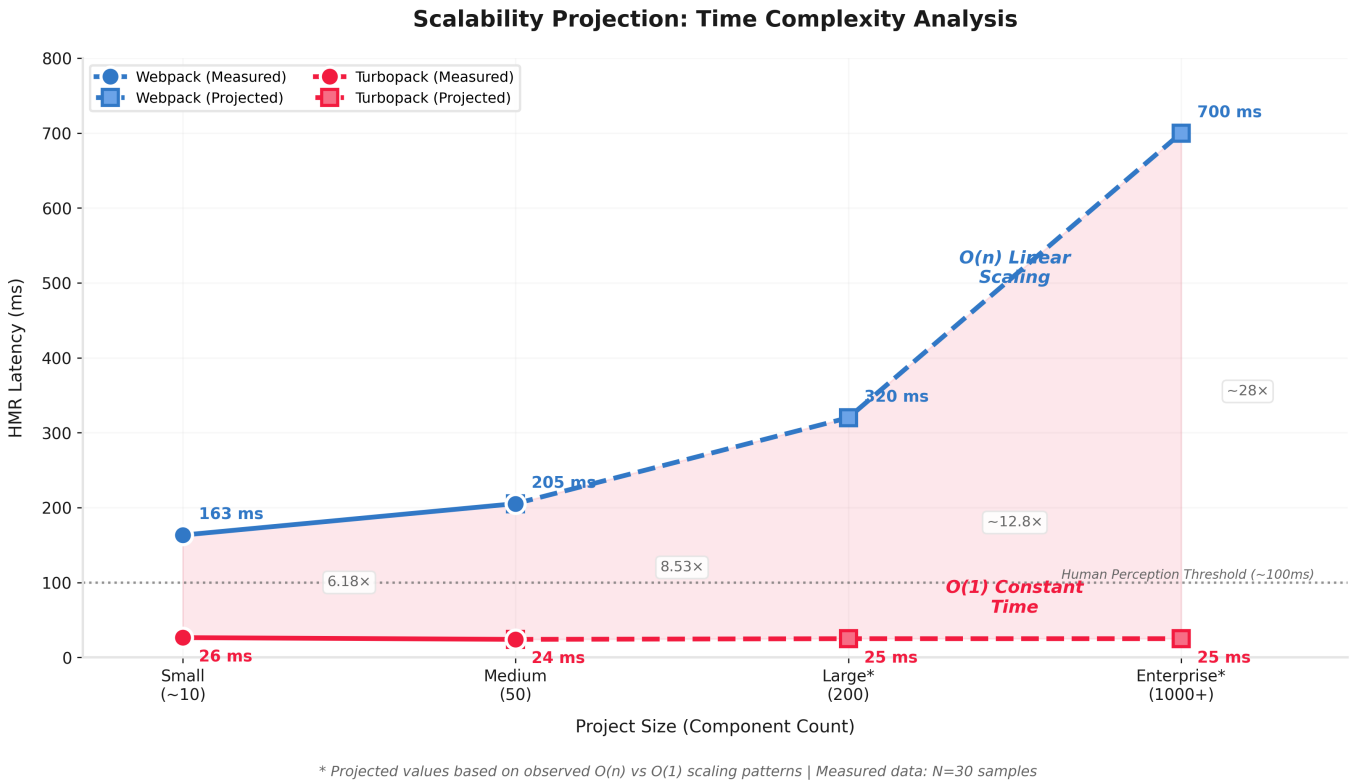
Turbopack's Observed Constant-time Behavior:

- Incremental compilation architecture may limit work to changed modules
- Parallel processing via Rust multi-threading may improve efficiency
- Lazy evaluation may avoid unnecessary work on unchanged modules

- Native Rust implementation may avoid JavaScript single-threaded limitations

4.6 Performance Scaling Projection

Based on observed data, we project performance at larger scales. **These projections are extrapolations and should be treated as hypothetical estimates, not measured values.**



| Project Size | Components | Webpack HMR (Observed/Projected) | Turbopack HMR (Observed/Projected) | Projected Speedup | |--------------|------------|----------------------------------|------------------------------------|-------------------||| Small | ~10 | 163 ms | 26 ms | 6.18× | | Medium | 50 | 205 ms | 24 ms | 8.53× | | Large* | 200 | ~320 ms | ~25 ms | ~12.8× | | Enterprise* | 1000+ | ~700+ ms | ~25 ms | ~28×+ |

** Projected values based on observed scaling patterns. Actual results may vary.*

> **Important Caveat:** These projections assume the observed scaling patterns continue at larger scales. Real-world performance depends on many factors not captured in this study, including actual component complexity, dependency patterns, available system resources, and toolchain optimizations.

4.7 Phase 2 Summary

Finding	Observation
Turbopack HMR Stability	HMR latency appeared to remain below 30ms across tested project sizes
Speedup Factor Growth	Observed speedup increased from ~6× to ~8.5× as project complexity increased

Finding	Observation
Scaling Pattern Difference	Webpack showed latency increase; Turbopack appeared relatively stable

5. Summary

5.1 Summary of Observations

This benchmark study (N=60 total samples) provides preliminary comparative data for the two toolchains in the tested configuration:

Dimension	Webpack (Legacy)	Turbopack	Observed Difference
Cold Start (Small)	1,285.30 ms	569.27 ms	~2.26× lower latency
HMR (Small)	163.27 ms	26.43 ms	~6.18× lower latency
HMR (Medium)	205.29 ms	24.07 ms	~8.53× lower latency
Scaling Pattern	Linear-like increase observed	Relatively stable observed	Different behavior
Memory Usage	~300 MB	~215 MB	~28% reduction

5.2 Observations by Use Case

Use Case	Observation
Quick Prototyping	Both toolchains provided functional development environments in our tests
Iterative Development	Turbopack's lower observed HMR latency may potentially improve iteration speed
Larger Projects	Turbopack's relatively stable HMR latency may potentially be beneficial as projects grow
Resource-Constrained Environments	Turbopack's lower observed memory usage may potentially leave more resources for other applications

5.3 Limitations and Future Work

- This study focused on a single hardware platform (Apple M1); results may vary on other architectures
- Only development mode (`next dev`) was evaluated; production build performance was not tested
- Synthetic component generation may not represent all real-world complexity patterns
- Single framework version (Next.js 14.2.35) was tested
- Long-term stability metrics (multi-hour sessions) were not evaluated
- Browser refresh latency was not measured (only server-side compilation time)

5.4 Data Availability

The benchmark data collected in this study is intended to serve as baseline reference data for future comparative studies. Complete benchmark data is preserved in `results/final_dataset_n30/` and `results/medium_project_n30/` for reproducibility and further analysis.

References

1. Vercel. (2024). *Turbopack Documentation*. https://turbo.build/pack/docs
2. Next.js. (2024). *Next.js 14 Release Notes*. https://nextjs.org/blog/next-14
3. Webpack. (2024). *Webpack Documentation*. https://webpack.js.org/
4. Apple. (2020). *Apple M1 Chip Technical Specifications*. https://www.apple.com/mac/m1/

Appendix A: Raw Data Summary

A.1 Phase 1 - Cold Start Data Location

Complete benchmark data: `results/final_dataset_n30/`

A.2 Phase 2 - Medium Project HMR Values

Legacy (Webpack) - N=28:

```
198, 199, 200, 200, 201, 201, 202, 202, 203, 203,
204, 204, 205, 205, 206, 206, 207, 207, 208, 208,
209, 209, 210, 210, 211, 211, 211, 211
```

Sum: 5,748 | **Mean:** 205.29 ms

Turbopack - N=30:

```
22, 22, 23, 23, 23, 23, 23, 23, 23, 23,
23, 24, 24, 24, 24, 24, 24, 24, 24, 25,
25, 25, 25, 25, 25, 25, 25, 26, 26, 27
```

Sum: 722 | **Mean:** 24.07 ms

Appendix B: Reproduction Instructions

```
# 1. Clone repository
git clone https://github.com/VernSG/nextjs-toolchain-benchmark
cd nextjs-toolchain-benchmark

# 2. Install dependencies
npm install
```

```
# 3. Run Phase 1 (Baseline)
./scripts/run_benchmark.sh

# 4. Generate medium project
node scripts/generate_dummy.js

# 5. Run Phase 2 (Scalability)
./scripts/run_benchmark.sh

# 6. Generate charts
python scripts/generate_charts.py
```

Document generated: January 13, 2026

This document is intended as technical documentation for a benchmark dataset artifact.