Agentic Compactness

Measuring the Cost of Machine Learning Workflows in R

Imad El Badisy

Mohammed VI Center for Research and Innovation (CM6RI), Rabat, Morocco

R+AI 2026 · 10 November 2026



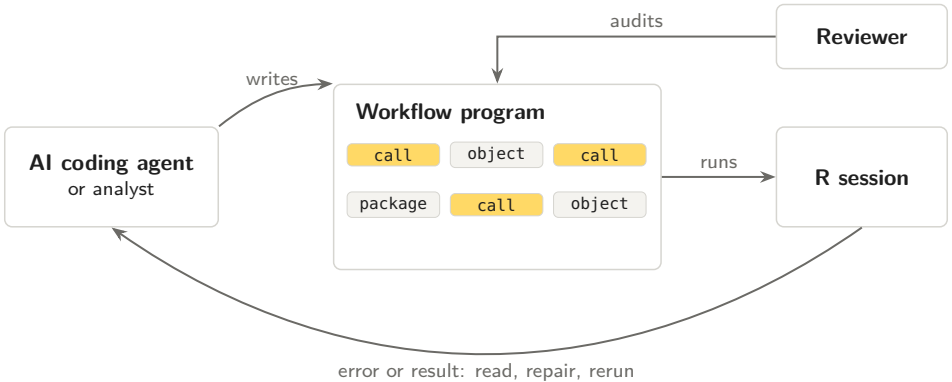
About me



- Health data scientist and computational methodologist, Mohammed VI Center for Research and Innovation (CM6RI), Rabat
- PhD, Aix-Marseille University (SESSTIM): missing data and machine learning for survival analysis
- Open-source software for machine learning and computational biostatistics; several R packages on CRAN, including `funcml`
- Research: scientific machine learning, survival analysis, missing data imputation, agentic data science

elbadisyimad.com · github.com/ielbadisy

What an agent has to get right



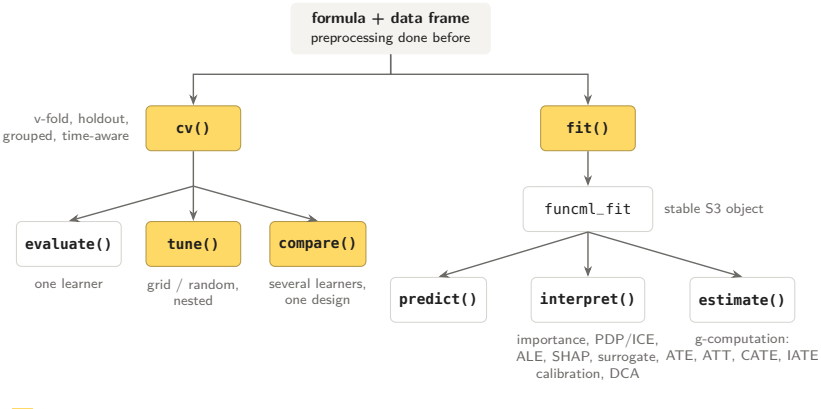
Every framework call, object and package is something to recall, keep consistent and fix. **Compactness** = how few of these pieces a workflow needs for the same validated result.

Compactness is measured with five counts on the analysis code

Counted: resampling, comparison or tuning, preprocessing, fit, prediction.
Not counted: data loading, the split and scoring, which all three share.

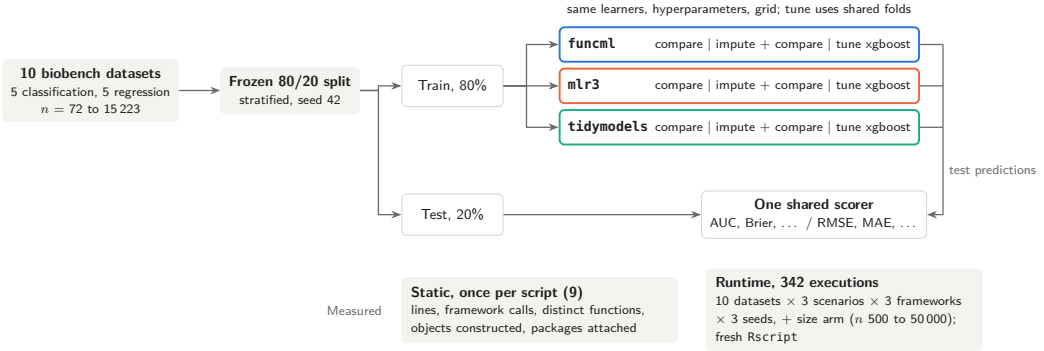
1. **Lines of code**: non-blank, non-comment
2. **Framework calls**: functions exported by the framework (and $ methods for R6)
3. **Distinct framework functions**
4. **Objects constructed**: tasks, learners, resamplings, recipes, workflows
5. **Packages attached** beyond R's six defaults and the data package

The case study: `funcml`, seven verbs on formulas and data frames



No task objects, learner containers, workflow graphs or recipes.

Same statistics, three programs: only the workflow differs



Fixed learners: glmnet (lasso, penalty 0.01), ranger (500 trees), XGBoost (100 rounds, depth 6, rate 0.3); tune grid of 12 points.

Disclosure: I develop `funcml` and wrote all three workflows.

Ten health datasets from biobench, 72 to 15,223 records

Dataset	Task	Outcome	Summary	*n*	Predictors
covidtest	Class.	SARS-CoV-2 test positive	6% Yes	15,223	9 (60)
nhanes	Class.	Obesity (BMI $>$ 35)	15% Yes	6,445	13 (17)
opt	Class.	Pregnancy ended before 37 weeks	12% Yes	812	56 (59)
pima2	Class.	Diabetes	35% Yes	768	8
glow500	Class.	Fracture in first year	25% Yes	500	11 (12)
aidsdelay	Regr.	AIDS cases reported	mean 11 (SD 25)	570	5
birthwt	Regr.	Birth weight (g)	mean 2945 (SD 729)	189	8
laryngoscope	Regr.	Intubation time (s)	mean 37 (SD 21)	99	6
smartpill	Regr.	Whole-gut transit time (h)	mean 58 (SD 100)	94	18
anorexia	Regr.	Weight after treatment (lb)	mean 85 (SD 8)	72	2

6 of 10 have missing values. Predictors column: count after dummy coding (raw). Predictors passed in alphabetical order. One fixed 80/20 split per dataset, stratified for classification. Speed test: 8 simulated datasets, 500 to 50,000 rows.

The same tuning: 2 framework calls, against 17 and 16

funcml 8 lines · **2** framework calls

```
tuned <- tune(train, form, model = "xgboost", grid =
    TUNE_GRID,
                resampling = group_cv(v = CV_FOLDS,
                   group = folds),
                metric = if (is_classif) "auc" else
                   "rmse", seed = seed, nthread = 1)
pred <- if (is_classif) {
  predict(tuned$fit_best, newdata = test, type =
     "prob")[, "Yes"]
} else {
  predict(tuned$fit_best, newdata = test)
}
```

mlr3 26 lines · **17** framework calls

```
set.seed(seed)
tt <- if (is_classif) {
  as_task_classif(train, target = outcome, positive = "Yes")
} else {
  as_task_regr(train, target = outcome)
}
rs <- rsmp("custom_cv")
rs$instantiate(tt, f = factor(folds))
learner <- lrn(if (is_classif) "classif.xgboost" else "regr.xgboost",
    nthread = 1)
if (is_classif) learner$predict_type <- "prob"
at <- auto_tuner(
  tuner = tnr("design_points", design =
     data.table::as.data.table(TUNE_GRID)),
  learner = learner,
  resampling = rs,
  measure = msr(if (is_classif) "classif.auc" else "regr.rmse"),
  search_space = ps(nrounds = p_int(1, 1000), max_depth = p_int(1,
     20),
                    eta = p_dbl(0, 1))
)
at$train(tt)
new_tt <- if (is_classif) {
  as_task_classif(test, target = outcome, positive = "Yes")
} else {
  as_task_regr(test, target = outcome)
}
prediction <- at$predict(new_tt)
pred <- if (is_classif) prediction$prob[, "Yes"] else
   prediction$response
```

tidymodels 21 lines · **16** framework calls

```
set.seed(seed)
spec <- set_mode(
  set_engine(boost_tree(trees = tune(), tree_depth = tune(),
     learn_rate = tune()),
             "xgboost", nthread = 1),
  if (is_classif) "classification" else "regression")
wf <- workflow(recipe(stats::reformulate(".", outcome), data =
    train), spec)
grid <- data.frame(trees = TUNE_GRID$nrounds, tree_depth =
  TUNE_GRID$max_depth,
                   learn_rate = TUNE_GRID$eta)
metric <- if (is_classif) "roc_auc" else "rmse"
metrics <- if (is_classif) metric_set(roc_auc) else metric_set(rmse)
rs <- manual_rset(lapply(seq_len(CV_FOLDS), function(k) {
  make_splits(list(analysis = which(folds != k), assessment =
     which(folds == k)), train)
}), paste0("Fold", seq_len(CV_FOLDS)))
res <- tune_grid(wf, rs, grid = grid, metrics = metrics)
best <- select_best(res, metric = metric)
final <- fit(finalize_workflow(wf, best), data = train)
pred <- if (is_classif) {
  predict(final, test, type = "prob")$.pred_Yes
} else {
  predict(final, test)$.pred
}
```

Highlighted = counted framework calls. Taken from the measured scripts.

In `funcml`: one `tune()` call, then the standard `predict()`

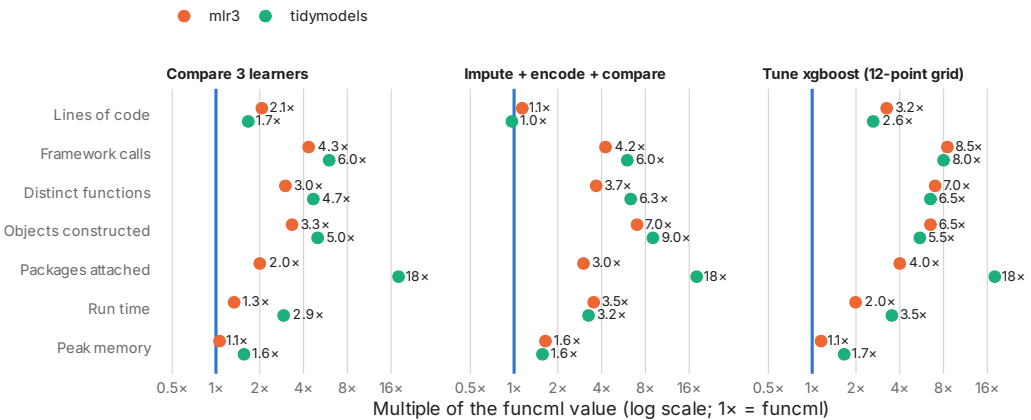
```
tuned <- tune(train, form, model = "xgboost", grid = TUNE_GRID,
              resampling = group_cv(v = CV_FOLDS, group = folds),
              metric = if (is_classif) "auc" else "rmse", seed = seed, nthread = 1)
pred <- if (is_classif) {
  predict(tuned$fit_best, newdata = test, type = "prob")[, "Yes"]
} else {
  predict(tuned$fit_best, newdata = test)
}
```

`funcml`: fewest calls and packages in every scenario

	Code surface (counts)					Execution (median of 30 runs)		
Framework	Lines	Calls	Functions	Objects	Packages	Load, s	Total, s	Peak, MiB
Compare 3 learners								
`funcml`	**15**	**3**	**3**	**3**	**1**	**0.01**	**1.5**	**259**
`mlr3`	31	13	9	10	2	0.22	2.0	274
`tidymodels`	25	18	14	15	18	2.78	4.5	404
Impute + encode + compare								
`funcml`	29	**4**	**3**	**2**	**1**	**0.01**	**1.5**	**256**
`mlr3`	33	17	11	14	3	0.28	5.4	421
`tidymodels`	**28**	24	19	18	18	2.66	5.0	402
Tune xgboost (12-point grid)								
`funcml`	**8**	**2**	**2**	**2**	**1**	**0.01**	**1.8**	**231**
`mlr3`	26	17	14	13	4	0.30	3.6	266
`tidymodels`	21	16	13	11	18	2.81	6.3	384

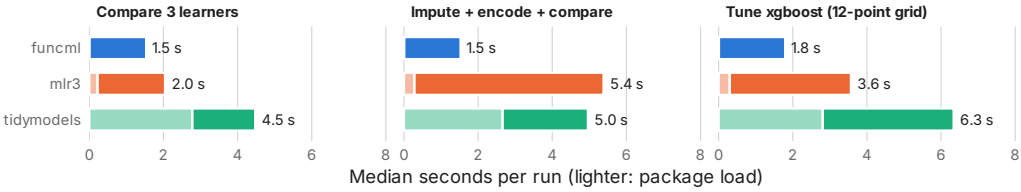
Bold = lowest. Medians of 10 datasets × 3 seeds; fresh R process per run, single-threaded.

`mlr3` and `tidymodels` need 4 to 8.5 times as many framework calls

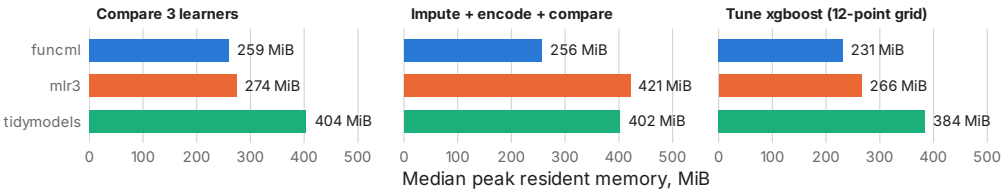


`funcml` runs fastest and uses the least memory

Wall-clock time



Peak memory



As data grow, the framework matters less



Compare scenario, simulated data. At 50,000 records: 307 s (`funcml`), 309 s (`mlr3`), 315 s (`tidymodels`). Model fitting dominates; the fixed load cost fades.

With shared folds, the three programs give the same answer

Scenario	Same selection	Identical score	Max AUC gap	Max RMSE gap (rel.)
Compare 3 learners	12/30	10/30	0.060	7.1%
Impute + encode + compare	13/30	6/30	0.029	7.1%
Tune xgboost (12-point grid)	30/30	30/30	0	0

- **Tune** (shared folds): identical choice and test score in every run, although 7 of the 12 grid points won at least once.
- **Compare scenarios** (each framework draws its own folds and random-forest stream): the small gaps come from resampling, not from the framework.
- Parity is the goal: the compact program loses nothing; it does not fit better.

Making the frameworks agree exposed five failure points

- **E1** `tidymodels`: `metric_set(if (...) roc_auc else rmse)` names the metric after the expression, so `select_best()` fails. Fix: one `metric_set()` per branch.
- **E2** `tidymodels`: parsnip 1.3.3 passes `ntreelimit`, removed in xgboost 3.x: every fit failed, hidden behind E1. Fix: update parsnip and tune.
- **E3** `mlr3`: Tasks sort predictors alphabetically; XGBoost breaks split ties by column order. Fix: alphabetical column order everywhere.
- **E4** all: Each framework drew its own folds: different choices on 2 of 5 classification datasets. Fix: shared folds.
- **E5** `funcml`: Its ranger learner ignored `num.threads` (CPU 160% vs 100%), so it looked up to twice as fast. Fix: one thread, full rerun.

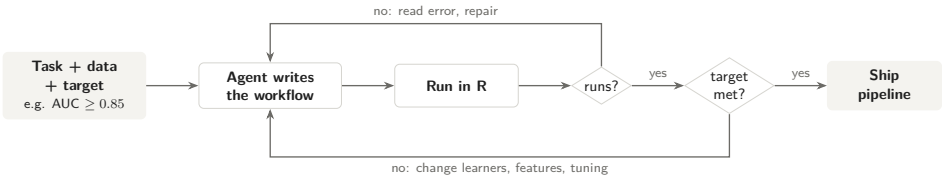
E1 and E2 showed errors that pointed away from the real cause. Each issue has a reprex in the code repository.

What this does not show

- **One author, one implementation per framework**, and that author develops `funcml`. Another expert could write shorter `mlr3` or `tidymodels` code.
- **Counts are syntax-sensitive.** They proxy the surface; they are not a validated measure of effort or maintainability.
- **`funcml` has no preprocessing verbs.** With imputation its script is as long as the others, though it still needs the fewest calls. A compact interface moves work; it does not remove it.
- **Ten small-to-medium tabular health datasets, one split each**, fixed learners; no framework defaults, no ranking on predictive performance.
- **No agent in the loop yet.** First-run success, repairs and tokens are not measured here.
- **Timing and memory** depend on hardware and package versions.

Next: put the agent in the loop

PLANNED STUDY, NOT PART OF THESE RESULTS



Measured per framework: first-run success · number and type of repairs · tokens · wall-clock and compute cost · performance of the shipped pipeline

Same three frameworks, same task and target. Does a smaller interface let an agent reach a working, accurate pipeline with fewer failed attempts and less cost?

Take-home

1. Compactness can be **measured from the program**: five static counts, plus run time and memory.
2. Framework calls and packages per scenario:

`funcml`	**2–4 calls**	**1 package**
`mlr3`	13–17 calls	2–4 packages
`tidymodels`	16–24 calls	18 packages

3. Fewer calls cost nothing: lowest run time and memory, and **the same tuned model** once folds are shared.
4. Making the frameworks agree exposed five failure points: exactly what an agent must get right.

Report compactness together with a check that the programs give the same answer, and a record of every issue found.

Thank you

Questions?

Code & results	`https://github.com/ielbadisy/agentic-compactness-r-ai-2026`
`funcml`	`https://CRAN.R-project.org/package=funcml`
Website	`https://elbadisyimad.com`
GitHub	`https://github.com/ielbadisy`

Release v1.0.0 holds the workflows, the runner, all 342 run results and the reprex for issues E1 to E5.