METHODS NOTE · R SOFTWARE FOR BIOMEDICAL MACHINE LEARNING

Leakage defence in layers How splitGraph, fastml and bioLeak divide the work of honest performance estimation

Selçuk Korkmaz

26 September 2026

`splitGraph 0.4.0` `fastml 0.7.10` `bioLeak 0.3.8`

ABSTRACT

Information leakage in cross-validation can come from the *structure* of the data (repeated samples, related subjects, shared batches), from the *procedure* (preprocessing estimated on the full data) or from *undeclared* artefacts that only the data reveal (re-deposited specimens, proxy features). Each source needs a different kind of information to control: the study design, control over execution order, and the data themselves. We describe three R packages built around these three kinds of information. splitGraph turns sample metadata into a typed dependency graph and derives the grouping that must not be split. fastml trains and compares models with the preprocessing steps it declares re-estimated inside each fold. bioLeak audits a fitted design for near-duplicates, batch–fold confounding, proxy features and non-random signal.

We specify the interfaces that let the packages work as one pipeline, including a feedback path from the audit back to the graph. In a 30-replicate simulation of a family-structured cohort with repeated visits, batch effects and hidden re-deposited specimens, naive cross-validation overstated the external AUC by 0.27 (logistic regression) and 0.24 (random forest). Each of four successive steps, taken in this order, lowered the estimate by a statistically significant amount (paired Wilcoxon $p < 10^{-5}$ at every step). The full pipeline left a bias of −0.012 and −0.016, indistinguishable from zero at this precision. The steps are not additive: upsampling before otherwise honest folds added 0.18–0.24 to the AUC when the copies scattered across folds, against 0.02–0.05 under row-wise folds, and it turned pure noise into a cross-validated AUC of 0.97. The audit recovered all 720 injected duplicate pairs with no false positives. Honest estimates were noisier: with 40 independent families, cross-validation selected the externally better model in 70% of replicates, against 90–93% for the leaky designs, which chose random forest in every replicate.

Keywords: data leakage · cross-validation · grouped resampling · dependency graphs · permutation tests · reproducibility

1 Three sources of leakage, three kinds of knowledge

Cross-validation (CV) estimates how well a modelling procedure, trained on samples of about the available size, will perform on new units drawn from the target population [1]. The estimate is honest only when each assessment fold behaves like such new units, so that nothing about them has shaped the model that scores them. Kaufman et al. define leakage as the introduction of information about the prediction target that should not legitimately be available to the model [2]. We use the term in the same spirit for any path by which information about the assessment units reaches the fitted model, including dependence between observations and unsupervised preprocessing. A survey of published critiques found leakage reported in 17 fields, affecting 294 papers [3]. In connectome-based prediction, leakage through feature selection and repeated subjects inflates performance sharply, while other forms have smaller effects [4].

Adapting the taxonomy of Kapoor and Narayanan [3] (its non-independence, preprocessing, feature-selection, duplicate and illegitimate-feature types), we regroup the leakage paths relevant here by the knowledge needed to close them:

1. **Structural leakage.** Assessment observations are statistically dependent on training observations, for example the same subject at another visit, a sibling, or a sample from the same processing batch. This counts as leakage when the intended use is on units that share no such dependence with the training data, such as new patients rather than new visits of known patients [5,6]. In genomics, relatedness between training and test individuals is a known source of optimistic prediction accuracy [7]. Closing this path requires knowledge of the *study design*, which lives in metadata rather than in the feature matrix.
2. **Procedural leakage.** A data-dependent transformation (scaling, imputation, class rebalancing, feature selection) is estimated on data that include the assessment fold. Bias arises even when the transformation never sees the outcome, and it can then go in either direction [8]. Oversampling before cross-validation is a well-documented case [9], and tuning on the same folds used for reporting is a related one [10]. Closing this path requires control over *execution order*.
3. **Latent leakage.** A dependence between observations, or a feature's link to the label, exists but was never declared. Examples are a specimen re-profiled and deposited under a new identifier [11], or a derived feature that encodes the label [2,3]. The sample metadata do not record it, and a correct pipeline carries it along faithfully. Within the analysis it can be revealed only empirically from the data, or from provenance outside the sample table.

These kinds of knowledge sit in different places, so a tool built around one of them cannot supply the others. A grouping derived from metadata cannot group samples whose relationship is absent from the metadata. A pipeline that refits preprocessing inside every fold cannot know which rows belong together unless it is told. An empirical audit can flag suspicious similarity, but it needs a declared design to interpret what it finds. We argue that these are complementary layers of one defence, and that the interfaces between them deserve as much care as the layers themselves.

This note makes three contributions. It states the responsibilities of splitGraph [12], fastml [13,14] and bioLeak [15] precisely enough that a reader can see what each one provides and what it cannot see (Section 2, Table 1). It specifies a working composition, including a feedback path from audit to design (Section 3). And it quantifies, for one data-generating process and one ordering of the layers, what each layer contributes to an honest estimate (Section 4). All computations use R [16].

2 Three packages, three responsibilities

2.1 splitGraph: declaring what must not be split

splitGraph represents dataset structure as a typed, directed, heterogeneous graph. Samples are nodes, and so are the entities they depend on. The schema is closed: 11 node types (Sample, Subject, Batch, Study, Timepoint, Assay, FeatureSet, Outcome, Site, Region, Platform) and 17 typed relations, such as `sample_belongs_to_subject`, `sample_processed_in_batch`, `timepoint_precedes` and the thresholded, undirected `subject_related_to` and `sample_adjacent_to`. `graph_from_metadata()` builds the graph from a one-row-per-sample table, and `validate_graph()` checks it. Validation enforces single-target rules (for example, at most one subject per sample), acyclic time precedence consistent with `time_index`, and the absence of dangling edges. It also reports leakage-level issues, such as the advisory `repeated_subject_samples` and the warning `subject_cross_study_overlap`. Figure 1 shows the graph for a toy cohort, drawn with the package's own `plot()` method.

`derive_split_constraints()` reduces the graph to a partition of samples:

- **Direct modes** (subject, batch, study, time, site, region, platform, assay). The group of a sample is the target of its single edge of that relation. A sample with no such edge becomes its own singleton group (group_id `<mode>:unlinked:<sample_id>`), which is recorded in the constraint's metadata warnings but raises no R warning. A sample with several is an error, except in subject mode under the `allow_multi_subject_samples` override, where the first listed subject is used.
- **Pairwise modes.** The groups are connected components. For relatedness mode they are components of the `subject_related_to` edges, which `relatedness_edges_from_kinship()` builds from kinship coefficients above a threshold, together with each sample's subject edge, so a subject's samples and its relatives' samples share a group. For spatial mode they are components of the thresholded `sample_adjacent_to` edges. Dependence is thus closed transitively: if P1 is related to P2 and P2 to P3, all three share a group even when P1 and P3 are unrelated. Lowering a kinship threshold or widening a spatial radius can only merge groups.
- **Composite mode** combines relations in two ways. `strict` takes connected components over the union of the chosen relations, and `rule_based` assigns each sample to the highest-priority relation for which it has an assignment.

The derivation is deterministic, with no randomness and no fold construction. It returns a `split_constraint`, which `as_split_spec()` converts into a `split_spec`. That specification is a tool-agnostic contract. It holds a sample table with `group_id`, optional block, time and ordering columns, a recommended resampling family, and provenance (package versions, thresholds, a derivation timestamp). It serialises to JSON under a published schema (version 0.3.0). It is read by bioLeak's `as_leaksplits()`, by rsample through adapters given in the package vignette, and by a shipped Python reader that feeds scikit-learn's `GroupKFold`.

Version 0.4.0, used throughout this note, adds several features:

- the constraint derivations run in linear time (building pairwise edges does not: `spatial_edges_from_coords()` forms a full distance matrix, and a kinship matrix is expanded to all pairs);
- composite `via` can include the pairwise relations;
- `add_edges()` and `combine_graphs()` compose graphs;
- kinship can be supplied as a matrix, with threshold provenance recorded in the specification;
- an optional `stratum` annotation carries the outcome for downstream stratification;
- `SummarizedExperiment` objects are accepted as input;
- every error is a classed `splitgraph_error` with a `code` field, which holds a machine-readable code for schema, reference, ambiguity, validation and I/O failures, is `invalid_argument` for an invalid choice of argument or a malformed `validation_overrides` at graph construction, and is `NA` for other plain argument checks.

splitGraph's R code deliberately does not create folds, balance classes, stratify or fit anything; its shipped Python reader only wraps scikit-learn's `GroupKFold` and `StratifiedGroupKFold` [17]. Its guarantee is conditional. If a consumer never places one `group_id` on both sides of a split, then no dependence carried by the relations the chosen mode uses crosses the split. In relatedness mode those are the subject and kinship edges; in a strict composite they are the union of the `via` relations. There are two exceptions: under the `allow_multi_subject_samples` override only each sample's first subject is honoured, and in time mode the grouping keeps each timepoint together, so look-ahead must be prevented by respecting `order_rank`. Dependence through other declared relations, such as batch, can still cross and must be handled as blocks or tested.

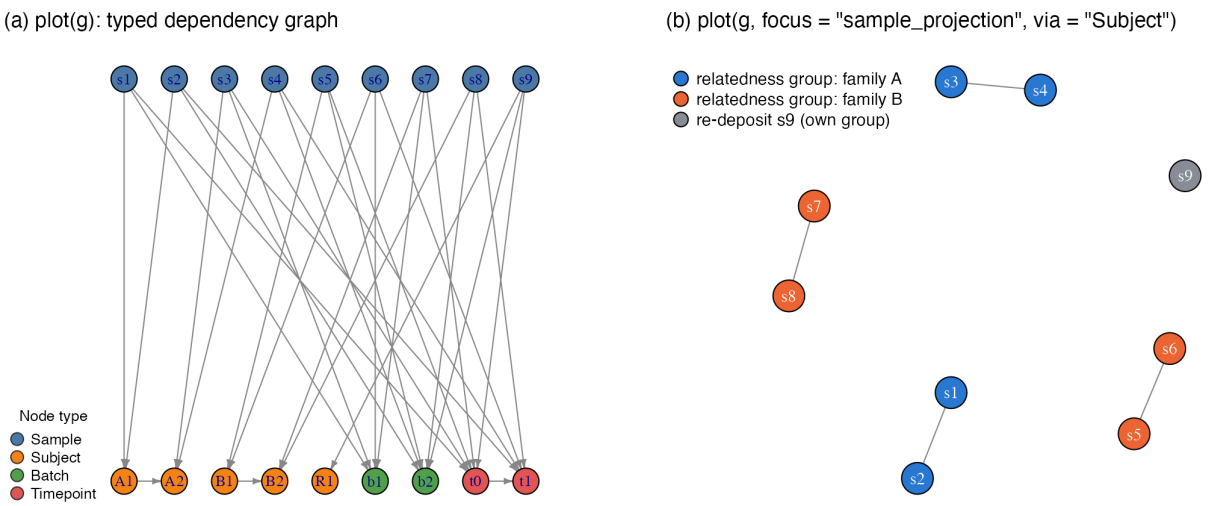

Figure 1. A toy cohort drawn with splitGraph's `plot()`. Two families, each with two siblings seen twice, give eight samples in two batches. Sample s9 is a copy of s1 deposited under a new subject ID (R1). **(a)** The typed layout: samples on top; subject, batch and timepoint targets below, with the kinship edges A1–A2 and B1–B2 and the precedence edge t0→t1. Kinship is symmetric, but splitGraph stores the edge with a direction, so it is drawn as an arrow. **(b)** `focus = "sample_projection", via = "Subject"` joins samples of the same subject, giving five subject groups. Colour, which the figure script passes to `plot()`, shows the relatedness grouping (three groups): transitive closure merges each pair of siblings into one family group, while s9 stays alone because nothing in the metadata links it to s1.

2.2 fastml: executing without procedural leakage

fastml trains, tunes and compares models through one call, mostly using engines from the tidymodels ecosystem. Its registry offers 15 classification, 14 regression and 11 survival algorithm options (9 distinct survival learners, since two survival names are aliases). Its defining design choice is that the preprocessing arguments (`impute_method`, `encode_categoricals`, `scaling_methods`, `balance_method`) compile to an *untrained* recipe, which is estimated only on each split's analysis rows. The one step outside the folds is a class-wise zero-variance filter: when `discrim_quad` is requested and no recipe is supplied, fastml drops from the whole training set, for every algorithm in the call, each numeric predictor that is constant within an outcome class.

GUARDED RESAMPLING IN FASTML (FOLD LOOP IN FASTML_GUARDED_RESAMPLE_FIT)

```
for each split k = 1 … K:
    stop if a non-bootstrap analysis set contains every training row   # sanity check only
    R_k ← prep(untrained recipe, analysis_k)        # imputation, scaling, upsampling …
    M_k ← fit(model, bake(R_k, analysis_k))
    m_k ← metric(M_k, bake(R_k, assessment_k))   # assessment only transformed
report mean and SD of m_1 … m_K                  # folds with an undefined metric are dropped
refit on all training data (train_models); score the test set (fastml)   # outside this loop
```

The loop does not check that analysis and assessment sets are disjoint; it trusts the resampling object. Guards reject some configurations that could smuggle outside information into it. A recipe that has already been passed through `prep()` is refused before training, with the message "*Pretrained recipes are not allowed; provide an untrained recipe.*" Recipe steps whose expressions name the global environment or an environment-walking helper (`.GlobalEnv`, `globalenv()`, `parent.frame()`), that carry a function defined in the global environment, or that embed a data frame are also refused. The check is heuristic: it skips a fixed list of step types, mostly those fastml itself adds (dummy, novel and unknown levels, zero-variance, centring, scaling and normalising, median, k-NN and bagged imputation, NA removal, and up- and down-sampling steps), ordinary free variables and values inlined into a step are not detected, and a string such as `"globalenv"` can trigger it.

fastml supports the resampling designs that structural leakage calls for:

- `grouped_cv`, which needs `group_cols`;
- `blocked_cv` and `rolling_origin`, which need a `block_col` on data sorted by it (`blocked_cv`, `rolling_origin` and the time-ordered holdout all cut by row position, so tied values can fall on both sides of a cut);
- `nested_cv`, which needs `outer_folds` and separates tuning from estimation. Without it, a tuned model's CV estimate is computed on the same folds used to select its configuration and is optimistic [10].

It stops rather than relaxing a grouping or ordering constraint that cannot be honoured. Some secondary requests are relaxed with a warning: stratification; the test proportion of a grouped, time-ordered holdout when the nearest admissible cut misses it by more than `test_size_tolerance`; and, outside survival tasks, the fold count of `cv` or `repeatedcv` when it reaches the number of rows (for survival outcomes this is an error). A fold count above the number of groups or blocks is an error. The initial holdout follows the declared dependence:

- whole groups when `group_cols` is given, with `test_size` applied to the number of groups rather than rows;
- the latest rows when `block_col` is given;
- when both apply, the admissible cut nearest the requested test proportion at which no group spans the boundary, stopping if none exists.

Any rsample `rset` built on `train_data` can be supplied through `resamples`, and this is the seam our composition uses. fastml fits the fold models on the data embedded in that `rset` and the final model on `train_data`, without checking that the two agree.

fastml cannot undo preprocessing done before the call, and it does not know which rows are dependent unless told. `group_cols` and `block_col` govern the folds only under `grouped_cv` or `nested_cv` and under `blocked_cv` or `rolling_origin`; with the default `resampling_method = "cv"` they govern the holdout alone, and the folds split groups without a warning. Conversely, dependence declared only through `resamples` governs the folds but not the holdout. The study in Section 4 therefore supplies an external `test_data` rather than letting fastml split `data` itself. Section 4.4 isolates the effect of preprocessing done before the call.

2.3 bioLeak: auditing what remains

bioLeak covers both prevention and detection, and its auditing layer is what the other two packages lack.

`make_split_plan()` builds subject-grouped, batch-blocked, leave-one-study-out and time-series plans, with horizon, purge and embargo options for time series. It also builds *combined* plans. There, primary groups linked through a constraint axis are merged into components for fold assignment, and training rows that share any constraint-axis level with the test fold are excluded. When there are fewer components than folds, bioLeak warns and assigns groups at random.

`fit_resample()` fits learners on these plans with a guarded preprocessing chain estimated on each training fold: winsorisation, imputation, normalisation, filtering and feature selection, with no class rebalancing step. It drops the outcome and the columns the plan names (group, batch, study, time) from the predictors. At the commit used here, further identifier columns can be excluded with `id_cols`, and ID-like character columns trigger a warning.

`audit_leakage()` then examines the fitted design with four diagnostics:

- **Permutation gap.** In refit mode with an unrestricted shuffle this is the label-permutation test of Ojala and Garriga (their Test 1) [18]; the restricted and block variants below extend it, and the gap is bioLeak's effect-size summary.
  - For an observed metric T_{obs} computed on pooled out-of-fold predictions and B null draws T_b, the gap is T_{obs} − mean(T_b). The p-value is $(1 + \#\{T_b \geq T_{obs}\}) / (1 + B)$, which is never zero [19]; for the block-resampled time-series null it is approximate.
  - In refit mode each null draw refits the pipeline on permuted outcomes over the same splits. The draws can respect the design: whole outcome vectors are swapped between groups of the plan's group column that have equal size (the rows of groups whose size is unique are pooled and shuffled together); labels are shuffled within batch or study; time series are resampled in blocks (circular blocks by default [20], or the stationary bootstrap [21]), so their null is a block-resampling rather than a strict permutation distribution.
  - The restriction needs the refit data to carry the outcome and the plan's group column under its own name (`plan@info$group`). When `fit_resample()` is given a feature-only frame, supply them through `perm_refit_spec$coldata` , with row names matching the refit data. Otherwise each row is treated as its own group and the null is an unrestricted shuffle.
  - The folds stay fixed at those of the plan. In a null draw where a test fold's permuted labels are all one class, that fold is dropped, so the draw pools fewer predictions than the observed statistic.
  - With `perm_refit = FALSE` the null is always a global shuffle of the pooled predictions.
- **Batch–fold association.** A χ^2 test of the fold × batch table in each repeat, reported with Cramér's V; at the commit used here also with a Holm-adjusted p-value across batch columns and repeats.
- **Proxy-target scan.** Univariate association of every reference feature with the outcome ($|\text{AUC} - 0.5| \cdot 2$, η^2 or $|r|$ for numeric features; Cramér's V or η^2 for categorical ones), flagged at 0.9. A multivariate scan (a cross-validated model on leading principal components and their interactions, with a permutation p-value) runs by default.
- **Near-duplicates.** Cosine similarity of L2-normalised feature rows. The search is exact when $n \leq 500$ and $n \cdot p \leq 1.5 \times 10^6$; otherwise, when the suggested package RANN is installed, it is restricted to each row's 50 nearest neighbours. Pairs at or above 0.995 are flagged, and at most 5,000 pairs are reported. By default only pairs that straddle a train/test split are reported; `duplicate_scope = "all"` reports every pair, with a `cross_fold` indicator.

For comparing a leaky and a guarded pipeline on identical splits, `delta_lsi()` gives three things:

- a Huber M-estimate of the per-repeat difference;
- a BCa interval, when there are at least 10 paired repeats;
- a sign-flip test: exact enumeration up to 15 repeats, Monte Carlo above.

Its evidence tiers depend on the number of paired repeats R: tier A needs $R \geq 20$, and tier D means $R < 5$. `cv_ci(method = "nadeau_bengio", n_train =, n_test =)` reports the corrected resampled-t interval of Nadeau and Bengio [22], in the K-fold form of Bouckaert and Frank [23]. Its variance factor $(1/K + n_{test}/n_{train})$, with K the total number of folds across repeats, approximately corrects for the correlation that overlapping training sets induce. Without `n_train` and `n_test` it silently returns the uncorrected interval, which is also the default `method` .

> **Reading the audit correctly.** A significant permutation gap shows that the pipeline extracts non-random signal under the given splits. It does not show whether that signal is genuine or leaked. The duplicate, batch and proxy diagnostics are leakage-indicative evidence, and each still needs a design-level explanation. A flagged pair may be a declared technical replicate, a batch–fold imbalance is not leakage in itself, and a strong feature may be a real biomarker. The bioLeak documentation adds a complementary caveat [15]: a non-significant test is not proof that there is no leakage.

Table 1. How each layer handles each leakage mechanism. "Declares" means the layer records the structure and derives a grouping, but its R code builds no folds; "Needs a grouping" means the layer honours a grouping only if one is supplied.

MECHANISM	SPLITGRAPH	FASTML	BIOLEAK
Repeated samples per subject	**Declares grouping (subject mode)**	Needs a grouping (grouped_cv)	**Prevents (subject-grouped plan); detects near-copies**
Related subjects (kinship)	**Declares grouping (relatedness, transitive closure)**	Needs a grouping (grouped_cv)	Needs a grouping or a family constraint axis
Batch or site confounding	Declares; batch or site mode	Needs a grouping (grouped_cv)	**Prevents (batch-blocked plan); detects (χ^2, Cramér's V)**
Temporal look-ahead	Declares order; time mode	**Prevents (rolling_origin, except for tied times); blocked_cv does not**	**Prevents (time-series plan with purge, embargo)**
Preprocessing fitted on all data	Out of scope	**Prevents, for the steps it declares**	**Prevents (guarded chain; no rebalancing step)**
Tuning on reporting folds	Out of scope	**Prevents, with nested_cv**	**Prevents (nested tune_resample)**
Undeclared re-deposited specimens	**Cannot see**	**Cannot see**	**Detects (cosine ≥ 0.995)**
Proxy feature encoding the label	**Cannot see**	**Cannot see**	**Detects (target scan)**

3 Composition: interfaces and a feedback path

The three packages meet at explicit seams:

- splitGraph emits a grouping keyed on `sample_id`: `grouping_vector()` of the constraint in R, or the `group_id` column of the serialised `split_spec` for other tools.
- bioLeak turns a grouping into a `LeakSplits` plan and converts that plan to an rsample `manual_rset` with `as_rsample()`.
- fastml accepts an `rset` built on its `train_data` through `resamples`. This hand-off is positional: `as_rsample()` maps fold row indices onto the frame it is given.

The same `LeakSplits` object also drives `fit_resample()` and `audit_leakage()`, so the audit examines exactly the folds on which fastml estimated performance. Its permutation gap, however, is computed for bioLeak's own refit rather than for the fastml models. Figure 2 shows the flow and the one arrow that makes the design iterative.

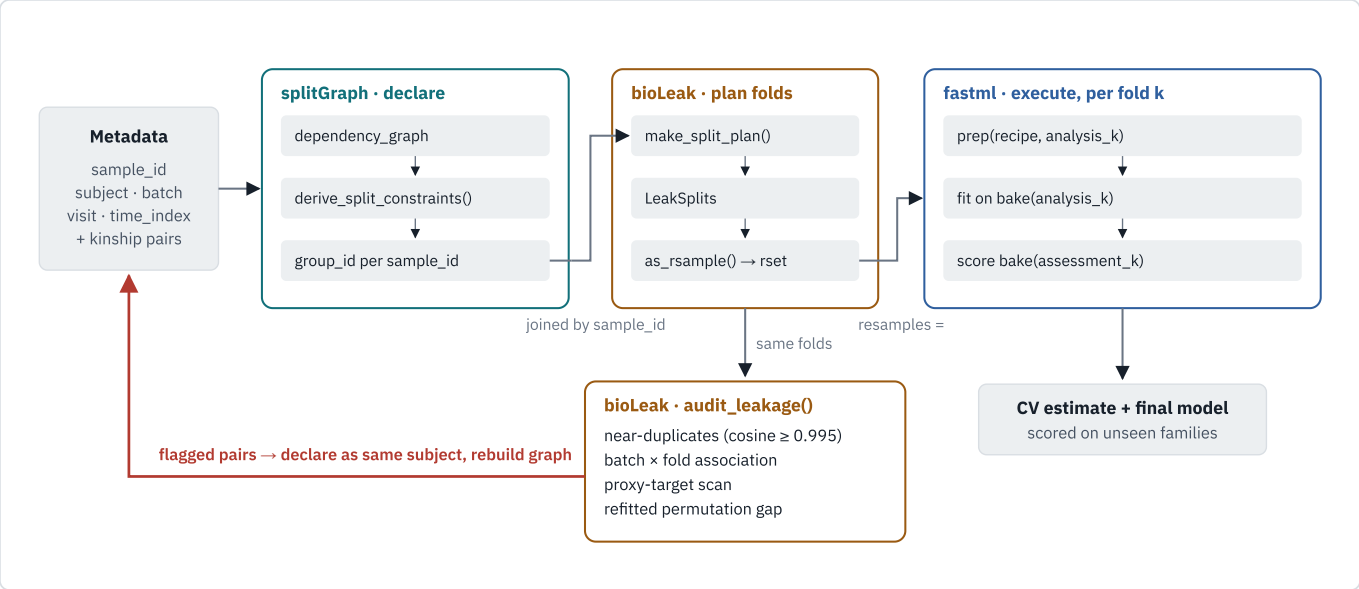

Figure 2. Object flow in the composed pipeline. Structure is declared once (splitGraph), turned into folds once (bioLeak), and then consumed by both the estimator (fastml) and the auditor (bioLeak). The red path is the feedback loop: dependence that only the data reveal is written back into the metadata, so the next derivation severs it.

The core of the pipeline is short. The listing below condenses the code used in Section 4; the audit settings are those of the replicate shown in Section 4.3, while the replicate loop audits with `perm_refit = FALSE` and $B = 50$ because it needs only the duplicate scan. `meta` holds one row per sample, `tr` holds the outcome and features only in `meta`'s row order, `feats` holds the 30 feature names, `ext` is the external cohort, and `glm_spec` is a parsnip logistic-regression specification.

```
# 1  declare structure: subjects, batches, visits and kinship (splitGraph)
g0  <- graph_from_metadata(meta[, c("sample_id", "subject_id", "batch_id",
                                    "timepoint_id", "time_index")])
rel <- relatedness_edges_from_kinship(kin, threshold = 0.125)   # kinship ≥ 0.125
g   <- add_edges(g0, rel)
con <- derive_split_constraints(g, mode = "relatedness")
grp <- unname(grouping_vector(con)[meta$sample_id])              # align by sample_id, drop names

# 2  turn the grouping into folds; one object serves estimator and auditor (bioLeak)
plan <- make_split_plan(data.frame(sample_id = meta$sample_id, y = meta$y, group_id = grp),
                        outcome = "y", mode = "subject_grouped", group = "group_id",
                        v = 5, stratify = TRUE, seed = 1)
rs   <- as_rsample(plan, data = tr)                  # the same frame as train_data below

# 3  estimate with fold-local preprocessing (fastml)
fit <- fastml(train_data = tr, test_data = ext, label = "y",
              algorithms = c("logistic_reg", "rand_forest"), resamples = rs,
              balance_method = "upsample", tuning_strategy = "none", seed = 1,
              store_fold_models = TRUE)           # fold models for the reference in 4.1

# 4  audit the same folds with a restricted refit null (bioLeak)
bf   <- fit_resample(tr, outcome = "y", splits = plan, learner = glm_spec, metrics = "auc")
spec <- bf@info$perm_refit_spec
spec$coldata <- data.frame(y = meta$y, group_id = grp, row.names = rownames(spec$x))
aud  <- audit_leakage(bf, metric = "auc", B = 100, perm_refit = TRUE, perm_refit_spec = spec,
                      coldata = data.frame(batch = meta$batch_id), X_ref = tr[, feats],
                      target_scan_multivariate = FALSE, duplicate_scope = "all")
dup  <- audit_duplicates(aud)                 # pairs (i, j) with cosine ≥ 0.995

# 5  feedback: declare each later member as the same subject as the earlier one, repeat 1-3
j <- pmax(dup$i, dup$j); i <- pmin(dup$i, dup$j)
meta$subject_id[j] <- meta$subject_id[i]
meta$family_id[j]  <- meta$family_id[i]      # the kinship table is rebuilt from families
```

Five composition rules follow from the interfaces as they stand. Each marks a place where a correct component can be wired into an incorrect pipeline.

1. **Keep identifiers out of the predictor frame.**
   - At the commit used here, fastml's default recipe gives `group_cols` the role "grouping" and `block_col` the role "ordering", so neither reaches the model. Every other non-outcome column is a predictor, so a character `sample_id` would be dummy-encoded; `exclude` removes columns only when fastml splits `data` itself and is ignored with `train_data`. The CRAN release of 0.7.10 still uses `group_cols` and `block_col` as predictors, dummy-encoding them when they are categorical (Section 6).
   - bioLeak's `fit_resample()` drops the outcome, the columns its plan names and, at the commit used here, any `id_cols`. Any other character identifier is one-hot encoded.
   - Passing folds through `resamples` with a feature-only frame, as above, avoids both problems.
2. **Build the rset on the training frame and join by key.**
   - The `rset` carries its own copy of the frame given to `as_rsample()`. fastml fits fold models on that copy and the final model on `train_data` without checking that they agree, so pass the same frame to both.
   - The splitGraph → bioLeak hand-off is keyed on `sample_id`; the bioLeak → fastml hand-off is positional, so `tr` must keep `meta`'s row order. Check `identical()` on the IDs after every join.

3. **Check the number of groups after a composite derivation.**
   - Transitive closure over several relations can merge everything. In our cohort, `mode = "composite"`, `strategy = "strict"` over Subject and Batch produced a single group, because six batches bridge all subjects. splitGraph issued no warning, and `validate_split_spec()` accepted the one-group specification; at the commit used here bioLeak's `as_leaksplits()` refuses it.
   - Use `rule_based`, or keep batch as a blocking annotation that bioLeak tests, unless the estimand is performance on new batches.
   - By contrast, strict composite over Subject and relatedness reproduced the relatedness partition exactly.
4. **Know what the adapter covers.** bioLeak's `as_leaksplits()` maps splitGraph's subject, batch, study and time specifications to the corresponding plans. At the commit used here it also maps the site, region, platform, assay, relatedness, spatial and composite specifications to subject-grouped plans on `group_id`; the CRAN release of 0.3.8 fails for those modes. We call `make_split_plan(group = "group_id")` directly.
5. **Carry the grouping into the permutation null.** A restricted refit null needs the outcome and the plan's group column (`plan@info$group`) in `perm_refit_spec$coldata`, with row names matching the stored refit data. With a feature-only frame and no `coldata`, the null is silently an unrestricted shuffle. With misaligned row names it is also unrestricted, but bioLeak warns "restricted permutations disabled".

4 Simulation study: what does each layer buy?

4.1 Design

Data-generating process. Each replicate simulates 40 families of two siblings, each seen at three visits (240 samples). Each sample is processed in one of six batches. The binary outcome is drawn once per family (prevalence 0.3) and shared by its members, mimicking a heritable trait. Thirty features are generated as the sum of a family effect $N(0, 1)$, a subject effect $N(0, 0.7^2)$, a batch effect $N(0, 0.5^2)$ and residual noise $N(0, 0.5^2)$. Cases carry a shift of +1.2 on three of the 30 features. We then add 24 *re-deposits*: copies of randomly chosen samples with technical noise $N(0, 0.02^2)$. Each copy gets new subject and family identifiers and a batch label redrawn at random from the six batches, while its features keep the original batch effect. The total is 264 samples with 104 subject IDs. The re-deposits are the latent structure that metadata do not reveal. Siblings are declared through a kinship table (coefficient 0.25).

Estimand and references. The target is performance on *new families*. Each replicate also draws an external cohort of 400 new families (two siblings, one visit each; 800 samples) under the same batch effects. We report two references:

- the external AUC of each configuration's final model, fitted on the whole cohort;
- the mean external AUC of its five fold models (obtained with `store_fold_models = TRUE`), each fitted on the analysis rows of one fold: about four-fifths of the frame given to fastml, which for C1, C6 and C7 is the upsampled cohort.

The second matches the training size behind the CV estimate. Honest 5-fold CV is expected to fall slightly below the first, because its models see less data [1]. Bias is the CV estimate minus a reference.

Configurations. All configurations use fastml with logistic regression (glm) and random forest [24] (ranger [25], 500 trees), five folds, no tuning, and upsampling of the minority class. They differ only in where upsampling happens and which folds are used (Table 2). C1–C5 add one layer at a time. C6 and C7 test whether the procedural leak depends on the folds: both upsample the whole cohort before C5's honest grouping. In C6 each copy carries its source's group. In C7 the copies arrive without an identifier, as when rows are duplicated before the metadata are attached. Because their assessment folds also contain upsampled copies, C6 and C7 estimate a slightly different quantity from C5.

All grouped plans are built by `make_split_plan()`, stratified by each group's majority class, and passed to fastml through `as_rsample()`. The C5 feedback is automatic. `audit_leakage(duplicate_scope = "all")` runs on a bioLeak glm fit over the C4 plan. The later member of each flagged pair is then reassigned to the subject and family of the earlier member, and the relatedness groups are re-derived. We ran 30 independent replicates (seeds 1–30).

Table 2. Configurations, and how many dependent units their folds split, averaged over 30 replicates. The last four columns count units split across folds; the denominators are 80 multi-visit subjects, 40 families, 24 re-deposit pairs and, for C6–C7, about 107 upsampled copies (a copy counts as split when it lands in a different fold from its source). *Original rows only: counting each copy as a member of its source, C7 splits on average 21.1 subjects and 11.1 families, and C6 splits none.

CONFIG	PREPROCESSING	FOLDS	SUBJECTS	FAMILIES	RE-DEPOSIT PAIRS	COPIES
C1	Upsample all, then CV	Row-wise	—	—	—	—
C2	fastml, per fold	Row-wise	77.6	40.0	19.1	—
C3	fastml, per fold	Subject groups	0	32.7	18.9	—
C4	fastml, per fold	Relatedness groups	0	0	19.7	—
C5	fastml, per fold	C4 after feedback	0	0	0	—
C6	Upsample all; copies keep group	C5 groups	0*	0*	0	0
C7	Upsample all; copies without ID	C5 groups; copies as singletons	0*	0*	0	86.3

4.2 Results

Figure 3 shows the CV estimates of every replicate against each configuration's external AUC, and Table 3 gives the means, both references and the biases.

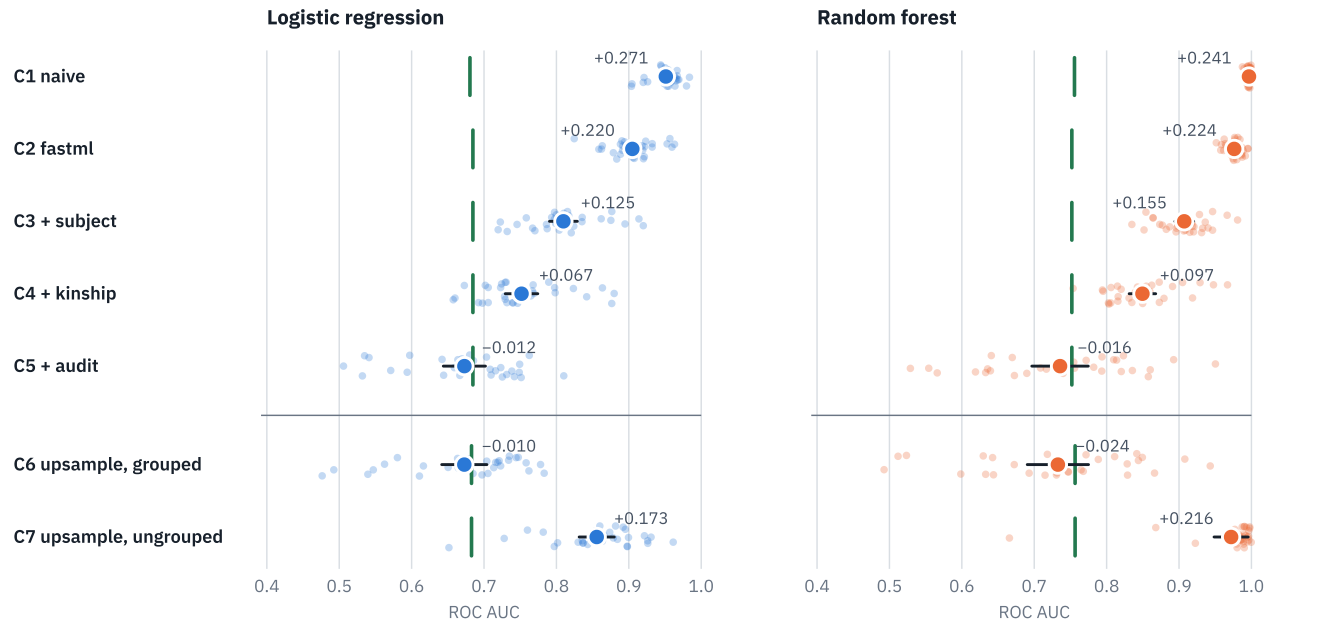

Figure 3. Cross-validated AUC under each configuration: faint dots show the 30 replicates, the large dot the mean, and the whisker the 95% t-interval of the mean. The short green tick marks the configuration's mean external AUC (final model), and the label gives the mean paired bias. C1–C5 move the estimate toward the external value; among C1–C5 only C5's interval covers it for both learners, and C6's does as well. C6 shows that upsampling before honest folds is harmless when copies stay with their source; C7 shows that it is not when they scatter. Hover over a dot for its values.

Table 3. Mean CV AUC, mean external AUC of the final model and of the fold models, bias relative to each reference (mean and 95% t-interval over 30 replicates), and the root-mean-square error of the CV estimate against the final model's external AUC.

CONFIG	MODEL	CV AUC	EXTERNAL, FINAL	EXTERNAL, FOLD MODELS	BIAS VS FINAL	BIAS VS FOLD MODELS	RMSE
C1	Logistic	0.951	0.681	0.674	+0.271 [+0.248, +0.293]	+0.278 [+0.258, +0.297]	0.277
C2	Logistic	0.905	0.685	0.667	+0.220 [+0.194, +0.246]	+0.238 [+0.217, +0.259]	0.231
C3	Logistic	0.810	0.685	0.658	+0.125 [+0.093, +0.157]	+0.151 [+0.127, +0.176]	0.150
C4	Logistic	0.752	0.685	0.655	+0.067 [+0.032, +0.103]	+0.097 [+0.072, +0.123]	0.115
C5	Logistic	0.673	0.685	0.650	−0.012 [−0.048, +0.024]	+0.023 [−0.005, +0.051]	0.095
C6	Logistic	0.673	0.683	0.652	−0.010 [−0.049, +0.029]	+0.021 [−0.009, +0.052]	0.103
C7	Logistic	0.856	0.683	0.668	+0.173 [+0.135, +0.211]	+0.188 [+0.156, +0.220]	0.199
C1	Forest	0.997	0.756	0.754	+0.241 [+0.227, +0.255]	+0.242 [+0.228, +0.256]	0.244
C2	Forest	0.976	0.752	0.750	+0.224 [+0.209, +0.239]	+0.226 [+0.212, +0.240]	0.228
C3	Forest	0.907	0.752	0.746	+0.155 [+0.142, +0.168]	+0.161 [+0.149, +0.174]	0.159
C4	Forest	0.849	0.752	0.744	+0.097 [+0.079, +0.116]	+0.106 [+0.088, +0.123]	0.109
C5	Forest	0.736	0.752	0.739	−0.016 [−0.051, +0.019]	−0.004 [−0.038, +0.030]	0.094
C6	Forest	0.733	0.756	0.739	−0.024 [−0.062, +0.014]	−0.007 [−0.044, +0.030]	0.102
C7	Forest	0.972	0.756	0.749	+0.216 [+0.189, +0.242]	+0.223 [+0.197, +0.249]	0.226

Table 4 gives the paired reduction in the CV estimate at each step, within a replicate. Along C1 → C5 every step reduces the estimate in at least 27 of 30 replicates, with paired Wilcoxon signed-rank $p < 10^{-5}$ (normal approximation with continuity correction; the largest is 9.8×10^{-6}, and the exact test gives $p \le 3.9 \times 10^{-7}$). With 30 replicates these tests mainly confirm that the Monte Carlo error is small; the intervals in Table 4 carry the substance. For random forest the largest step is the last one, C4 → C5 (0.114, against 0.079 for logistic regression; the difference between learners is 0.035, 95% CI 0.007–0.063). That reduction is attributable to 24 re-deposits in a cohort of 264, which were absent from the metadata that splitGraph and fastml receive. Random forests fit their training data closely (forests of fully grown trees interpolate it [26]; the probability forests used here stop splitting nodes of fewer than 10 rows), so a near-copy in the assessment fold falls in the same leaves as its training twin and tends to inherit its prediction.

Table 4. Paired reduction in the CV AUC at each step (earlier minus later configuration) and, for C6 − C5 and C7 − C5, the stated difference (mean, 95% t-interval over 30 replicates, number of replicates in which the value is positive).

STEP	WHAT CHANGES	LOGISTIC REGRESSION	RANDOM FOREST
`C1 → C2`	fastml: fold-local upsampling	0.046 [0.038, 0.055] · 30/30	0.020 [0.016, 0.024] · 30/30
`C2 → C3`	splitGraph: subject grouping	0.095 [0.079, 0.112] · 29/30	0.069 [0.057, 0.081] · 30/30
`C3 → C4`	splitGraph: relatedness closure	0.058 [0.041, 0.075] · 27/30	0.058 [0.046, 0.070] · 29/30
`C4 → C5`	bioLeak: duplicate feedback	0.079 [0.054, 0.104] · 28/30	0.114 [0.084, 0.143] · 29/30
`C6 - C5`	upsample first, copies grouped	0.000 [−0.011, 0.011] · 14/30	−0.003 [−0.014, 0.008] · 13/30
`C7 - C5`	upsample first, copies ungrouped	0.183 [0.149, 0.217] · 29/30	0.236 [0.193, 0.279] · 29/30

Residual bias. The full pipeline's bias relative to the final model, −0.012 [−0.048, 0.024] for logistic regression and −0.016 [−0.051, 0.019] for random forest, is indistinguishable from zero at this precision (the intervals are about ±0.035 wide). Relative to the training-size-matched fold models it is +0.023 [−0.005, 0.051] and −0.004 [−0.038, 0.030]. Only C5 and C6 contain zero on both scales for both learners. Removing the bias did not make single estimates accurate: between C4 and C5 the RMSE fell only from 0.115 to 0.095 (logistic regression) and from 0.109 to 0.094 (random forest), and under C5 the logistic-regression CV estimate showed no correlation with its external AUC across replicates (r = −0.01, 95% CI −0.37 to 0.35).

The audit itself. Across all replicates, the near-duplicate scan flagged 720 pairs, exactly the 720 injected re-deposit pairs. No other pair in any replicate reached the 0.995 threshold. Under the C4 plan about 19.7 of the 24 pairs per replicate crossed folds; after feedback none did. Figure 4 shows the separation for replicate 1. There, re-deposited pairs have cosine similarity above 0.9997, and the most similar genuine pair (two visits of one subject) reaches 0.950. Across replicates, re-deposits never fall below 0.9996 and genuine pairs never exceed 0.965, so the threshold separated the two classes in all 30 replicates.

The price of honesty. The leaky estimates sit near the ceiling (random forest CV SD 0.003–0.034 across replicates under C1–C3). The honest estimate rests on about eight unseen families per fold and 40 in total (random forest CV SD 0.105 under C5). Model selection illustrates the cost:

- Random forest was the externally better model in 27 of 30 replicates (28 for the final models of C1, C6 and C7, which are fitted to data upsampled before the call).
- CV chose random forest in every replicate under C1–C4, so its selection accuracy there equals the share of replicates in which random forest was better: 93% under C1 and 90% under C2–C4.
- Under C5, CV chose random forest in 22 replicates and selected the externally better model in 21, a selection accuracy of 70% (exact 95% CI 51–85%). Against C4's 27, the exact McNemar test gives p = 0.07. The equally honest C6 selected correctly in 24 of 30 replicates (80%).

The leaky designs did not discriminate between the models; they ranked random forest first in every replicate. The CV gap between the learners was 0.045 under C1 (against 0.075 for C1's final models, which are fitted to the upsampled cohort), 0.071 under C2 and 0.098 under C3 and C4 (against 0.067 externally). The leaks therefore distorted the gap in both directions: the ceiling compressed it under C1, while C3 and C4 exaggerated the flexible learner's advantage. Because the leaks favoured random forest, they would be expected to mask a case in which the simpler model is better; we did not simulate such a case.

4.3 Inside one replicate

Replicate 1 illustrates what each package reports. The graph has 377 nodes and 834 edges. Validation passes with 80 `repeated_subject_samples` advisories; splitGraph has no rule for related subjects. Table 5 gives the group counts under each derivation mode.

Table 5. Group counts by derivation mode, replicate 1 (264 samples).

MODE	GROUPS	RECOMMENDED RESAMPLING
subject	104	grouped CV
relatedness	64	grouped CV
batch	6	blocked CV
time	3	ordered split
composite, strict (subject + batch)	1	custom grouped CV
composite, rule-based (subject > batch)	104	grouped CV
composite, strict (subject + relatedness)	64	custom grouped CV

Every derivation, including the strict composites, completed in under 0.01 s. The relatedness `split_spec` carries `batch_group` as a block variable and `order_rank` as its time variable. It passes `validate_split_spec()` with no issues, serialises to 116 kB of JSON and round-trips with an identical grouping. The bioLeak plan built on it has test folds of 64, 63, 48, 37 and 52 samples. The sizes are uneven because stratification balances groups, not samples. `check_split_overlap()` finds no group or family on both sides.

The audit of the C4 design took about 9 s with $B = 100$ refitted, group-restricted permutations. It reports:

- A pooled out-of-fold AUC of 0.814 for the bioLeak glm, against a null mean of 0.452 (SD 0.098). The gap is 0.361 and $p = 1/101 = 0.0099$, the smallest value attainable with $B = 100$ (Figure 5b). An unrestricted shuffle of the same design centres at 0.509 (gap 0.305). The low restricted null is an artefact of permuting whole groups across the observed, label-stratified folds and pooling AUC across them (see below).
- No significant batch–fold association ($\chi^2 = 30.3$ on 20 df, $p = 0.066$, Cramér's $V = 0.17$).
- No proxy feature: the top univariate score, 0.58 for a genuine signal feature, is below the 0.9 flag.
- 24 near-duplicate pairs, 18 of them crossing folds.

The mechanism summary flags `non_random_signal` and `duplicate_overlap`. The first reflects genuine signal, inflated here by the leak; the second identifies the leak. After the feedback step, an audit of the C5 design with the same refitted, group-restricted null ($B = 100$; duplicate scan only) finds no cross-fold near-duplicates. The pooled AUC falls to 0.652, against a null mean of 0.454 (gap 0.198, $p = 0.040$).

The group-restricted null is anti-conservative with fixed folds and pooled AUC. bioLeak's refit null keeps the plan's folds, which were stratified on the true labels. Permuting whole groups across those fixed folds makes each fold's class balance vary much more than in the observed design. Pooled AUC then falls below 0.5

under the null, because each fold model's baseline tracks its training class balance, which is anti-correlated with its test fold's balance [27]. Our own re-implementation (plain glm, same group-level permutation, $B = 100$) shows the size of the effect:

- With the folds held fixed and pooled AUC, the null centres at 0.460 (C4) and 0.437 (C5), reproducing bioLeak's downward shift.
- Re-drawing the stratified grouped folds from the permuted labels for every draw, and scoring the mean of per-fold AUCs, centres it at 0.495 and 0.509. Either change alone removes most of the shift: fixed folds with per-fold AUCs give 0.487 and 0.503, and re-drawn folds with pooled AUC give 0.487 and 0.486.
- Against each null's own observed statistic in the re-implementation, the C4 gap falls from 0.350 to 0.318, with $p = 1/101 = 0.0099$ under both. The C5 gap falls from 0.235 to 0.158, and p rises from 0.0099 to $5/101 = 0.050$.

bioLeak also drops a fold from a null draw when its permuted test labels are all one class; in replicate 1 this happened in 8 of 100 draws for C4 and 34 of 100 for C5. Those folds carry the most extreme class imbalance, so dropping them slightly raises the null mean (refitting plain glm on bioLeak's own permuted labels, from 0.450 to 0.452 for C4 and from 0.443 to 0.453 for C5): it partly offsets the downward shift rather than causing it. Because of the shift, the audit's p-values under this null should be read as too small, most visibly for the weaker C5 signal. With $B = 100$ the audit's own C5 value ($p = 0.040$) differs from the re-drawn value by a single draw.

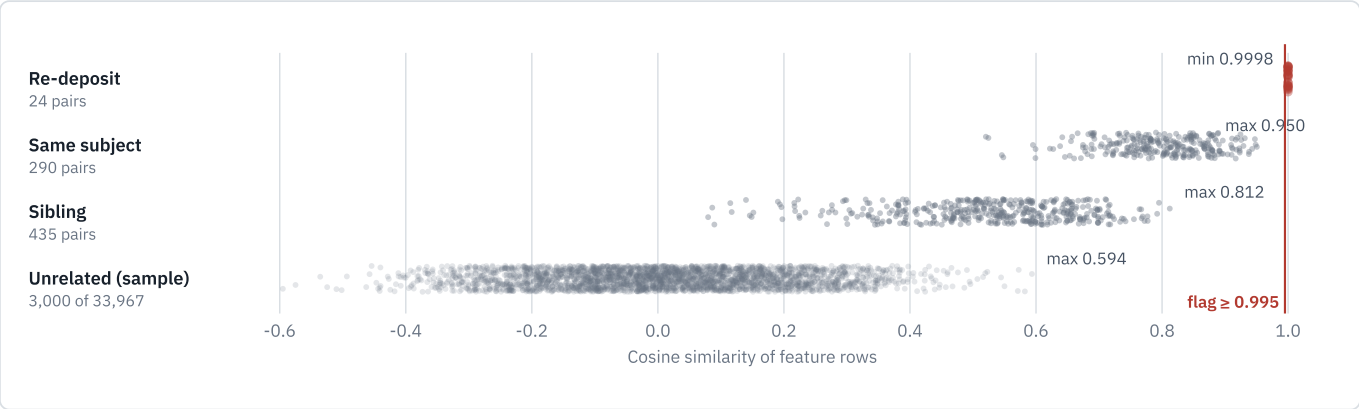

Figure 4. Cosine similarity between feature rows in replicate 1, grouped by true pair relationship; the plot shows every related pair and 3,000 random unrelated pairs. The most similar genuine pair lies 0.045 below the 0.995 threshold (0.030 in the closest of the 30 replicates), and re-deposits lie at most 0.005 above it. Real technical replicates carry more noise than 0.02 SD, so this margin is the first thing to check on real data.

4.4 Procedural leakage depends on the folds

In Table 4 the fastml step (C1 → C2) is the smallest, which could suggest that procedural leakage matters least. C6 and C7 show that this ordering is specific to the design.

- **C6 (copies stay grouped):** when upsampling precedes otherwise honest folds but the copies stay with their source's group, the estimate is unchanged (C6 − C5 ≈ 0).
- **C7 (copies scatter):** when the copies scatter across folds, upsampling alone adds 0.183 (logistic regression) and 0.236 (random forest). That is four to twelve times the 0.046 and 0.020 it adds under row-wise folds, where the dependence already supplies near-copies of each assessment row. Much of that ratio reflects how little headroom the row-wise designs leave (1 − AUC = 0.095 and 0.024 under C2); on the logit scale the mean paired effects differ by a factor of about 1.5 (C7 − C5: 1.13 and 3.38; C1 → C2: 0.73 and 2.22).

The layers' contributions are therefore not additive, most clearly on the AUC scale, and a sequential decomposition such as Table 4 is order-dependent.

The same mechanism turns pure noise into an apparently excellent classifier. We generated 20 datasets of 400 rows, each with 50 N(0, 1) features and an outcome drawn independently with 20% prevalence. Each was fitted with fastml's random forest under its defaults (a 20% holdout, then stratified 10-fold CV on the remaining rows): once after upsampling the full data, and once with `balance_method = "upsample"` (Figure 5a).

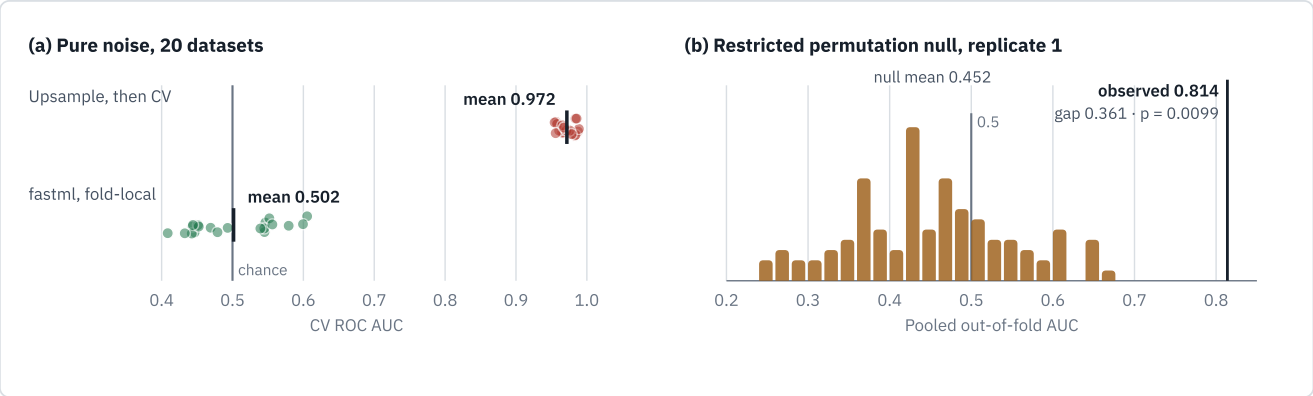

Figure 5. (a) On pure noise, upsampling before CV reports a mean CV AUC of 0.972 (SD 0.011, range 0.954–0.989). The same call with fold-local upsampling reports 0.502 (SD 0.062), which is chance. **(b)** Group-restricted, refitted permutation null of the bioLeak audit in replicate 1 ($B = 100$) against the observed pooled AUC. The null centres below 0.5 because the folds, stratified on the true labels, are held fixed while whole groups are permuted: fold class balance then varies, and pooled AUC is biased downward [27]. Re-drawing the folds for each draw and averaging per-fold AUCs centres the null near 0.5 (Section 4.3).

5 Discussion

Layers and feedback. The simulation is consistent with the thesis of Section 1. In a design constructed to contain all three sources, each step, taken in this order, removed a statistically significant part of the remaining bias; Section 4.4 shows that the size of the procedural step, at least, depends on what precedes it, so the decomposition is order-dependent. Each step was carried out by the package that holds the relevant information: fastml for fold-local preprocessing, splitGraph for the declared subject and kinship structure, and bioLeak for undeclared duplicates. Subject grouping alone could have come from fastml's `grouped_cv` or bioLeak's plans, and a family column would have given the relatedness partition without a graph. Only the last step required information that neither the metadata nor the execution layer contains. Because the re-deposits were absent from the metadata by construction, no metadata-derived grouping and no fold-local procedure could have removed that component.

The feedback path changes the role of an audit. An audit that ends in a report leaves the analyst to judge how much a flag matters. An audit whose findings are written back into the declared structure turns a detected dependence into a severed one. The next estimate then indicates the size of the effect: 0.079 and 0.114 AUC on average over 30 replicates here. In a single dataset, a paired comparison over repeated splits with `delta_lsi()` is the better measure.

Honest does not mean precise. Fixing leakage raised the variance of the estimate. Three practical consequences follow:

- Report intervals computed on grouped folds, for example `cv_ci(method = "nadeau_bengio", n_train =, n_test =)` on repeated grouped CV (without the two sizes it silently returns the uncorrected interval). That correction adjusts for overlapping training sets, not for the number of groups, and is approximate for K-fold designs.
- Prefer repeated grouped CV to a single split; we did not evaluate how much it helps.
- Treat a small difference between models as unresolved rather than as a ranking.

The lower selection accuracy under C5 (70%, against 90–93%; exact McNemar $p = 0.07$ against C4) should not be read as a failure of the pipeline. The leaky designs looked better only because they always chose random forest, which was usually right. Unlike the leaky designs, the honest designs sometimes chose logistic regression (in 8 of 30 replicates each), but it was externally better in only three (C5) and two (C6) replicates, of which CV identified one and two; 30 replicates cannot show how well honest CV discriminates between these learners. Its accuracy of 70% (C5) and 80% (C6) is limited, we expect, by the few independent families behind a single 5-fold split; we did not test that explanation.

A working procedure. For a new biomedical dataset we suggest the following order:

1. Encode every known dependency in splitGraph, including kinship and sites. Validate the graph, and inspect `n_groups` for every mode considered.
2. Choose the grouping that matches the estimand; for new families here, that is relatedness mode. Unless the estimand is performance on new batches, keep batch as a block to be tested rather than a group to be merged.
3. Build the plan once in bioLeak, and use the same `LeakSplits` object for estimation and for the audit.
4. Estimate with fastml on a feature-only frame. Declare all preprocessing as fastml arguments, and supply `train_data` and `test_data` split by group.
5. Audit with `duplicate_scope = "all"`, batch columns supplied through `coldata`, the target scan, and refitted permutations whose `perm_refit_spec$coldata` carries the outcome and the plan's group column. Treat p-values from a group-restricted refit null over fixed, label-stratified folds with pooled AUC as too small (Section 4.3); where the p-value matters, recompute the null outside bioLeak, whose refit null keeps the plan's folds, with folds re-drawn from the permuted labels for each draw or with the mean of per-fold AUCs, as in our re-implementation (Section 4.3; `null_restrat.R`). Write any duplicate or unexpected dependence back into the metadata, and repeat from step 1.
6. Report the final estimate with a group-aware interval, and the audit summary alongside it.

Limitations

- **Simulation, not a benchmark.** This is one data-generating process with known kinship and near-exact duplicates. Real re-deposits carry more technical noise, and kinship is often estimated. Both affect the audit's recall and the relatedness graph. The step sizes also follow from the chosen variance components: the family effect is about half of each feature's variance and the outcome is set per family, so the relatedness step is large by construction. In connectome data, family leakage had little effect where few participants had relatives in the dataset and inflated performance only modestly in a twin subset ($\Delta r \leq 0.04$); in the authors' simulations its effect grew with the share of multi-member families [4]. Every family here has two members.
- **A separated logistic regression.** Unpenalised logistic regression is separated or nearly so in these data: the 264 samples form only 40 families (about 12 of them cases on average), each sharing a family effect on every feature and a single outcome, against 30 features. On the full cohort glm did not converge in 13 of 30 replicates, and in each of those every fitted probability lay within 10^{-6} of 0 or 1 (15% on average in the other 17). On the C5 training folds 115 of 150 fits did not converge. Its AUCs are therefore largely those of non-converged fits, and a penalised model may behave differently.
- **Permutation nulls with fixed folds.** bioLeak's refit null keeps the observed, label-stratified folds and scores pooled AUC. Under group-level permutation this null, scored by pooled AUC, is biased downward (Figure 5b), so the gap is inflated and the p-value too small; re-drawing the folds for each draw or averaging per-fold AUCs removes most of the bias (Section 4.3). The CRAN release of bioLeak 0.3.8 also orients AUC automatically, and there the same null centres at 0.587 (Section 6).

- **Interfaces still being hardened.** Several rough edges are present at the commits used here:
  - fastml's treatment of identifier columns other than `group_cols` and `block_col` as predictors, and its row-wise folds when `group_cols` is combined with the default `resampling_method = "cv"`;
  - the silent fall-back to an unrestricted null, and the fixed-fold refit null;
  - the acceptance of a one-group specification by splitGraph.

  `summarize_leakage_risks()` also marks `repeated_subject_samples` as not severed under relatedness mode, although every relatedness group contains whole subjects. Conversely, it can mark it as severed under a rule-based composite that splits subjects. The composition in Section 3 works around all of these except the fixed-fold refit null, whose p-values it can only qualify (Section 4.3).
- **Scope.** The study covers binary classification with two learners and no tuning. fastml and bioLeak also offer survival outcomes and nested tuning, which we did not evaluate here.

6 Reproducibility

All numbers in this note come from executed code. The software was R 4.5.2 on an Apple M4. The three packages were installed from their repositories at these commits: splitGraph 0.4.0 (c12682e), fastml 0.7.10 (f30f7ad) and bioLeak 0.3.8 (e5e730c2). Other packages were rsample 1.3.1 [28], recipes 1.3.1 [29], themis 1.0.3 [30], parsnip 1.4.1 [31], yardstick 1.3.2 [32], ranger 0.17.0 [25] and igraph 2.2.1 [33].

Development commits and CRAN releases. All three packages are on CRAN under the same version numbers as the commits used here [12,13,15]: splitGraph 0.4.0 (published 17 September 2026), fastml 0.7.10 (28 August 2026) and bioLeak 0.3.8 (21 May 2026). splitGraph's R code at c12682e is identical to its CRAN release. The fastml and bioLeak commits post-date their releases, and in those releases the following behaviours differ from what this note describes:

- **fastml 0.7.10 on CRAN** uses `group_cols` and `block_col` as predictors, dummy-encoding them when they are categorical, and requires them in the data passed to `predict()` (fixed in commit 1f810cb). Every tuned fit fails when the tuning metric is a probability metric (roc_auc, the default, as well as logloss, brier_score and ece), whether tuning uses a grid, Bayesian optimisation or `nested_cv`; racing (`adaptive = TRUE`) fails with every metric. In each case `fastml()` stops with "No models were successfully trained." (fixed in commit c49db44). A user-supplied recipe built on data with integer or character columns fails for every algorithm when fastml splits `data` (fixed in commit b758b70).
- **bioLeak 0.3.8 on CRAN** computes AUC with pROC's automatic direction when the suggested package pROC is installed, as it was here, so AUC never falls below 0.5: a perfectly inverted predictor scores 1 rather than 0. `as_leaksplits()` accepts only subject, batch, study and time specifications. `fit_resample()` has no `id_cols`. The batch-confounding rule is not corrected for multiplicity.

We re-ran the replicate study (`run_all.R`), the pure-noise experiment (`noise.R`) and the Section 4.3 audit (`showcase.R`, `null_compare.R`) against the three CRAN releases, installed from their CRAN tarballs. Every number in Tables 2–4 and Figures 3 and 5a is identical, because the study uses no tuning, no `group_cols` and no bioLeak AUC in its estimates; Figure 4 depends only on the data-generating process, whose similarity values are identical in both runs. Only the audit in Section 4.3 differs. With automatic orientation the restricted null of the C4 design centres at 0.587 (gap 0.227). After feedback, the C5 design gives a null mean of 0.582 and a gap of 0.070 ($p = 0.149$), against 0.454, 0.198 and $p = 0.040$ at the commit used here.

The scripts are:

- `sim_lib.R`: data-generating process, configurations and feedback;
- `run_all.R`: seeds 1–30;
- `aggregate2.R`: Tables 2–4, Figure 3, the paired Wilcoxon tests (normal approximation and exact), the C5 CV–external correlation and the selection statistics;

- `c67_counts.R` : the C6/C7 split counts under Table 2 and the logit-scale effects in Section 4.4;
- `showcase.R` and `null_compare.R` : Section 4.3 and Figure 5b;
- `null_restrat.R` and `fold_drops.R` : the fixed- and re-drawn-fold nulls and the fold-drop counts in Section 4.3;
- `sep_check.R` : the logistic-regression separation diagnostics in the Limitations;
- `simtypes.R` : Figure 4;
- `noise.R` : Figure 5a;
- `fig_splitgraph.R` : Figure 1;
- `part_check.R` : scaling;
- `study_cran/` : `run_all.R` , `showcase.R` , `null_compare.R` and `noise.R` run against the CRAN releases.

One replicate of the seven configurations and the audit takes about 15 s (median 16 s in an independent re-run). The pure-noise experiment takes about 2 minutes. splitGraph derivations are not a bottleneck: on a 5,280-sample cohort (20 stacked replicates), building the graph took about 1 s and the relatedness derivation 0.01 s.

As a cross-version check, `xver_splitgraph.R` re-derives the C3–C5 groupings for all 30 seeds under splitGraph 0.3.0, assembling the relatedness graph with `build_dependency_graph()` because `add_edges()` is new in 0.4.0. C5 is derived there from the injected duplicate pairs, which equal the pairs the audit flagged in every replicate. Every group vector, labels included, is identical to 0.4.0. The folds, and every CV estimate that depends on splitGraph, are therefore unchanged between versions.

References

[1] Bates S, Hastie T, Tibshirani R. Cross-validation: what does it estimate and how well does it do it? *Journal of the American Statistical Association*. 2024;119(546):1434–1445. doi:10.1080/01621459.2023.2197686

[2] Kaufman S, Rosset S, Perlich C, Stitelman O. Leakage in data mining: formulation, detection, and avoidance. *ACM Transactions on Knowledge Discovery from Data*. 2012;6(4):15. doi:10.1145/2382577.2382579

[3] Kapoor S, Narayanan A. Leakage and the reproducibility crisis in machine-learning-based science. *Patterns*. 2023;4(9):100804. doi:10.1016/j.patter.2023.100804

[4] Rosenblatt M, Tejavibulya L, Jiang R, Noble S, Scheinost D. Data leakage inflates prediction performance in connectome-based machine learning models. *Nature Communications*. 2024;15:1829. doi:10.1038/s41467-024-46150-w

[5] Roberts DR, Bahn V, Ciuti S, et al. Cross-validation strategies for data with temporal, spatial, hierarchical, or phylogenetic structure. *Ecography*. 2017;40(8):913–929. doi:10.1111/ecog.02881

[6] Saeb S, Lonini L, Jayaraman A, Mohr DC, Kording KP. The need to approximate the use-case in clinical machine learning. *GigaScience*. 2017;6(5):gix019. doi:10.1093/gigascience/gix019

[7] Wray NR, Yang J, Hayes BJ, Price AL, Goddard ME, Visscher PM. Pitfalls of predicting complex traits from SNPs. *Nature Reviews Genetics*. 2013;14(7):507–515. doi:10.1038/nrg3457

[8] Moscovich A, Rosset S. On the cross-validation bias due to unsupervised preprocessing. *Journal of the Royal Statistical Society: Series B*. 2022;84(4):1474–1502. doi:10.1111/rssb.12537

[9] Santos MS, Soares JP, Abreu PH, Araújo H, Santos J. Cross-validation for imbalanced datasets: avoiding overoptimistic and overfitting approaches. *IEEE Computational Intelligence Magazine*. 2018;13(4):59–76. doi:10.1109/MCI.2018.2866730

[10] Varma S, Simon R. Bias in error estimation when using cross-validation for model selection. *BMC Bioinformatics*. 2006;7:91. doi:10.1186/1471-2105-7-91

[11] Waldron L, Riester M, Ramos M, Parmigiani G, Birrer M. The Doppelgänger effect: hidden duplicates in databases of transcriptome profiles. *Journal of the National Cancer Institute*. 2016;108(11):djw146. doi:10.1093/jnci/djw146

[12] Korkmaz S. splitGraph: dataset dependency graphs for leakage-aware evaluation. R package version 0.4.0. 2026. doi:10.32614/CRAN.package.splitGraph

[13] Korkmaz S, Goksuluk D, Karaismailoglu E. fastml: guarded resampling workflows for leakage-aware machine learning in R. R package version 0.7.10. 2026. doi:10.32614/CRAN.package.fastml

[14] Korkmaz S, Goksuluk D, Karaismailoglu E. fastml: guarded resampling workflows for safer automated machine learning in R. arXiv:2604.05225. 2026. doi:10.48550/arXiv.2604.05225

[15] Korkmaz S. bioLeak: leakage-safe modeling and auditing for genomic and clinical data. R package version 0.3.8. 2026. doi:10.32614/CRAN.package.bioLeak

[16] R Core Team. R: a language and environment for statistical computing. Version 4.5.2. Vienna: R Foundation for Statistical Computing; 2025. https://www.R-project.org/

[17] Pedregosa F, Varoquaux G, Gramfort A, et al. Scikit-learn: machine learning in Python. *Journal of Machine Learning Research*. 2011;12:2825–2830.

[18] Ojala M, Garriga GC. Permutation tests for studying classifier performance. *Journal of Machine Learning Research*. 2010;11:1833–1863.

[19] Phipson B, Smyth GK. Permutation p-values should never be zero: calculating exact p-values when permutations are randomly drawn. *Statistical Applications in Genetics and Molecular Biology*. 2010;9(1):39. doi:10.2202/1544-6115.1585

[20] Lahiri SN. *Resampling Methods for Dependent Data*. New York: Springer; 2003. doi:10.1007/978-1-4757-3803-2

[21] Politis DN, Romano JP. The stationary bootstrap. *Journal of the American Statistical Association*. 1994;89(428):1303–1313. doi:10.1080/01621459.1994.10476870

[22] Nadeau C, Bengio Y. Inference for the generalization error. *Machine Learning*. 2003;52(3):239–281. doi:10.1023/A:1024068626366

[23] Bouckaert RR, Frank E. Evaluating the replicability of significance tests for comparing learning algorithms. In: *Advances in Knowledge Discovery and Data Mining* (PAKDD 2004). Lecture Notes in Computer Science, vol 3056. Springer; 2004:3–12. doi:10.1007/978-3-540-24775-3_3

[24] Breiman L. Random forests. *Machine Learning*. 2001;45(1):5–32. doi:10.1023/A:1010933404324

[25] Wright MN, Ziegler A. ranger: a fast implementation of random forests for high dimensional data in C++ and R. *Journal of Statistical Software*. 2017;77(1):1–17. doi:10.18637/jss.v077.i01

[26] Wyner AJ, Olson M, Bleich J, Mease D. Explaining the success of AdaBoost and random forests as interpolating classifiers. *Journal of Machine Learning Research*. 2017;18(48):1–33.

[27] Parker BJ, Günter S, Bedo J. Stratification bias in low signal microarray studies. *BMC Bioinformatics*. 2007;8:326. doi:10.1186/1471-2105-8-326

[28] Frick H, Chow F, Kuhn M, Mahoney M, Silge J, Wickham H. rsample: general resampling infrastructure. R package version 1.3.1. 2025. doi:10.32614/CRAN.package.rsample

[29] Kuhn M, Wickham H, Hvitfeldt E. recipes: preprocessing and feature engineering steps for modeling. R package version 1.3.1. 2025. doi:10.32614/CRAN.package.recipes

[30] Hvitfeldt E. themis: extra recipes steps for dealing with unbalanced data. R package version 1.0.3. 2025. doi:10.32614/CRAN.package.themis

[31] Kuhn M, Vaughan D. parsnip: a common API to modeling and analysis functions. R package version 1.4.1. 2026. doi:10.32614/CRAN.package.parsnip

[32] Kuhn M, Vaughan D, Hvitfeldt E. yardstick: tidy characterizations of model performance. R package version 1.3.2. 2025. doi:10.32614/CRAN.package.yardstick

[33] Csárdi G, Nepusz T. The igraph software package for complex network research. *InterJournal, Complex Systems*. 2006;1695.