PEER REVIEW RECORD

rush: Scalable Asynchronous Distributed Computing via Shared State in R

Marc Becker and Bernd Bischl

Vol. 1 (2026), Art. 5 · https://doi.org/10.68365/jrepstat.2026.5

The decision letters, the reports the authors read, their answers and the editor's signed decisions, as written. The reports come from the journal's review engine. The authors could have kept this record private until they confirmed the proof.

DECISION PATH

2026-09-05	Minor revision
2026-10-12	Revised
2026-10-13	Accepted

Round 1: Minor revision

SIGNED DECISION

Confirmed and signed by Matthias Templ on 2026-09-05 · ORCID 0000-0002-8638-5276

DECISION LETTER

Thank you for your submission to JRepStat. We are inviting a minor revision. The reviewers' points follow, grouped by weight.

rush is an R package providing a Redis-backed shared-state coordination layer that brings decentralized asynchronous parallelism (as in Optuna and DeepHyper) to R, where existing packages offer only centralized controller-worker parallelism. It matters to R users running hyperparameter and parallel black-box optimization at scale, and to the mlr3 ecosystem it now backs. The reviewers found nothing to fix in the writing itself. A minor revision means the contribution stands and the remaining work is small. We would be glad to see the revised version.

How to read this letter. The decision rests on the items marked required. If a required item seems wrong to you, or cannot be done within this paper, say so in your response letter and give the reason. A reasoned decline is a legitimate answer, and the editor weighs it. Reviewers can be mistaken. Items marked recommended or optional are advice from the reviewers. Adopt them, decline them with a sentence, or leave them for later work. None of that counts against the paper. The reviewer reports below carry the same marks. The editor who signs this decision may weigh an item differently in the accompanying message, and that message prevails.

Reproducibility items marked required are the journal's minimum for a paper that ships code with its results: code and a command that runs it, and by acceptance a captured computational environment (dependencies/lockfile/container) and a signed determinism attestation for the entry-point command. They document what you already have. They do not ask for new analysis.

REQUIRED (the decision rests on these):

- [software-quality] Version inconsistency in the archived artifact. The Zenodo record (https://zenodo.org/records/22256435) is titled "benchmark 2026 rush 1.1.1", while Sections 4.1 and 5.1 state the benchmarks use rush 1.2.0. Reconcile the artifact label with the version actually used, or note which package version the deposited code pins, so the benchmarks can be reproduced against the correct release.
- [statistical-rigor] The headline BO comparison in Section 5 rests on n=1: "Each task–algorithm combination is run once." Table 2 reports utilization (94.81%, 95.62%, etc.) and evaluation counts to several

significant figures, and the text claims "roughly 300 times higher utilization", with no replication and no dispersion measure. The reader cannot tell how much of the gap is stable versus run-to-run scheduling variance. The effects span orders of magnitude, so the qualitative conclusion is plausible, but the precision implied by the reported digits is unsupported. Add at least two or three replicate runs per task–algorithm cell and report a range or standard deviation, or qualify the Section 5 numbers explicitly as a single-run illustration and reduce the reported precision (for example " 95%","roughly two orders of magnitude") rather than "94.81%".

RECOMMENDED:

- [consistency-check] Figure 5′s caption states the algorithms "start from the same initial design of 100 configurations that was evaluated before the timed runs (Section 5.1)", but Section 5.1 does not mention a 100-point initial design (the only initial designs described are the 25-point toy example in Section 3.2). Add the initial-design description to Section 5.1 or correct the cross-reference, since the figure points readers to text that does not contain the stated fact.
- [software-quality] The paper does not describe a test suite. For a coordination layer whose value rests on concurrency, error handling, and lost-worker detection, the absence of any stated tests for these paths is the main gap. State whether the package ships unit tests that exercise the documented failure modes (worker crash, heartbeat expiry, failed tasks) and the caching invariants, so a reader can judge coverage beyond the happy path.
- [software-quality] Portability depends on an external Redis server that must be installed and started separately, and the multi-node runs use a specific LRZ cluster. The install link is provided and mirai handles remote/SSH/SLURM deployment, so this is a declared caveat, not a hidden one. Record the declared dependency versions (redux, mirai, ranger, lhs, data.table) alongside the Redis version used, so a new user can build the same environment.
- [software-quality] CRAN/pkgcheck readiness cannot be judged from the text. The paper does not state whether rush itself is on CRAN or passes R CMD check, though several dependencies (bbotk, mlr3tuning) are cited as released packages. State the package's distribution channel and check status.
- [statistical-rigor] Comparative claims about budget adherence ("CL can overshoot … sometimes substantially") are drawn from single wall-clock values in Table 6 with no uncertainty, so the direction is asserted rather than demonstrated across runs. Tie this claim to the same replicate runs suggested above, or state that it is a single-run observation.
- [visualisation] Figure 3 plots runtimes that span from about 1 ms to over 40,000 ms (Table 3). State explicitly in the caption whether the y axis is log-scaled and label its units, since a linear axis would compress the cached curve to a flat line and hide the very speedups the figure claims.
- [visualisation] Figures 3 and 4 encode the two conditions as red and blue. Red versus blue is generally distinguishable under the common red-green deficiencies, so this is acceptable, but add a redundant channel (line type in Figure 3, fill pattern or direct labels in Figure 4) so the distinction survives greyscale printing.
- [visualisation] Figure 5 overlays three algorithm traces (CL, ACBO, ADBO). Confirm the three lines use a colourblind-safe palette plus distinct line types or direct end-of-line labels, because three overlapping time series separated by colour alone are the most common readability failure.
- [ethics-and-integrity] No competing-interest statement appears, though the authors develop the package and related mlr3 ecosystem tools they benchmark. This is standard for open-source method papers and is not an undisclosed financial interest. Add a short competing-interests declaration noting authorship of the rush, bbotk, and mlr3tuning packages.

OPTIONAL:

- [methods-soundness] The utilization metric U = Toptimization/TCPU counts surrogate fitting and acquisition optimization as productive time, and in ADBO each of the 448 workers refits its own surrogate on the full archive. Table 6 shows this dominates: on credit-g the cumulative surrogate time is 212,896 s versus 10,625 s of learner training, so most of the reported 94.81% "utilization" is redundant surrogate computation rather than objective evaluation. The headline "94–99% CPU utilization" therefore measures how busy the cores are, not how much useful optimization work is done. State

plainly in Section 5 that a large share of ADBO's counted time is surrogate refitting, and separate the evaluation share from the surrogate/acquisition share when reporting utilization.
- [statistical-rigor] The microbenchmark tables (Table 1, Table 3) report only the median over 10,000 replicates. No spread is given, so the reader cannot judge tail latency, which is the quantity that matters for per-task overhead under contention. The single large-payload cell (17.77 ms) suggests the distribution is heavy-tailed and a median hides this. Add an interquartile range or a high quantile (for example median and 95th percentile) to Tables 1 and 3.

REVIEWS

Report Reviewer 1, recommends minor

Written by the journal's review engine in the role of the application reviewer, and read by the editor before the decision. How the engine works is on the peer review page.

This paper addresses a real and clearly stated gap. The R parallel-computing ecosystem (parallel, future, mirai, batchtools, crew, rrq) is uniformly controller-worker, while decentralized, shared-state coordination has become standard in Python HPO frameworks such as Optuna and DeepHyper. Providing a Redis-backed coordination layer that lets R workers run their own ask-and-tell loops is a useful contribution to the applied HPO and black-box-optimization community, and the integration with bbotk and mlr3tuning gives it an immediate user base. The related-work section is accurate and fair about where existing R tools sit. The core-function benchmarks in Section 4 are careful and convincing: the per-task overhead numbers and the caching behaviour are well characterised across payload sizes and task counts, and the caching result (386 ms to 4 ms at 10,000 tasks) is a concrete, credible win.

My main concern is the framing of the Bayesian optimization benchmark, which carries the abstract's headline number (94 to 99 percent utilization versus single-digit). The utilization metric U counts surrogate fitting, acquisition optimization, and model training as productive CPU time. Under ADBO each of 448 workers refits its own surrogate on the growing shared archive at every proposal, so a large and growing share of the "utilized" CPU is redundant surrogate work rather than objective evaluation. Table 6 makes this visible: on credit-g, ADBO's cumulative surrogate time is 212,896 s against 10,625 s of learner time, so the objective evaluations are a small fraction of the busy CPU. High utilization here is partly a restatement that ADBO does far more surrogate fitting, not that it uses the budget more usefully. The discussion is honest that all three methods reach similar final error and that the extra evaluations mostly reach a given error sooner, but that caveat needs to appear where the 94 to 99 percent claim is made, in the abstract and Section 5.2, not only in Section 6. As written, a reader will take utilization as a proxy for optimization quality, which the paper's own Table 6 does not support.

Related to this, the overhead benchmarks that justify "sub-millisecond per-task overhead" are measured for a single worker against a local Redis over a Unix socket. The scalability story, however, rests on 448 concurrent workers reading and writing a shared store over TCP (footnote 3 notes TCP "adds some overhead" but gives no number). Contention on a single shared Redis instance under hundreds of concurrent workers is exactly the regime where a shared-state design can develop its own bottleneck, and it is not characterised. Either qualify the scalability claims to say the overhead figures are single-worker isolated measurements, or add a concurrency measurement. A sentence on Redis as a shared dependency (contention, single point of failure, in-memory growth as the archive grows) would also help an applied reader judge deployment risk.

Two comparability points are worth surfacing for the applied reader. First, CL evaluates each configuration on 10 cores (parallel CV) while ADBO evaluates each on a single core, so the per-evaluation runtimes and the meaning of "an evaluation" differ across methods; the paper notes the runtime differences but does not draw out the granularity difference. Second, each task-algorithm combination is run once, so the evaluation counts and final errors have no variance estimate. The utilization gaps are large enough to survive single runs, but the near-identical final errors deserve an explicit single-run caveat. I did not attempt to reproduce the runs; I note that the text cites rush 1.2.0 while the Zenodo artifact reference reads rush 1.1.1, which should be reconciled and pinned for reproducibility. None of these points undermines the package itself, which is the actual contribution and looks well engineered; they concern how the demonstration is presented and how much a reader should conclude from it.

Numbered suggestions to the authors:

1. REQUIRED In the abstract and Section 5.2, qualify the 94 to 99 percent utilization result: state that U counts surrogate fitting and model training as productive time, that ADBO's high utilization reflects large and growing surrogate-fitting cost (cite the Table 6 credit-g figures), and that higher utilization did not produce lower final error.
2. REQUIRED State plainly that the Section 4 overhead numbers are single-worker measurements against a local Redis over a Unix socket, and qualify the 448-worker scalability claims accordingly, or add a concurrent-access measurement.
3. REQUIRED Reconcile the rush version between the text (1.2.0) and the Zenodo reference (1.1.1), and pin the exact version and commit used for the benchmarks.
4. RECOMMENDED Note that CL evaluates each configuration on 10 cores while ADBO uses one core per evaluation, and explain how this granularity difference affects the evaluation-count comparison.
5. RECOMMENDED Add a short paragraph on the shared Redis store as a contention point, single point of failure, and source of memory growth at scale.
6. RECOMMENDED Add an explicit single-run caveat to the evaluation-count and final-error results in Table 2 and Table 6.
7. OPTIONAL An asynchronous random-search baseline on the same tasks would isolate how much of ADBO's utilization advantage is specific to Bayesian optimization rather than to asynchronous scheduling alone.

Report Reviewer 2, recommends minor

Written by the journal's review engine in the role of the methodology reviewer, and read by the editor before the decision. How the engine works is on the peer review page.

This is a clear and well-motivated software paper. The gap it identifies is real: R parallel frameworks (parallel, future, mirai, batchtools, crew) are centralized controller-worker designs, and none supports the shared-state, worker-autonomous pattern that decentralized optimization needs. The Redis data model (hashes for tasks, sets for state tracking, lists for the queue and finished-task ordering) is a sensible mapping, and the caching design that reads only keys past the current cache length is the right idea for iterative algorithms. The core-function benchmark in Section 4 is methodologically sound: medians over 10,000 microbenchmark replicates, a clean sweep over field count and payload size, and a local Unix-socket Redis to isolate library overhead from network cost. The sub-millisecond per-task claim is supported by Table 1, and the caching benefit in Table 3 is large and credible (386 ms versus 4 ms at 10,000 tasks). My comments concern how the Bayesian optimization benchmark in Section 5 is interpreted, not whether the infrastructure works.

My main concern is the effective CPU utilization metric. You define U as Toptimization over TCPU, where Toptimization sums surrogate fitting, acquisition optimization, and model training across all workers. Under ADBO every worker fits its own surrogate on every iteration, and Table 6 shows this dominates the compute. On credit-g the learner (objective) time is 10,625 s while surrogate fitting is 212,896 s, so about 4% of ADBO's counted "optimization" time is actual objective evaluation and roughly 84% is surrogate refitting. The 94.8% utilization figure is therefore literally correct as "cores are busy" but does not mean "cores are doing productive optimization work." The redundant per-worker surrogate fits are precisely what fills the cores. This matters because the sentence "higher utilization translates directly into more configurations explored, potentially improving final optimization performance" reads as an efficiency claim, and the paper's own discussion (similar final errors, growing surrogate cost) partly contradicts it. The fix is not new experiments. Table 6 already holds the decomposition. Report the share of CPU going to objective evaluation versus surrogate, or soften the interpretive sentences, so a reader does not equate utilization with optimization efficiency.

The second issue is replication. Each task–algorithm combination is run once, with no variance estimate. For the headline utilization contrast this is acceptable, since 0.3% versus 95% will not flip under re-runs. For the evaluation counts and the final classification errors it is not, and those differences are small (for example 0.1850 versus 0.1800 on credit-g). State the single-run design where the comparison is made, and keep the performance claims qualitative. You already note that ADBO mainly reaches a given error earlier rather than reaching a lower one, which is the honest reading.

On reproducibility, the manuscript states benchmarks use rush 1.2.0 but the cited Zenodo record is titled "benchmark 2026 rush 1.1.1." Please reconcile these so a reader knows which artifact reproduces the tables. The BO runs use random draws (rexp for the per-worker lambda, runif for candidate sampling), and with 448 asynchronous workers exact reproducibility is timing-dependent and out of reach, which is fine, but documenting the seeding scheme for the deterministic parts would help. Finally, Section 5 does not state which surrogate and acquisition each of CL, ACBO, and ADBO uses in the LightGBM benchmark. The toy example uses a ranger random forest with a lower confidence bound, and mlr3mbo is cited, but the reader should not have to infer the benchmark configuration.

Numbered suggestions to the authors:

1. REQUIRED Using the Table 6 decomposition, report how much of ADBO's counted CPU time is objective evaluation versus surrogate refitting, or qualify the sentences stating that higher utilization translates directly into more configurations and better performance, so utilization is not read as optimization efficiency.
2. REQUIRED State next to the comparative results that each task–algorithm combination is run once, and limit the evaluation-count and final-error comparisons to qualitative statements; keep the utilization contrast, which is large enough to hold.
3. RECOMMENDED Add a throughput metric (completed objective evaluations per CPU-second) alongside utilization, so evaluation efficiency is visible separately from core busyness.
4. RECOMMENDED Reconcile the version mismatch (text: rush 1.2.0; Zenodo record title: rush 1.1.1) and document the RNG seeding used in the benchmark scripts.
5. RECOMMENDED Specify the surrogate model and acquisition function used for each of CL, ACBO, and ADBO in the LightGBM benchmark.
6. OPTIONAL A direct comparison against DeepHyper or Optuna would settle how rush stands relative to the established Python decentralized frameworks.

Report Reviewer 3, recommends minor

Written by the journal's review engine in the role of the software reviewer, and read by the editor before the decision. How the engine works is on the peer review page.

This is a well-scoped software paper. It identifies a genuine gap in the R ecosystem, that every popular R parallel backend uses a central controller and none supports decentralized shared-state loops, and it fills that gap with a clean Redis-backed design. The API walkthrough in Section 2 is concrete and the mapping of the task model onto Redis hashes, sets, and lists is described clearly enough that I can follow the intended semantics. The ADBO case study is a good motivating example and the core-function microbenchmarks are the right thing to measure for a coordination layer. From a software-and-reproducibility standpoint my concerns are about verifiability rather than design.

The most concrete problem is a version mismatch. The text states throughout that benchmarks use rush 1.2.0, while the cited Zenodo record (Becker, 2026) carries the title "benchmark 2026 rush 1.1.1". A reader trying to confirm that the reported numbers are the code's results cannot tell which version was actually run. This needs one clear statement of the version that produced Table 1, Table 2, and Table 3, and the matching code archived under that version.

The environment is under-specified for a timing paper. Table 1 reports sub-millisecond medians and Table 3 reports fetch times that depend heavily on the Redis build and transport. The Redis server version is not given anywhere, and the microbenchmark section names only some R package versions. A sessionInfo dump or a lockfile, plus the Redis version and whether it was a stock build, would let a reader reproduce the timing regime rather than just trust it. This is the reproducibility minimum for a paper whose central claims are numeric overheads.

The BO benchmarks state that each task-algorithm combination was run once. The optimizer draws random points, fits random forests, and samples a per-worker lambda from Exp(1), so results depend on RNG state. I am not asking for new runs, the effect sizes in Table 2 are large enough that a single run is defensible for the utilization claim. I am asking that seed handling be documented and that the single-run nature be stated plainly, so the numbers are not read as averages. Relatedly, the utilization metric

deserves a caveat. Table 6 shows that for credit-g the ADBO cumulative surrogate time is 212,896 s against 10,625 s of learner time, so the 94.8% utilization is dominated by redundant per-worker surrogate fitting rather than objective evaluation. The cores are busy, but much of that work is each worker refitting its own model. The paper is honest that final error is similar across methods, so a sentence distinguishing occupied cores from useful throughput would keep the headline number from being over-read.

On code soundness, the concurrency guarantees are asserted but not explained. Section 3.1 promises registration "in a concurrent, conflict-free manner" and the design relies on multiple workers writing to the same running-tasks set and the same finished list. Whether atomicity comes from Redis single-threaded command execution, from MULTI/EXEC transactions, or from Lua scripts matters for anyone building on rush, and it is the kind of thing a coordination layer must get right under 448 concurrent workers. A short paragraph on the atomicity mechanism would raise my confidence in the correctness claims. A small local-Redis example that reproduces the qualitative utilization gap on a handful of workers would also let readers without LRZ access exercise the package.

Numbered suggestions to the authors:

1. REQUIRED Reconcile the version discrepancy between the text (rush 1.2.0) and the Zenodo artifact title ("rush 1.1.1"). State the exact version that produced Tables 1, 2, and 3, and archive that version alongside the paper.
2. REQUIRED Report the full software environment for both benchmark sections, including R version, Redis server version and build, and mirai, redux, and ranger versions. Provide a captured environment such as sessionInfo output or a lockfile.
3. REQUIRED Document seed and RNG handling for the BO benchmarks, and state explicitly that each task-algorithm combination was run once, or qualify the utilization figures as single-run measurements.
4. RECOMMENDED Add a short description of how concurrent Redis operations are kept conflict-free (single-threaded execution, transactions, or Lua scripts), since the paper asserts conflict-free registration without giving the mechanism.
5. RECOMMENDED Add a caveat to the utilization results noting that, per Table 6, most of ADBO's Toptimization is redundant per-worker surrogate fitting, so high utilization does not by itself indicate more useful objective throughput.
6. RECOMMENDED Provide a small-scale, local-Redis reproducible example (a few workers) so readers without HPC access can reproduce the utilization pattern qualitatively.
7. OPTIONAL A direct comparison against DeepHyper or Optuna would help position rush across languages, though this can reasonably be deferred to future work.

REVISION, RECEIVED 2026-10-12

This revision was answered under the earlier rule: a short note on what changed, and a reason for each required point the authors declined.
The authors' response letter, as uploaded, follows as Appendix A.

Round 2: Accepted

SIGNED DECISION

Signed by Matthias Templ on 2026-10-13 · ORCID 0000-0002-8638-5276

Final acceptance on v2 (d8df9547) after the founder read the preview of 2026-10-13 as PDF and page and asked for the availability section in the authors' words (ADR 0095). Score confirmed as proposed (02ecf608).

DECISION LETTER

Congratulations, your manuscript has been accepted for publication in JRepStat. Production begins now.

Where it is not on file yet, we still need the following for production.

- a valid ORCID iD for every author
- a captured computational environment (dependencies/lockfile/container)
- a signed determinism attestation for the entry-point command

REVIEWS

Report Reviewer 1, recommends accept

Written by the journal's review engine in the role of the application reviewer, and read by the editor before the decision. How the engine works is on the peer review page.

Of the three required points and four lighter ones I raised, all are either met or declined with a reason I find convincing.

The two framing concerns that carried my earlier recommendation are resolved. Table 6 now reports learner utilization separately from the summed CPU utilization, so a reader can see that most of ADBO's counted time on the short tasks is surrogate fitting rather than objective evaluation (on credit-g, 4.2% learner utilization against 94.6% overall), and Section 5.2 states that the higher utilization reaches a given error earlier without lowering the final error. The overhead benchmarks are now labelled as single-worker measurements against a local Redis over a Unix socket, and Section 6 states that per-operation latency under concurrent TCP access was not measured. The added five replications, means and standard deviations in Tables 2 and 6, and the best-to-worst bands in Figure 5 settle the single-run concern well beyond what I asked for.

On the point where the authors pushed back, whether per-worker surrogate fitting in ADBO is "redundant," their response is the right kind of answer. They checked how often workers fit a surrogate on an identical archive with no running points and found one such case across 20 ADBO runs, with the proposals still differing because of the randomized acquisition. That is a direct empirical rebuttal, and I accept it. The learner-utilization column does the separation I actually needed, so the terminology disagreement does not need to be litigated in the text.

The two declined recommendations are reasonable. The Redis-contention paragraph was declined on the ground that the benchmarks do not show contention or memory limiting performance at the tested scale (archives of a few tens of megabytes, 448 workers against a default limit of 10,000 connections); the short note the authors did add covers the single-point-of-failure and persistence points adequately for a reader judging deployment. The asynchronous random-search baseline was declined as out of scope for a software paper centered on the infrastructure, which I agree with. One small residual: the abstract still leads with the headline utilization figure, and the qualification lives in Sections 5.2 and 6 rather than the abstract. For a summary this is acceptable and I would not hold the paper for it.

Numbered suggestions to the authors:

(none — all required points are met and the declined items are reasonably justified)

Report Reviewer 2, recommends accept

Written by the journal's review engine in the role of the methodology reviewer, and read by the editor before the decision. How the engine works is on the peer review page.

Of the two required points and three lighter ones I raised, all are met, and the two optional items were declined with reasons I find reasonable.

The utilization-interpretation point is settled. Table 6 now carries a learner-utilization column that separates time spent evaluating configurations from time spent on surrogate and acquisition work, and Section 5.2 reports both the decomposition (on credit-g, 211,614 s of surrogate fitting against 11,164 s of learner training) and the conclusion that drove my original concern, that higher utilization does not by itself imply a lower final error and mainly buys earlier convergence. The authors' separate argument that per-worker surrogate fitting is not redundant because workers operate on different archive states is a fair methodological point, and it does not conflict with reporting the split, which is what I asked for. Both coexist in the revised text.

The replication point is more than met. Rather than qualifying a single run, the authors reran with five replications per task-algorithm cell and report means and standard deviations in Tables 2 and 6, with Figure 5 showing best-to-worst bands. The utilization contrast holds within less than a percentage point

across ADBO replications, and the final-error differences are now reported with their dispersion and described as small, which is the honest reading. The airlines cell, where ACBO utilization ranges from 20% to 41% across runs, is exactly the case where single-run reporting would have misled, so the replication was worth the compute.

On the three lighter asks, the version mismatch is resolved (rush 1.3.0, Zenodo record retitled), seeding is documented in Section 5.1 with the correct caveat that asynchronous result ordering is timing-dependent, and the surrogate and acquisition configuration for all three strategies is now stated (random forest surrogate, confidence bound, local search). The throughput metric was declined on the ground that evaluations-per-CPU-second conflates optimizer behaviour with the heterogeneous runtime of the selected points; given that the learner-utilization column now exposes evaluation share directly, I accept that reasoning. The cross-language DeepHyper/Optuna comparison is deferred to future work and named as a limitation, which is the right call for a software paper of this scope.

Numbered suggestions to the authors:

(none)

Report Reviewer 3, recommends accept

Written by the journal's review engine in the role of the software reviewer, and read by the editor before the decision. How the engine works is on the peer review page.

Of the three required points and four lighter ones I raised, all are met or reasonably declined.

The version discrepancy is resolved: the text uses rush 1.3.0 throughout and the Zenodo record (22899537) matches. The environment is now specified for both benchmark sections, with R 4.6.0, Redis 8.10.1 from conda-forge, a renv.lock, and setup_hpc.sh, which is enough to rebuild the timing regime. The single-run concern is more than addressed: the BO benchmarks now use five replications per task-algorithm cell, Tables 2 and 6 report means and standard deviations, and Section 5.1 documents the fixed seeds, the shared initial designs, and the point that async result ordering is not bit-reproducible under fixed seeds. The microbenchmark tables now carry the 95th percentile alongside the median, which answers the tail-latency question I did not raise but which strengthens the overhead claims.

On the concurrency mechanism, Section 2.7 now names the three guarantees: single-threaded command execution for individual operations, MULTI/EXEC for multi-key writes such as push_running_tasks, and Lua scripts for competing state transitions in finish_tasks and fail_tasks. This is the level of detail someone building on rush needs to trust the correctness claims.

On the utilization caveat, the authors added a learner utilization column to Table 6 and separated evaluation time from surrogate and acquisition time in Section 5.2, and the discussion states that higher utilization did not produce lower final error. They declined to call ADBO's per-worker surrogate fitting "redundant" and gave a careful argument: workers fit on differing archives (different imputed running points, stochastic acquisition lambda, random forest randomness), and they measured only one post-initial-design case across 20 runs where four workers saw an identical archive. That reasoning convinces me. Reporting learner utilization separately was the substantive ask, and it is done, so the distinction between busy cores and useful throughput is now visible to the reader.

The local-Redis example was declined with a reason I accept: the utilization advantage arises when the central controller cannot keep workers fed, which does not occur at a handful of workers, so a small example would not reproduce the pattern qualitatively, and the vignette already covers functional demonstration. The DeepHyper/Optuna comparison was optional and is noted as future work. I have no open points and recommend acceptance.

Numbered suggestions to the authors:

(none)

APPENDIX A

The authors' response letter to round 1

As uploaded with the revision received 2026-10-12. Its own pages follow.

Dear Professor Templ and Reviewers,

Thank you for the constructive reviews of our manuscript and for the opportunity to submit a minor revision. We have rerun the benchmarks with `rush` 1.3.0 and added five replications to the Bayesian optimization benchmark. We have also expanded the descriptions of the software environment, testing, and concurrency mechanisms, improved the figures, and added a competing-interests declaration. Section, table, and figure references refer to the revised manuscript.

Editor

1. [software-quality] Version inconsistency in the archived artifact. The Zenodo record (https://zenodo.org/records/22256435) is titled "benchmark 2026 rush 1.1.1", while Sections 4.1 and 5.1 state the benchmarks use rush 1.2.0. Reconcile the artifact label with the version actually used, or note which package version the deposited code pins, so the benchmarks can be reproduced against the correct release.

Version 1.1.1 referred to the benchmark repository, not the `rush` package. We reran the benchmarks with `rush` 1.3.0. The record name "benchmark 2026 rush 1.3.0" identifies the package version used now (https://zenodo.org/records/22899537).

2. [statistical-rigor] The headline BO comparison in Section 5 rests on n=1: "Each task–algorithm combination is run once." Table 2 reports utilization (94.81%, 95.62%, etc.) and evaluation counts to several significant figures, and the text claims "roughly 300 times higher utilization", with no replication and no dispersion measure. The reader cannot tell how much of the gap is stable versus run-to-run scheduling variance. The effects span orders of magnitude, so the qualitative conclusion is plausible, but the precision implied by the reported digits is unsupported. Add at least two or three replicate runs per task–algorithm cell and report a range or standard deviation, or qualify the Section 5 numbers explicitly as a single-run illustration and reduce the reported precision (for example " 95%","roughly two orders of magnitude") rather than "94.81%".

We repeated the Section 5 benchmarks with `rush` 1.3.0 and five replications per task-algorithm combination. Table 2 and 6 now report means and standard deviations. Figure 5 shows the mean incumbents and the shaded bands representing the best and worst replications. For each task, ADBO utilization varies by less than one percentage point across replications. This variation is small relative to the differences between algorithms. On credit-g, ADBO utilization is 270 to 300 times that of CL and ACBO. The introduction and conclusion now report utilization above 93% in every ADBO run. The largest variation occurs on airlines. Here, ACBO utilization ranges from 20% to 41%, and ADBO utilization is 2.4 to 4.9 times that of ACBO in the corresponding replications. Final classification errors remain similar across methods. ADBO has the lowest mean error on all four tasks. The differences are small: up to 0.8 percentage points on credit-g and less than 0.1 percentage points on the other tasks. The discussion in section 6 states that the benchmark is not designed to assess final optimization performance.

3. [consistency-check] Figure 5's caption states the algorithms "start from the same initial design of 100 configurations that was evaluated before the timed runs (Section 5.1)", but Section 5.1 does not mention a 100-point initial design (the only initial designs described are the 25-point toy example in Section 3.2). Add the initial-design description to Section 5.1 or correct the cross-reference, since the figure points readers to text that does not contain the stated fact.

Section 5.1 now describes the initial design. We randomly sampled initial designs from the search space and evaluated them using a 10-fold cross-validation. We did this for each replication and task to ensure a consistent starting point across the algorithms.

4. [software-quality] The paper does not describe a test suite. For a coordination layer whose value rests on concurrency, error handling, and lost-worker detection, the absence of any stated tests for these paths is the main gap. State whether the package ships unit tests that exercise the documented failure modes (worker crash, heartbeat expiry, failed tasks) and the caching invariants, so a reader can judge coverage beyond the happy path.

The new Section 2.9 describes the test suite. `rush` includes 105 unit tests implemented with `testthat` that run on GitHub Actions on the release and development versions of R. The tests cover the core functionality, error handling, lost-worker detection, concurrent access, and caching behavior of the package.

5. [software-quality] Portability depends on an external Redis server that must be installed and started separately, and the multi-node runs use a specific LRZ cluster. The install link is provided and mirai handles remote/SSH/SLURM deployment, so this is a declared caveat, not a hidden one. Record the declared dependency versions (redux, mirai, ranger, lhs, data.table) alongside the Redis version used, so a new user can build the same environment.

Section 4.1 now lists the versions of R, Redis, and `rush`. The benchmark repository includes `setup_hpc.sh` to install these versions. The `renv.lock` file records the whole R package environment including ranger, lhs and data.table.

6. [software-quality] CRAN/pkgcheck readiness cannot be judged from the text. The paper does not state whether rush itself is on CRAN or passes R CMD check, though several dependencies (bbotk, mlr3tuning) are cited as released packages. State the package's distribution channel and check status.

Section 2 now states that `rush` is available on CRAN, with version 1.3.0 current at the time of revision. The development version is hosted on GitHub. Having the package available on CRAN means that `R CMD check` passes.

7. [statistical-rigor] Comparative claims about budget adherence ("CL can overshoot … sometimes substantially") are drawn from single wall-clock values in Table 6 with no uncertainty, so the direction is asserted rather than demonstrated across runs. Tie this claim to the same replicate runs suggested above, or state that it is a single-run observation.

The budget comparison now uses five replications per task-algorithm combination. Table 6 reports the mean and standard deviation of wall-clock time.

8. [visualisation] Figure 3 plots runtimes that span from about 1 ms to over 40,000 ms (Table 3). State explicitly in the caption whether the y axis is log-scaled and label its units, since a linear axis would compress the cached curve to a flat line and hide the very speedups the figure claims.

The y axis now uses a logarithmic scale and includes the unit in its label. The caption states that both axes are log-scaled.

9. [visualisation] Figures 3 and 4 encode the two conditions as red and blue. Red versus blue is generally distinguishable under the common red-green deficiencies, so this is acceptable, but

add a redundant channel (line type in Figure 3, fill pattern or direct labels in Figure 4) so the distinction survives greyscale printing.

Figure 3 now uses circles for cached fetching and triangles for fetching without caching with an explantion in the caption. In Figure 4, bar height already distinguishes the phases: proposal uses one core, and evaluation uses ten. We also use light and dark fills to distinguish the phases in grayscale. The caption describes these fills.

10. [visualisation] Figure 5 overlays three algorithm traces (CL, ACBO, ADBO). Confirm the three lines use a colourblind-safe palette plus distinct line types or direct end-of-line labels, because three overlapping time series separated by colour alone are the most common readability failure.

Figure 5 now uses orange, blue, and green from the Okabe-Ito palette. The algorithms also have distinct line types: dotted for CL, dashed for ACBO, and solid for ADBO. The caption identifies both encodings. Figures 3 and 4 use the same palette.

11. [ethics-and-integrity] No competing-interest statement appears, though the authors develop the package and related mlr3 ecosystem tools they benchmark. This is standard for open-source method papers and is not an undisclosed financial interest. Add a short competing-interests declaration noting authorship of the rush, bbotk, and mlr3tuning packages.

A competing-interests section now follows the acknowledgments. It states that the authors develop and maintain `rush`, `bbotk`, `mlr3tuning`, and `mlr3mbo`. It also declares that the authors have no financial competing interests.

12. [methods-soundness] The utilization metric U = T optimization/T CPU counts surrogate fitting and acquisition optimization as productive time, and in ADBO each of the 448 workers refits its own surrogate on the full archive. Table 6 shows this dominates: on credit-g the cumulative surrogate time is 212,896 s versus 10,625 s of learner training, so most of the reported 94.81% "utilization" is redundant surrogate computation rather than objective evaluation. The headline "94–99% CPU utilization" therefore measures how busy the cores are, not how much useful optimization work is done. State plainly in Section 5 that a large share of ADBO's counted time is surrogate refitting, and separate the evaluation share from the surrogate/acquisition share when reporting utilization.

We agree that showing the share of time spent on learner training is useful. We added a learner utilization column to table 6 and describe the learner utilization in section 5.1 and 5.2.

However, we believe that surrogate fitting is not "redundant" in the context of ADBO. All three strategies fit one surrogate per proposed configuration and this is part of the optimization process rather than overhead. We think this is likely a (partial) misunderstanding, but also a subtle point, so let us explain this here carefully. Let's start with the metric U. We measure the time where CPUs are active (surrogate fitting, acquisition optimization, and learner training, summed over workers) divided by the complete time of an optim trace. Is this is $< 100\%$ CPUs must have been idle. And compute resources have been "underused", which is bad. We think nobody is disagreeing here. What we take from the editors criticism (repeated by multiple reviewers) is: If this is close to 100% this is not necessarily "ideal" *if* computation is redundantly / unnecessarily replicated i.e. the "redundant" fitting of the surrogates on the archive. But the whole point of ADBO is that this surrogate fitting / acquisition function optimization is *not* redundant as it does not operate on the same state! Let's go through a few examples. Let's take 2 workers. Worker 1 reads the archive. It fits a surrogate and then proposes a point. This point is evaluated. We wait. While we wait

worker 2 reads the archive. This now includes the proposed point of W1 – with an imputed fitness values. State is different. Now it could happen that W2 reads the archive while W1 still proposes. Even then the state or proposal differs: a) The acquisition function is randomized (random lambda parameter in LCB per worker) and there is randomness in the random forest and acquisition function optimizer. b) Even the training data of the surrogate model is most of the time different. We can – in a multi-worker setup – likely assume that SOME points are under eval currently. These are also then imputed randomly. We added a footnote to section 5.1 explaining why surrogate fitting in ADBO is not redundant.

We checked how often the workers fitted a surrogate on the same archive with no running points. In this case the workers would indeed see the same archive because nothing is imputed.
In the first iteration of each worker, this happens often because the workers see only the initial design. But each time a different configuration was proposed due to the randomness in the acquisition function. After the first iteration, we found one case in the 20 ADBO runs where four workers saw the same archive with no running points (First replication of ADBO on credit-g). So redundant work is unlikely to happen during the optimization process.

13. [statistical-rigor] The microbenchmark tables (Table 1, Table 3) report only the median over 10,000 replicates. No spread is given, so the reader cannot judge tail latency, which is the quantity that matters for per-task overhead under contention. The single large-payload cell (17.77 ms) suggests the distribution is heavy-tailed and a median hides this. Add an interquartile range or a high quantile (for example median and 95th percentile) to Tables 1 and 3.

Tables 1 and 3 now report the median with the 95th percentile in parentheses. The largest configuration in Table 1 has 100 fields with 10,000 values each. Its median push time is 12.98 ms, and its 95th percentile is 25.41 ms. The corresponding finish times are 12.07 ms and 20.80 ms. The 95th percentiles are approximately 2.0 and 1.7 times the medians.

Reviewer 1

1. [required] In the abstract and Section 5.2, qualify the 94 to 99 percent utilization result: state that U counts surrogate fitting and model training as productive time, that ADBO's high utilization reflects large and growing surrogate-fitting cost (cite the Table 6 credit-g figures), and that higher utilization did not produce lower final error.

We agree that showing the share of time spent on learner training is useful. We added a learner utilization column to table 6 and describe the learner utilization in section 5.2. Section 5.2 also mentions now that the higher utilization did not translate into lower final error but rather into finding the optimal configurations earlier.

2. [required] State plainly that the Section 4 overhead numbers are single-worker measurements against a local Redis over a Unix socket, and qualify the 448-worker scalability claims accordingly, or add a concurrent-access measurement.

Section 4.1 now specifies that the measurements use a single worker and a local Unix socket. We write in Section 6 now that per-operation latency under concurrent TCP access was not measured separately.

3. [required] Reconcile the rush version between the text (1.2.0) and the Zenodo reference (1.1.1), and pin the exact version and commit used for the benchmarks.

Version 1.1.1 referred to the benchmark repository, not the `rush` package. We reran the benchmarks with `rush` 1.3.0. The record name "benchmark 2026 rush 1.3.0" identifies the package version used now (https://zenodo.org/records/22899537).

4. [recommended] Note that CL evaluates each configuration on 10 cores while ADBO uses one core per evaluation, and explain how this granularity difference affects the evaluation-count comparison.

Section 5.1 now describes how the cores are allocated to each configuration. CL distributes the 10 resampling iterations across 10 cores and evaluates 44 configurations in parallel. ACBO and ADBO run these iterations sequentially on one core, allowing up to 448 concurrent configurations. Each evaluation therefore represents a complete 10-fold cross-validation under all three strategies.

5. [recommended] Add a short paragraph on the shared Redis store as a contention point, single point of failure, and source of memory growth at scale.

The shared Redis server is a single point of failure: coordination stops while it is unavailable. Worker or manager failures, however, do not erase the task state held in Redis. Redis can also be configured to save data to disk.

As more workers access Redis and more results are stored, contention and memory use can increase. In our benchmarks, the largest archive contains approximately 16,000 tasks and occupies a few tens of megabytes; the 448 workers complete at most 27 evaluations per second, each requiring a small number of database commands.

We have not added the suggested paragraph because our benchmarks do not show that contention or memory capacity limits performance at the tested scale. Larger benchmarks would be needed to determine the scale at which Redis contention or memory capacity could impact performance. Redis allows up to 10,000 simultaneous connections in its default configuration.

6. [recommended] Add an explicit single-run caveat to the evaluation-count and final-error results in Table 2 and Table 6.

We reran the benchmark with five replications per task-algorithm combination. Tables 2 and 6 now report means and standard deviations. These summaries show the variation in evaluation counts and final errors across runs.

7. [optional] An asynchronous random-search baseline on the same tasks would isolate how much of ADBO's utilization advantage is specific to Bayesian optimization rather than to asynchronous scheduling alone.

We have not added this now (we are in general not aginst adding more methods), because we politely do not see what this would achieve (in the context of this paper).

a) The paper is about rush, as a computational async infrastructure, which might be especially needed / convenient in the context of HPO.

b) It is not about a general comparison of sync / async optimizers and their general pros and cons. This is why we mainly focus on the technical aspect of "cpu utilization" (and less on "classif error" / quality which we mainly report for completeness sake).

c) Now, the reviewer would like cpu util be reported for async RS, but … that would be ~ 100% by construction? You can argue that a (scientific) comparison of this method vs a sync BO technique (in the case of heterogenous runtimes) is interesting, but this is exactly what we

explained in b) is not the focus of this paper and somewhat out of scope? In our opinion this would also require a much larger / complex benchmark – if done properly.

Reviewer 2

1. [required] Using the Table 6 decomposition, report how much of ADBO's counted CPU time is objective evaluation versus surrogate refitting, or qualify the sentences stating that higher utilization translates directly into more configurations and better performance, so utilization is not read as optimization efficiency.

We agree that showing the share of time spent on learner training is useful. We added a learner utilization column to table 6 and describe the learner utilization in section 5.2. Section 5.2 also mentions now that the higher utilization did not translate into lower final error but rather into finding the optimal configurations earlier.

2. [required] State next to the comparative results that each task-algorithm combination is run once, and limit the evaluation-count and final-error comparisons to qualitative statements; keep the utilization contrast, which is large enough to hold.

We reran the benchmark with five replications per task-algorithm combination. Sections 5.1 and 5.2 state the replication count. Tables 2 and 6 report means and standard deviations for utilization, evaluation counts, and final classification errors. The utilization advantage persists across all replications. Sections 5.2 and 6 report similar final errors and describe the small observed differences. They do not claim a general advantage in final optimization performance.

3. [recommended] Add a throughput metric (completed objective evaluations per CPU-second) alongside utilization, so evaluation efficiency is visible separately from core busyness.

If the reviewer insists, we could do that (with a potential footnote / warning in the paper), but would argue against this (now, here). a) Asking how much overhead a certain optimizer uses / produces is a valid question. And directgly comparable (per task). Of course, optimizers with a high overhead can still perform better, when they eval at "smarter" points, the whole design paradigm of BO, but it is interesting to compare how much this incurs. Table 6 in the appendix now makes this clearly visible. b) "Throughput" could potentially be misleading as this depends now also on the runtime of the points being selected (which is usually being heterogenous).

–> We can do this, with such an explanation, but worry that this might send a wrong message.

4. [recommended] Reconcile the version mismatch (text: rush 1.2.0; Zenodo record title: rush 1.1.1) and document the RNG seeding used in the benchmark scripts.

Version 1.1.1 in the Zenodo title referred to the benchmark repository, not the `rush` package. We reran all benchmarks with `rush` 1.3.0. The record name "benchmark 2026 rush 1.3.0" identifies the package version used now (https://zenodo.org/records/22899537). Section 5.1 states now that fixed random seeds were used. It also describes the separately seeded initial designs shared across strategies and the fixed OpenML cross-validation splits. Fixed seeds do not ensure exact reproducibility in asynchronous execution. The order of results depends on timing (Mentioned in Section 5.1 now).

5. [recommended] Specify the surrogate model and acquisition function used for each of CL, ACBO, and ADBO in the LightGBM benchmark.

Section 5.1 now gives the configuration for all three strategies. They use the `mlr3mbo` defaults: a random forest surrogate, a confidence bound acquisition function, and local search for acquisition

optimization.

6. [optional] A direct comparison against DeepHyper or Optuna would settle how rush stands relative to the established Python decentralized frameworks.

A comparison with DeepHyper and Optuna would help assess `rush` across languages. This comparison is beyond the scope of the present software paper. It is a possible topic for a future benchmark study.

Reviewer 3

1. [required] Reconcile the version discrepancy between the text (rush 1.2.0) and the Zenodo artifact title ("rush 1.1.1"). State the exact version that produced Tables 1, 2, and 3, and archive that version alongside the paper.

Version 1.1.1 referred to the benchmark repository, not the `rush` package. We reran the benchmarks with `rush` 1.3.0. The record name "benchmark 2026 rush 1.3.0" identifies the package version used now (https://zenodo.org/records/22899537).

2. [required] Report the full software environment for both benchmark sections, including R version, Redis server version and build, and mirai, redux, and ranger versions. Provide a captured environment such as sessionInfo output or a lockfile.

Section 4.1 now reports R 4.6.0 and Redis 8.10.1 from conda-forge. The benchmark repository includes `setup_hpc.sh` to install these versions. The `renv.lock` file records the whole R package environment.

3. [required] Document seed and RNG handling for the BO benchmarks, and state explicitly that each task-algorithm combination was run once, or qualify the utilization figures as single-run measurements.

We reran the benchmark with five replications per task-algorithm combination. Tables 2 and 6 report means and standard deviations. Section 5.1 states now that fixed random seeds were used. It also describes the separately seeded initial designs shared across strategies and the fixed OpenML cross-validation splits. Fixed seeds do not ensure exact reproducibility in asynchronous execution. The order of results depends on timing (Mentioned in Section 5.1 now).

4. [recommended] Add a short description of how concurrent Redis operations are kept conflict-free (single-threaded execution, transactions, or Lua scripts), since the paper asserts conflict-free registration without giving the mechanism.

Section 2.7, on data storage, now describes the concurrency mechanism. Individual Redis commands execute on a single thread, making operations on the task sets and lists atomic. Operations involving several keys are grouped into `MULTI`/`EXEC` transactions that execute without interruption by other commands. For example, `$push_running_tasks()` adds task keys to the running-task set and the list of all tasks in a single transaction. Potentially competing state transitions, including those performed by `$finish_tasks()` and `$fail_tasks()`, use atomic Lua scripts to ensure that a task is not recorded in two states simultaneously.

5. [recommended] Add a caveat to the utilization results noting that, per Table 6, most of ADBO's Toptimization is redundant per-worker surrogate fitting, so high utilization does not by itself indicate more useful objective throughput.

We added a learner utilization column to table 6 and describe the learner utilization that separates evaluation utilization from the CPU utilization in section 5.2.

However, we believe that surrogate fitting is not "redundant" in the context of ADBO. All three strategies fit one surrogate per proposed configuration and this is part of the optimization process rather than overhead. We think this is likely a (partial) misunderstanding, but also a subtle point, so let us explain this here carefully. Let's start with the metric U. We measure the time where CPUs are active (surrogate fitting, acquisition optimization, and learner training, summed over workers) divided by the complete time of an optim trace. Is this is $< 100\%$ CPUs must have been idle. And compute resources have been "underused", which is bad. We think nobody is disagreeing here. What we take from the editors criticism (repeated by multiple reviewers) is: If this is close to 100% this is not necessarily "ideal" *if* computation is redundantly / unnecessarily replicated i.e. the "redundant" fitting of the surrogates on the archive. But the whole point of ADBO is that this surrogate fitting / acquisition function optimization is *not* redundant as it does not operate on the same state! Let's go through a few examples. Let's take 2 workers. Worker 1 reads the archive. It fits a surrogate and then proposes a point. This point is evaluated. We wait. While we wait worker 2 reads the archive. This now includes the proposed point of W1 – with an imputed fitness values. State is different. Now it could happen that W2 reads the archive while W1 still proposes. Even then the state or proposal differs: a) The acquisition function is randomized (random lambda parameter in LCB per worker) and there is randomness in the random forest and acquisition function optimizer. b) Even the training data of the surrogate model is most of the time different. We can – in a multi-worker setup – likely assume that SOME points are under eval currently. These are also then imputed randomly. We added a footnote to section 5.1 explaining why surrogate fitting in ADBO is not redundant.

We checked how often the workers fitted a surrogate on the same archive with no running points. In this case the workers would indeed see the same archive because nothing is imputed.
In the first iteration of each worker, this happens often because the workers see only the initial design. But each time a different configuration was proposed due to the randomness in the acquisition function. After the first iteration, we found one case in the 20 ADBO runs where four workers saw the same archive with no running points (First replication of ADBO on credit-g). So redundant work is unlikely to happen during the optimization process.

6. [recommended] Provide a small-scale, local-Redis reproducible example (a few workers) so readers without HPC access can reproduce the utilization pattern qualitatively.

We would in general do that, but we do not exactly see how this could generate the desired / a reasonable result: A local example with a few workers can demonstrate the functionality of `rush` / technical features of rush. These are already available in our package vignette, so people can quickly play around with the package. But a local benchmark would likely not reproduce the utilization advantage observed in our benchmarks. This advantage arises when the central controller cannot propose configurations fast enough to keep workers occupied. With few workers, the controller will keep pace with demand. A small example would therefore not provide a representative reproduction of the reported utilization pattern.

7. [optional] A direct comparison against DeepHyper or Optuna would help position rush across languages, though this can reasonably be deferred to future work.

A comparison with DeepHyper and Optuna would help assess `rush` across languages. This comparison is beyond the scope of the present software paper. It is a possible topic for a future benchmark study.

Extra

1. When repeating the BO benchmarks, we observed delayed worker startup for the `kddcup09_appetency` task. The main cause was the transfer of task data to each worker over the LRZ network. We reduced this overhead by storing the data on a shared file system through the existing `large_objects_path` option. Workers on the same node can then use cached file reads. This reduces repeated network transfers. The approach requires a shared file system accessible to all worker nodes. It can reduce startup time when network transfer is a bottleneck. With sufficient network bandwidth, the benefit may be small. We explain the `large_objects_path` option in section 2.7 and 6 now.
2. rush 1.3.0 introduced profiles. We explain them shortly after they are printed with `$worker_info`