2026-09-26

# `foil`: paired performance benchmarking for R packages

Visruth Srimath Kandali
California Polytechnic State University, San Luis Obispo

## 1 Executive Summary

R package developers need to detect runtime and memory regressions before changes are merged or released, but ordinary development machines and CI runners are noisy and R-level memory accounting does not capture memory usage from compiled code or child processes. `foil` is a Rust CLI for comparing the performance of two Git revisions using randomized paired benchmarks on the same machine. It models the paired measurements to account for drift and run-order effects and reports the estimated revision effect with uncertainty.

The funded work will extend the existing runtime measurement with process-tree peak memory measurement on Linux and Windows, binary distribution, and a strong R integration layer. Process-tree measurement captures memory used by compiled code, threads, and child processes. The project will publish prebuilt `foil` binaries for Linux, macOS, and Windows, together with an R package for running benchmarks and analyzing results. `foil` will also support opinionated initializations such as `foil init r-package`, which will write a template benchmark configuration and GitHub Actions workflow. Dependency installation, compilation, and other project-specific setup remain under the maintainer's control. Runtime comparison already works on macOS, Linux, and Windows; memory measurement will target Linux and Windows.

Work is scheduled from January through June 2027, with Linux memory measurement first, followed by distribution and R integration, `loo` case study, benchmark overhead and model diagnostics, Windows memory measurement, and release stabilization.

Requested funding is $10,000 USD for development labor across five milestones.

## 2 Signatories

### 2.1 Project team

Visruth Srimath Kandali is the sole developer of `foil`. He is a Master's student in Statistics at California Polytechnic State University, San Luis Obispo, and a member of the Stan development

team. His open-source work includes sustained development of `loo` as a Google Summer of Code contributor in 2025 and 2026. Relevant contributions to `loo` include testing and package infrastructure modernization and `touchstone`-based performance benchmarks. Visruth also develops R packages, including `bootstrapper`, an opinionated package for setting up package infrastructure and CI workflows.

More information is available at visruth.com.

### 2.2 Consulted

A number of people kindly read the proposal and provided feedback: Jonah Gabry, Aki Vehtari, Toby Hocking, Will Landau, Lorenz Walthert, Tyson Barrett.

`foil` will be trialed in a few Stan packages, beginning with `loo` and expanding as the Stan team evaluates it. `foil` will also be trialed in `crew` and `targets`. `data.table` has expressed interest in using `foil`.

## 3 The problem

Performance regressions can slip past unit tests and harm software, increasing runtime or memory usage. R package developers need a way to estimate how much a change affected performance. However, on most development machines and on CI runners, runtime measurements are noisy enough that distinguishing between normal variation and a real effect can be very challenging. One way to reduce that noise is to measure baseline and candidate revisions on the same machine, which reduces machine-to-machine variation (Laaber et al. 2019).

R packages often rely on compiled code or multiprocessing in order to speed up computation, which proves to be a problem for memory measurement; tracking R-level allocations will not capture true memory usage in these kinds of workloads. Reliable revision-level runtime and memory comparisons would let developers detect and investigate these regressions before they reach users, without requiring dedicated benchmarking hardware. Most benchmarking tools compare implementations, while profiling tools identify where time or memory is spent. `foil` instead targets revision-level regression testing: how did a code change affect runtime or memory usage, and how uncertain is that estimated change?

Existing R tools address parts of this problem: `bench` measures runtime and R allocation behavior (Hester and Vaughan 2025), `cross` runs expressions across package versions or Git branches (Vaughan 2026), and `atime` measures runtime and R-level memory across both package versions and data sizes, making it useful for detecting differences in asymptotic scaling (Hocking 2026; Amoakohene et al. 2026). `atime` can also compare multiple relevant package versions. `touchstone`, which compares a base and PR branch, comes closest to a paired revision experiment (Walthert and Wujciak-Jens 2026) and is already used in a variety of R packages. None of these tools measure process-tree peak memory and support arbitrary benchmark commands (see Section 7).

## 4 The proposal

### 4.1 Overview

`foil` is designed to evaluate how a change affects runtime or memory usage of software, most naturally by comparing a branch to `main`. `foil` already compares baseline and candidate Git revisions as

randomized pairs on the same machine, then models the paired measurements to account for drift and run-order effects and estimate the revision effect with uncertainty. The funded work adds support for measuring memory usage, as well as providing prebuilt binaries and an R package. No existing tool (see Section 7) provides the whole workflow required for developers to make effective decisions about performance changes.

`foil` will measure process-tree peak memory usage on Linux and Windows (see Section 8 for more details). This will let developers measure memory usage, even if packages use compiled code or parallelization, addressing a gap in existing tooling. `foil` reports the revision effect with uncertainty, ensuring that developers know when results are noisy. For R package developers, the funded work makes this workflow available through R and easier to run locally or in CI without installing a Rust toolchain.

From January through June 2027, the project will deliver Linux memory and the result schema, distribution and R integration, a `loo` case study and work on overhead and model diagnostics, Windows memory, and stable documented releases.

## 4.2 Detail

### 4.2.1 Model

`foil` models each paired repetition as a shared measurement level plus a candidate-minus-baseline effect. The model accounts for drift and run-order effects while estimating the adjusted difference between revisions. Pairing accounts for additive disturbances shared by the baseline and candidate measurements. See Section 9 for further details.

A core principle of `foil` is to preserve uncertainty rather than collapse it into a changed/unchanged decision. `foil` therefore focuses on reducing and quantifying uncertainty in estimating the revision effect, leaving the practical importance of the change to the maintainer (Gelman and Stern 2006; McShane et al. 2019).

### 4.2.2 Benchmark process-tree memory

Memory usage will be measured using OS facilities that track maximum memory usage across the benchmark process and its descendants. This captures memory used by native libraries, threads, and child processes. macOS memory measurement is out of scope due to lack of an OS interface for process-tree peak memory. Section 8 goes into more detail.

### 4.2.3 R integration and distribution

The R package will be released on CRAN and the R-multiverse. It will locate or help users install a `foil` executable[1] and provide a minimal R interface to `foil` to run benchmarks. It will also be able to import the raw measurements and uncertainty draws generated by `foil` for further analysis or visualization. The Rust crate will be published on crates.io; prebuilt binaries and installers will be published through GitHub Releases. `foil init r-package` will write a starter `foil.toml`, R benchmark entry point, and GitHub Actions workflow. The intended workflow is continuous regression testing on commits and pull requests, so performance changes can be measured soon after they are introduced rather than through occasional manual benchmarking before release. The integration is repository-level and does not require adding `foil` or its R package as a dependency of the package being benchmarked. Performance tests are typically distinct from unit tests, and benchmarks should use representative workloads to expose important runtime or memory differences.

[1] Similar to `cmdstanr`'s distribution model, which provides `cmdstanr::install_cmdstan()`.

#### 4.2.4 Results

`foil` already reports a summary of the runtime measurements and writes it to disk, as shown below. Memory results will extend the same structure, and the planned GitHub PR comment will also show the `foil` version and revisions compared.

```
release-compile: Comparing candidate (HEAD) to baseline (main) with 30 paired repetitions and 10000

Baseline:  25.7s
Candidate: 33.3s

Change: +7.7s (+29.90%)
  50% CrI: [+7.4s, +7.9s] (+28.82%, +30.89%)
  80% CrI: [+7.2s, +8.1s] (+27.79%, +31.74%)
  90% CrI: [+7.0s, +8.2s] (+27.12%, +32.25%)

P(candidate faster): 0.0% (0 of 10,000 draws)
```

This is a real result from `foil`, measuring binary build times. You can see the full results here, along with the `foil.toml` used to produce them.

#### 4.2.5 Minimum Viable Product

The MVP is a complete Linux workflow for measuring a performance change—including memory—and making the result available in R. A user can install `foil` without a Rust toolchain, compare two Git revisions locally or in CI, measure runtime and process-tree peak memory, and obtain an adjusted revision effect with uncertainty. The R package can locate or help install a compatible `foil` executable, run benchmarks, and import the resulting measurements and uncertainty draws.

#### 4.2.6 Architecture

A Rust CLI owns benchmark orchestration and inference. It resolves configuration and revisions, creates isolated Git worktrees, generates a randomized paired schedule, runs the user-defined benchmark command, and records each observation. Platform-specific backends contain the benchmark process tree and clean up descendants after each run. `foil` then estimates the revision effect from the paired measurements and uses a Bayesian bootstrap to quantify uncertainty without assuming a parametric error distribution (Rubin 1981). Raw measurements, uncertainty draws, and summary results are written in machine-readable form for downstream use. The R package will provide a thin interface over the Rust executable.

#### 4.2.7 Assumptions

`foil` deliberately makes few modeling assumptions: drift and run-order effects are linear, paired repetitions are independent after accounting for those effects, and disturbances shared by baseline and candidate are additive. Note that noise from CI may actually be multiplicative, which can be handled by modeling log ratios rather than raw differences—adding support for log ratios is simple and will be completed before the ISC grant work starts.

#### 4.2.8 External dependencies

`foil` depends on Git. Process-tree memory additionally requires cgroup v2 on Linux or Job Objects on Windows.

# 5 Project plan

## 5.1 Start-up phase

Development will build on the existing `foil` CLI, in the public GitHub repository. The current code is flexible and was built with support for memory measurement in mind. Work will start immediately by evaluating existing Rust cgroup libraries against the small internal implementation and implementing memory measurement on Linux.

`foil` will remain dual-licensed under MIT OR Apache-2.0; the R package will use BSD-3. Development, issue tracking, releases, milestone summaries, and scope changes will remain public.

## 5.2 Technical delivery

- By February 28, 2027: finalize the process-tree memory boundary and result schema; implement the Linux cgroup v2 backend; add controlled process-tree fixtures and measurement tests.
- By March 31, 2027: publish prebuilt `foil` binaries and installers; implement the R package for binary installation or discovery, compatibility checks, benchmark invocation, and result import; add `foil init r-package`, a GitHub Actions example, and compact terminal interval plots.
- By April 30, 2027: case study using `foil` with `loo`; improve diagnostics for repetition-position and run-order effects; measure and reduce the overhead introduced by `foil`, and add benchmarks to track overhead.
- By May 31, 2027: implement the Windows Job Object backend and add containment and measurement tests.
- By June 30, 2027: resolve release blockers, complete methodology documentation, stabilize the result schema and R interface, and publish stable releases of `foil` and the R package.

## 5.3 Other aspects

Code, documentation, issue tracking, and releases will remain public on GitHub.

An announcement post will invite maintainers to test early releases. Milestone posts will keep track of progress, and longer form midpoint and final posts will go into more detail. Updates will be published on visruth.com, offered to the R Consortium blog, and will automatically be submitted to R Weekly.

## 5.4 Budget & funding plan

Requested funding is $10,000 USD, entirely for labor. Under partial funding, Windows memory support will be cut first.

| Milestone | Target | Expected outcome | Budget allocation |
|---|---|---|---|
| Linux process-tree memory | February 28, 2027 | Memory contract and result schema; Linux cgroup v2 backend; controlled process-tree fixtures | $2,500 |

| Milestone | Target | Expected outcome | Budget allocation |
|---|---|---|---|
| Distribution and R integration | March 31, 2027 | Prebuilt releases and installers; R package with binary installation/discovery, compatibility checks, invocation, and result import; initializer and CI example | $2,500 |
| Benchmark overhead and model diagnostics + case study | April 30, 2027 | Measured and reduced `foil` overhead; regression benchmarks and improved model diagnostics; case study using `foil` in `loo` | $1,500 |
| Windows process-tree memory | May 31, 2027 | Windows Job Object backend with containment tests | $2,500 |
| Stabilization and release | June 30, 2027 | Stable CLI and R-package releases, methodology documentation, stabilized interfaces, and release blockers resolved | $1,000 |

# 6 Success

## 6.1 Definition of done

The project is complete when `foil` ships Linux and Windows process-tree memory backends, prebuilt binaries requiring no Rust toolchain, and an R package that can locate or help users install a `foil` executable and read its results. The release will also include a machine-readable result format, `foil init r-package`, documented R package GitHub Actions workflows, and extended documentation on the software and methodology.

## 6.2 Measuring success

Automated tests will verify the Linux and Windows memory backends and run them on controlled tasks to ensure measurement works. Benchmarks will track overhead introduced by `foil`, while model diagnostics will expose repetition-position and run-order effects. Release checks will verify that prebuilt binaries can be installed without a Rust toolchain, that the R package can locate or

install a compatible `foil` executable, run benchmarks, and read the machine-readable results, and that `foil init r-package` produces a working GitHub Actions workflow.

### 6.3 Future work

Development will continue after the funded period. Potential future work includes parallel benchmark execution, sampled memory measurement on macOS, additional statistical models, and a tool similar to `git bayesect` for locating performance regressions across commits (Jain 2026).

## References


Amoakohene, Doris, Anirban Chetia, Igor Steinmacher, and Toby Hocking. 2026. "The r Journal: Asymptotic Benchmarking Using the Atime Package." *The R Journal* 18: 169–87. https://doi.org/10.32614/RJ-2026-013.

Gelman, Andrew, and Hal Stern. 2006. "The Difference Between 'Significant' and 'Not Significant' Is Not Itself Statistically Significant." *The American Statistician* 60 (4): 328–31. https://doi.org/10.1198/000313006X152649.

GitHub. 2026. "Actions Limits."https://docs.github.com/en/actions/reference/limits#job-concurrency-limits-for-github-hosted-runners .

Hester, Jim, and Davis Vaughan. 2025. *bench: High Precision Timing of R Expressions*. https://bench.r-lib.org/.

Hocking, Toby. 2026. *atime: Asymptotic Timing*. https://doi.org/10.32614/CRAN.package.atime.

Jain, Shantanu. 2026. "Git Bayesect." March 22. https://hauntsaninja.github.io/git_bayesect.html.

Laaber, Christoph, Joel Scheuner, and Philipp Leitner. 2019. "Software Microbenchmarking in the Cloud. How Bad Is It Really?" *Empirical Software Engineering* 24 (4): 2469–508. https://doi.org/10.1007/s10664-019-09681-1.

Linux kernel developers. 2026. "Control Group V2." https://docs.kernel.org/admin-guide/cgroup-v2.html.

McShane, Blakeley B., David Gal, Andrew Gelman, Christian P. Robert, and Jennifer L. Tackett. 2019. "Abandon Statistical Significance." *The American Statistician* 73 (sup1): 235–45. https://doi.org/10.1080/00031305.2018.1527253.

Microsoft. 2021. "JOBOBJECT_EXTENDED_LIMIT_INFORMATION Structure." April 2.https://learn.microsoft.com/en-us/windows/win32/api/winnt/ns-winnt-jobobject_extended_limit_information .

Microsoft. 2025. "Job Objects."https://learn.microsoft.com/en-us/windows/win32/procthread/job-objects .

Rdatatable. 2026. "data.table." https://github.com/Rdatatable/data.table.

Rubin, Donald B. 1981. "The Bayesian Bootstrap." *The Annals of Statistics* 9 (1): 130–34. https://doi.org/10.1214/aos/1176345338.

Tandon, Akash, and Toby Dylan Hocking. n.d. "Rperform: Performance Testing for R Packages." https://github.com/analyticalmonk/Rperform.

Vaughan, Davis. 2026. *cross: Run Expressions Across Package Versions*. https://github.com/DavisVaughan/cross.

Walthert, Lorenz, and Jacob Wujciak-Jens. 2026. *touchstone: Continuous Benchmarking with Statistical Confidence Based on 'Git' Branches*. https://github.com/lorenzwalthert/touchstone.

# 7 Appendix A: Existing tools and evidence of demand

## 7.1 Evidence of demand

The R community has previously invested directly in revision-level performance testing. `Rperform`, developed through Google Summer of Code projects in 2015, 2016, and 2022, was designed to track runtime and memory across Git revisions and integrate performance testing into package CI. `Rperform` is no longer actively developed, but this repeated investment shows interest in detecting performance changes during R package development (Tandon and Hocking, n.d.).

An August 19, 2026 GitHub search for `touchstone` in `.Rbuildignore` files returned 35 public repositories across 22 owners after excluding repositories from the `touchstone` authors and the grant author. The repositories span statistical computing, epidemiology, Stan, graph analysis, simulation, and other R projects. Further, `data.table` uses `atime` to investigate timing and memory differences (Rdatatable 2026; Hocking 2026; Amoakohene et al. 2026).

## 7.2 Comparison with adjacent tools

| Tool | Pair modeled | Order/drift handling | Effect uncertainty | Memory | Workload | Asymptotics |
|---|---|---|---|---|---|---|
| `foil` | **Yes** | **Drift and run order modeled** | **Effect distribution** | **OS process-tree peak** | **Arbitrary command** | No |
| `bench` | No | None | No | R allocations and GC | R expression | No |
| `atime` | No | None | No | Asymptotic memory usage (`bench`) | R expression (`bench`) | **Yes** |

| Tool | Pair modeled | Order/drift handling | Effect uncertainty | Memory | Workload | Asymptotics |
|---|---|---|---|---|---|---|
| `cross` | No | None | No | R allocations and GC (`bench`) | R expression (`bench`) | No |
| `touchstone` | No | Randomized order, not modeled | **Confidence interval** | No | R expression | No |
| `Rperform` | No | None | No | Sampled R-process RSS | R package test files | No |
| `tango` | **Yes** | Randomized order, not modeled | Significance test | No | Rust microbenchmark | No |
| `zenbench` | **Yes** | Drift diagnosed, not modeled; execution order not modeled | **Confidence interval** | Allocations | Rust microbenchmark | No |
| CodSpeed | No | Controlled execution environment | Impact and regression thresholds | Allocations | Supported integrations/CLI | No |
| Bencher | No | None | Statistical thresholds and alerts | Varies | Varies | No |

`touchstone` is the closest R workflow, while `tango` and `zenbench` are the closest paired statistical designs. None of the tools reviewed combines arbitrary command revision comparison, pairing retained in inference, explicit adjustment for drift and run order, effect uncertainty, and OS process-tree peak memory.

# 8 Appendix B: Process-tree memory semantics

## 8.1 Measurement boundary

A measured invocation begins with the benchmark command and includes descendants created during that invocation. `foil` setup and teardown and any pre-existing processes remain outside the boundary. Native libraries and threads contribute through the measured process; parallel workers created as child processes contribute when they remain inside the platform containment boundary.

Linux and Windows use different OS mechanisms to implement the same process-tree measurement boundary.

### 8.2 Implementation details

For each measured invocation on Linux, `foil` will start the benchmark in a new cgroup, and read `memory.peak` after the command and measured descendants complete. Linux defines `memory.peak` as the maximum memory usage recorded for a cgroup and its descendants (Linux kernel developers 2026). If the required cgroup v2 access is unavailable, the metric is unsupported.

On Windows, `foil` will associate the benchmark with a Job Object before the measured workload proceeds and will report `PeakJobMemoryUsed` (Microsoft 2021) after the command and contained descendants complete. Child processes normally inherit job membership; nested jobs and breakaway settings can change containment (Microsoft 2025). Again, if unsupported, `foil` will only record runtime.

macOS memory measurement is not part of the proposal because macOS does not provide an interface like Linux cgroups or Windows Job Objects for contained process-tree peak memory. Support for macOS would require sampling, unlike Linux and Windows which provide exact measurements. Note that macOS runners are limited in GitHub Actions, supporting only five concurrent macOS jobs on the Free, Pro, and Team plans (GitHub 2026). Further, R package benchmark workflows typically use Linux runners, as seen in existing `touchstone` and `atime` GitHub Actions workflows.

## 9 Appendix C: Statistical model

For repetition $i$, let $B_i$ and $C_i$ be the baseline and candidate measurements, $r_i$ the centered repetition position, and $o_i \in \{-1, +1\}$ the run-order contrast indicating which branch was run first. Let $m_i$ and $d_i$ be the midpoint and difference in measurements:

$$m_i = \frac{B_i + C_i}{2}, \qquad d_i = C_i - B_i.$$

`foil` then fits two simple regressions:

$$m_i = \mu + \mu_r r_i + \mu_o o_i + \varepsilon_i^{(m)},$$

and

$$d_i = \Delta + \Delta_r r_i + \Delta_o o_i + \varepsilon_i^{(d)}.$$

`foil` uses the Bayesian bootstrap (Rubin 1981) to quantify uncertainty around the revision effect. `foil` also supports shrinkage, which acts only on the candidate-minus-baseline effect. The shrinkage parameter, $\lambda$, can be interpreted as the effective prior sample size for zero difference.