01 wp rust ffi

git 8b49fe6 | 2026-09-20 | Jermaine Harris, Hutts Enterprises / Ferrus Labs

Moving WordPress Plugin Compute to Rust FFI: An Experience Report on Sitemaps, Related Posts, and Redirects

Jermaine Harris
Hutts Enterprises / Ferrus Labs

date: "2026-09-20"

keywords: "cs.SE, cs.PF, WordPress, Rust, PHP FFI"

Abstract

We measure three WordPress plugin kernels that already exist as PHP companions plus Rust `cdylib`s: XML sitemap generation, related-posts scoring, and redirect lookup. On one workstation (Intel Core i5-12600KF, PHP 8.3.6 FFI enabled, rustc 1.97.1) we time a WordPress-free PHP fallback copied from the plugins, the same work via PHP FFI into `libhutts_*.so`, and the native Rust APIs. Workloads are synthetic (10 000 URLs; 2 000 related-post candidates; 5 000 redirect rules). Sitemap XML is faster in native Rust than in the PHP fallback (0.74 ms vs 1.74 ms

mean, n=12); PHP FFI is slower than PHP because of JSON marshalling. Related-posts PHP fallback is an order of magnitude *faster* than Rust because it implements a weaker token-containment ranker, not cosine n-grams. Redirect lookup is the intended win: a radix trie in Rust is about 700× faster than a linear PHP scan on this fixture (0.22 ms vs 157 ms). We do not claim a new ranking algorithm, and we did not cut over production WordPress.

Keywords

cs.SE; cs.PF; WordPress; Rust; PHP FFI; sitemaps; redirects

1. Introduction

Hutts WordPress plugins keep PHP hook-compatible fallbacks when a Rust `.so` is missing (`hutts-rust-loader.php`). That split is an engineering convenience. It is not, by itself, a performance result. This report answers a narrower question:

RQ. On the same hardware and the same synthetic corpus, how do wall time and error behavior change for sitemap generation, related-posts ranking, and redirect lookup when the PHP companion runs alone versus Rust (FFI or native)?

Contributions, each mapped to Table 1:

- C1. A reproducible harness that never opens MySQL or `WP_Query` (`harness/wp-rust-ffi/`).
- C2. Paired timings for three kernels and three implementations.
- C3. An explicit finding that “Rust FFI” can lose to PHP when the PHP path is a cheaper algorithm or when JSON/FFI overhead dominates.

2. System under test

Repository `hutts-wp-rust-plugins` at `b2e08abc8f192719fbb302a46a13476879304ce3`.

Kernels:

```
| Kernel | Rust API | PHP fallback |
|--------|----------|--------------|
|     Sitemap     |     `urlset`     /     `hutts_sitemap_generate`     |
`hutts_sitemap_php_fallback` (XML loop) |
|  Related  |  `score_candidates`  (token  +  character  n-gram  cosine,
category-first) | token containment + category overlap |
| Redirect | `RedirectEngine` radix trie | linear scan of rules |
```

Release `cdylib`s were rebuilt from that SHA so FFI symbols (`hutts_redirect_engine_new`) match the source. An older `.so` on disk lacked the engine API.

3. Method

Hardware. 12th Gen Intel Core i5-12600KF, 64 GiB RAM, Linux 6.17, 16 hardware threads.

Software. PHP 8.3.6 NTS with `ffi.enable=true`; rustc 1.97.1; `--release` native binary.

Workloads (generated, not production posts).

- Sitemap: 10 000 `{loc,lastmod}` objects.
- Related: 1 query + 2 000 candidates (`ngram=3`,

`top_k=5`, `min_score=0.12`).

- Redirect: 5 000 rules (every 17th is prefix) and 1 668 lookup paths.

Protocol. 2 warmup + 12 timed repetitions. Wall time via `hrtime` (PHP) and `Instant` (Rust). Mean and population stdev. No WordPress; PHP fallbacks are WP-free copies of the plugin loops (`esc_url` → `htmlspecialchars`). Native Rust times the in-process APIs, not a full PHP request.

What we did not do. Production plugin cutover; date-randomizer; nDCG quality evaluation of related posts.

4. Results

Table 1. Mean wall time (seconds), n=12. Source: `results/01/kernel_times.csv`.

Kernel	PHP fallback	PHP FFI	Rust native
Sitemap (10k URLs)	0.001740	0.003319	0.000739
Related (2k candidates)	0.002762	0.068484	0.065644

| Redirect (5k rules, 1.7k lookups) | 0.156897 | 0.000648 | 0.000222 |

No timed iteration returned an empty error document. Redirect FFI and native hit the trie; PHP scanned the rule list.

Ratios (PHP fallback / Rust native): sitemap 2.35×; related 0.042× (PHP cheaper); redirect 708×.

5. Discussion and threats to validity

Sitemap FFI is slower than the PHP loop: the FFI path JSON-encodes the URL list on every call. Native Rust builds XML from already-deserialized structs. Operators who call FFI per request with large JSON payloads should not expect a win.

Related-posts numbers are not a language comparison. Rust computes mixed cosine similarity; PHP counts tokens of length >2. The PHP path is the correct *compatibility* fallback, not an optimized twin. A speed paper that

ignored this would be misleading.

Redirect lookup matches the folklore: O(path) trie versus O(rules). FFI still beats PHP here because lookup does not re-encode 5 000 rules each time (engine constructed once).

Threats. (1) Synthetic URLs and lorem-like candidate text. (2) Native Rust omits PHP request bootstrap. (3) Single machine, no production PHP-FPM. (4) PHP sitemap fallback omits `lastmod` nodes that Rust emits. (5) RSS was not sampled per kernel (short CPU-bound runs).

6. Related work

PHP FFI is documented in the PHP manual [phpffi]. WordPress plugin I/O remains PHP [wp-plugin-api]. The sitemap protocol is independent of language [sitemaps]. We do not cite unpublished “Rust is fast” blogs as evidence.

7. Ethics and operational safety

Fixtures never touched the WordPress fleet. No `post_date` mutation. Production `.so` files were not replaced.

8. Worked example (redirect)

A linear scan of 5 000 rules on 1 668 paths is about 8.3 million string comparisons in the worst case (every miss walks the whole list). Mean PHP time 157 ms implies roughly 50 ns per comparison on this CPU, which is plausible for `str_starts_with` on short paths. The trie walks one byte of the path; 1 668 lookups at 0.22 ms is about 130 ns per lookup, independent of rule count except for cache effects. That is why C3 (algorithm) dominates C2 (language) for redirects, while sitemap shows only a 2.35× XML-builder difference.

9. Conclusion

Rust helped where the data structure changed (redirect trie) and helped modestly for sitemap XML when marshalling was out of the timed path. It did not help related-posts latency versus a weaker PHP fallback. Moving plugin compute to Rust is an API and safety project first; speed requires paired algorithms and attention to FFI JSON.

9. Artifact appendix

```
python3 harness/wp-rust-ffi/run_bench.py
```

Requires PHP FFI, `libhutts_{sitemap,related_posts,redirect_manager}.so` in the plugins `target/release`, and `cargo build --release -p wp-rust-ffi-bench`. CSV SHA is the git blob of `results/01/kernel_times.csv` in this repository.

References

- PHP FFI. https://www.php.net/manual/en/book.ffi.php

- Sitemaps XML format. https://www.sitemaps.org/protocol.html

- WordPress Plugin Handbook. https://developer.wordpress.org/plugins/