

Problem 48

Full performance objective, current evidence, plan, and proof

Required target: Make every Developer Dashboard subroutine at least 50× faster than version 5.88, evaluating MCE, FFI::Platypus, Inline::C, XS, SPVM, Memoize, and Perl's built-in class/method/field/signatures—without changing behavior. The target is not yet achieved.

Problem statement and cause

A source-line inventory finds 1,648 named `sub` declarations in the current worktree and 1,624 in the v5.88 HEAD tree; anonymous closures are excluded. They do not share one workload: some are tiny getters, some mutate runtime state, some wait on filesystem/process/network I/O, and some are CPU-bound. A worker pool, cache, or native-language boundary cannot safely or profitably accelerate all of them in the same way. The goal therefore requires profiling and proof at the level of each claimed routine; a single faster algorithm is not evidence for the rest.

NYTProf found startup/module-loading and filesystem/environment discovery in light CLI paths, plus a real CPU hotspot in helper-password PBKDF2-HMAC-SHA256. These are distinct causes and need distinct measures.

Measurements so far

Path / method	Measured outcome	Interpretation
dashboard version	56.7 ms original median. Latest 15-run median: 31.486 ms.	Lazy imports, lightweight version access, and validated-PWD lookup yield about 1.80×. Still not 50×.
dashboard paths / ps1	71.309 ms / 83.641 ms latest 15-run medians.	About 1.31× versus respective 5.88 baselines; command-level gains only.
PBKDF2, 210,000 rounds	Perl 278–300 ms, median 284 ms. OpenSSL FFI 29.7–31.4 ms, median 29.9 ms.	Exact digest parity; about 9.5× faster for this one function.
Inline::C / XS versus FFI	Warmed averages: 31.0 ms versus 32.4 ms for the same OpenSSL call.	About 4% difference. Inline::C generates XS glue; no significant extra speed or duplicate backend.
MCE synthetic independent batch	Ten 300 ms jobs: 3.001 s sequential, 0.906 s using four workers.	Not the collector runtime. Real collector startup must preserve sequential readiness checks and fail-fast rollback.
Tiny FFI boundary	Perl accessor: 12.43 million calls/s; FFI getenv: 5.58 million calls/s.	Same value in the measured environment; FFI was about 2.23× slower. Boundary cost makes blanket wrapping counterproductive.

Technology findings

- **FFI::Platypus**: used for the measured PBKDF2 hotspot, with Perl fallback and exact byte/digest checks.
- **Inline::C / XS**: benchmarked as an alternate wrapper for the same OpenSSL function; no meaningful improvement beyond FFI.
- **MCE**: useful for independent expensive tasks; no behavior-safe production batch is established.
- **SPVM**: SPVM and SPVM::Digest::SHA installed in Docker; no Dashboard subroutine speedup has been demonstrated.
- **Memoize**: no safe hot candidate beyond existing explicitly keyed runtime caches; caching mutable filesystem state or password outputs is unsafe.
- **Perl class/method/field/signatures**: not a native-code transformation. The class feature is experimental in tested Perl 5.40.1 and emits warnings.

Solution and current status

1. Profile representative behavior with NYTProf — used on CLI paths and path tests.
2. Implement evidence-backed optimizations with red tests — FFI PBKDF2 and lazy usage-renderer load implemented.
3. Preserve behavior — focused crypto/auth/runtime tests and 1,029 CLI smoke tests passed in Docker.
4. Latest Docker run: 265 files / 23,316 tests passed. Coverage is 100.0% across all four metrics (47,375 detail rows). Nine optional files and three MCE-only assertions were skipped in the standard image; the focused MCE test passed 11 assertions. Post-gate `perl doc perlsec` passed.
5. Security, integration, dependency, docs/release, blank-install, image-build, and commit gates — pending.

New route under test: resident runtime

In Docker, ten fresh `dashboard paths -o json` executions averaged 84.412 ms each. The same handler in a resident process took 5.636 ms/request when aliases were rebuilt. Reusing the validated path/config context while recomputing the path map and JSON took 1.018 ms/request, with byte-for-byte parity against the current CLI payload (981 bytes): an experiment-level 82.9× reduction. A second prototype added a compiled C client and resident Perl Unix-socket server; 100 separate client launches averaged 0.514 ms/request with exact output parity, an experiment-level 164× reduction. This is a credible route to 50× for repeated requests.

These are prototype measurements, not a shipped fix or proof for all subroutines. A synthetic Docker mutation test found second-resolution stat can miss same-second directory changes. Focused tests now confirm existing readers see skill creation/deletion, disable-marker changes, config replacement, new skill aliases, and runtime symlink retargeting. A dynamic `Folder.pm` method is reevaluated after its environment input changes. Persistent-cache invalidation, refreshed output parity, and watcher cost remain unproven.

50× audit: A bare Perl process in the same container takes 1.032–1.268 ms. A 50× speedup for the original 56.7 ms median version command would require about 1.134 ms total—near the measured interpreter startup floor. This disproves the target for that invocation on this runtime. No 50× proof exists for the other 1,648 current named routines. Problem 48 remains in progress.

Expected outcome and proof required

Preserve observable behavior while materially accelerating any routine shown by profiling to be a real bottleneck. Current proven gains are 1.80× for version, 1.31× for paths, 1.31× for ps1, and 9.5× for PBKDF2. Do not claim the 50× whole-dashboard or every-routine target until the corresponding evidence exists.

- Run tests inside the unique Problem 48 Docker Compose project and retain complete logs.
- For each claimed routine, use fixed representative inputs, compare v5.88 against the optimized version over repeated runs, and verify exact output/error parity.
- Keep NYTProf test-harness time separate from actual application-route timings.
- Finish security, integration, dependency, version alignment, packaging, Docker image, blank-install, and git delivery gates before calling the cycle complete.

Editable source: [doc/problem-48-statement.md](#) · Supporting measurements: [doc/problem-48-performance-research.md](#)