KurveRSC: The Staged Pipeline

Relational Representations, Production-Faithful Selection, and Reproducible Temporal Prediction

Wes Madrigal
Kurve AI

October 27, 2026 | Complete frozen 21-task reproduction

Contents

1 Abstract **2**

2 Scope and relationship to the original report **2**

3 What changed **3**

4 Relational compression and claims about flattening **4**

5 Representation and temporal contract **6**

6 Relational signal construction **6**

7 Bounded graph search with increasing fidelity **8**

8 Matching selection to final fitting **9**

9 Optional signal and readout stages **10**

10 Generalization to new databases **11**

11 Production deployment through Kurve **12**

12 Execution, memory and reproducibility **12**

13 Evaluation and current evidence **13**

14 Measured computational cost **15**

15 Separate new-schema check: Olist seller churn **17**

16 Related work and methodological context **17**

17 Limitations and next work **18**

18 Next direction: a foundation model for graph planning **18**

19 Appendix A: Frozen runtime and reproduction identity **20**

20 Appendix B: Implementation map **20**

21 Appendix C: Evidence **21**

22 References **21**

1 Abstract

KurveRSC learns a relational representation by evaluating GraphReduce programs with a downstream supervised learner. The published reference procedure searched graph configurations, reranked a small finalist set, and fitted the selected representation with CatBoost. The new procedure retains that architecture and extends the object being selected: both the relational program and the temporal fitting/readout procedure must survive comparisons that reproduce deployment.

This report specifies the frozen `staged-complete-reproducible-v1` implementation. Its main changes are bounded historical-target and peer signals, richer numeric and event-state representations, a staged graph search with explicit candidate budgets, production-faithful temporal comparisons, and validation-gated support and readout challengers. A benchmark-independent task context exposes the same complete procedure on a new database. Source identities, keyed prediction artifacts, phase-specific data snapshots, and an original cumulative deadline make the procedure auditable across interruptions and environments.

A fresh, single-freeze reproduction completed all 21 official task evaluations without failures or historical score substitutions. It gives 11 wins and 10 losses against the captured September 28, 2026 TabPFN-Rel client. Replacing the published KurveRSC rows in the captured 15-method RelArena panel yields 1948.6 Elo, compared with 1937.1 for that client. Mean rescaled loss is 0.073390 versus 0.073884. This is a completed author-run comparison; independent maintainer reproduction and public leaderboard acceptance remain separate steps.

The observed gains have material computational costs. The 21 freshly completed tasks take 0.72 to 14.17 hours and peak at 2.88 to 333.51 GiB of task cgroup memory on the measured server. A separate, unregistered seven-table Olist task completes the same staged procedure locally in 46.16 minutes with 0.787512 test ROC-AUC. The report distinguishes implemented behavior, measured evidence, and remaining limitations; it does not attribute the aggregate gains to any one feature without a controlled ablation.

The result also bears on a broader architectural question. Strong performance from task-selected relational compression followed by CatBoost challenges the claim that a flat learner interface is inherently uncompetitive. It does not show that arbitrary aggregation is lossless, or replace a matched comparison with KumoRFM-2. We examine the actual published claims, relevant prior work and the distinction between representation limits and selection regret below.

Production deployment is available through the Kurve product, which provides warehouse integration, operational compute graphs, model workflows and customer-controlled deployment for sovereign AI. Kurve integrates with existing ML and MLOps stacks 34, 35. The experiments in this report use the frozen DuckDB-based open-source reference implementation; product deployment capabilities are distinct from the benchmark configuration.

2 Scope and relationship to the original report

This is a companion to the original technical report. That document remains intact. It explains GraphReduce's directional reductions, the original representation-search formulation, and earlier benchmark captures. Some of its method sections were updated during development and describe a 12-cutoff pooled profile, while its main published results and the README's September 2 reference describe the earlier full-data, latest-cutoff procedure. Those profiles must not be conflated.

Here, **published reference procedure** means the earlier full-data graph search, three-finalist/three-fold rerank, and one initial production cutoff documented with the published results. **Frozen staged procedure** means the complete recipe identified above, including every later selection and readout stage. **Published scores** in the comparison tables mean the KurveRSC rows in the captured RelArena panel. That later panel has different values from the earliest table retained in the original report. The exact comparator is `tabpfn-rel-client-2026-09-28`, rather than an unspecified API or local version.

The final official leaderboard was generated at 13:34 UTC on October 27, 2026, after every task completed. Final resource receipts were generated immediately afterward. The comparison panel was captured on October 25; no score in the new KurveRSC column is filled from an earlier run. Earlier October 26 evidence is retained separately and is not overwritten by this revision.

2.1 Three interfaces with different contracts

Interface	Contract
`kurversc.fit()` and `predict()`	Native graph search/fitting and frozen-plan prediction. Caller controls their settings.
`kurversc.run_staged_task()`	Complete fixed staged procedure: graph search, finalist checks, temporal policy, signal/readout challengers, refitting, keyed prediction.
RelArena `KurveRSC20261022System.run`	Submitted thin adapter to the packaged staged entry point; the harness retains responsibility for held-out scoring.

Installing a package with the old version number and calling bare `fit()` does not identify or reproduce the reported complete system. The candidate wheels still carry development version labels `kurversc==0.1.1` and `graphreduce==1.10.17`; their exact bytes and the runtime lock identify this implementation. Appendix A records the hashes. The review branches prepare release versions 0.2.0 and 1.10.18 with the same Python payload before version stamping. Exact Git commits and a workspace lock identify the review installation; neither version is claimed published to PyPI by this report.

3 What changed

The essential difference is that representation selection, temporal support, and final inference now form one explicit, validated procedure. The following table compares the published reference procedure with the frozen staged procedure. Some capabilities appeared in intermediate versions; this table describes the net change rather than assigning each change to a particular release number.

Component	Published reference procedure	Frozen staged procedure
Graph evidence	Complete latest-cutoff graph search	Bounded 10K/50K screening and confirmation, historical promotion, then full-source temporal reranking
Search admission	Small finalist set after graph scoring	Structural coverage and confidence-frontier retention, with explicit per-stage ceilings
Finalists	Three graphs, three sequential cutoff folds	Up to 12 reranked graphs; an additional bounded MAE comparison can test one smaller surviving finalist
Topology	Path-specific acyclic expansion	Bounded repeated-table paths, represented by distinct aliases in an acyclic execution graph
Observed outcomes	Target and entity keys excluded from learner payload	Keys remain structural; observed labels can return through an availability-aware historical child relation
Peer signal	Ordinary child aggregates	Separate time-filtered peer-label moments and recency-weighted summaries, excluding the query entity's own outcomes
Numeric distributions	Primarily scalar reductions	Bounded empirical-CDF sketches inside `base`; frozen recipe enables them
Event state	Aggregate counts and window summaries	Searchable ordered lags, latest values, and trailing event-count windows via `state`
Source columns	Bounded source-column funnel	Explicit intelligent selection, detailed feature audit, and bounded categorical tie refinement after promotion
Temporal fitting	One initial production cutoff	Preserve that incumbent; test pooled/support alternatives separately using the production lifecycle
Readout	Direct CatBoost output	Bounded text, exposure, domain, zero-aware, recurrence, semantic, count and complementary-view challengers, subject to applicability and admission
Temporal validation	Graph-fold score and stability	Recent-fold safety plus older-fold stability; paired entity-aware evidence for later procedure changes

Component	Published reference procedure	Frozen staged procedure
New databases	Library/benchmark-specific assembly	Explicit task context with source keys, clocks, horizons, observed labels and target-free requests
Reproduction	Frozen graph and fitted learner	Checksummed complete procedure, phase snapshots, model/member identity, keyed outputs and resumable stages
Execution cost	Smaller graph-only workflow	More post-search comparisons; disk-backed historical capture and reduced copies control avoidable cost

The downstream estimator remains a conventional CPU learner. The reported complete recipe uses CatBoost, with bounded linear or other CatBoost readouts where declared by a challenger. It does not invoke TabPFN or TabFM to produce these benchmark predictions. Optional foundation-model backends in the native library are a separate capability.

3.1 Division of responsibility

GraphReduce implements relational execution: table/edge operations, time-aware filtering, annotations, type-aware reductions, feature families, and replayable plans. KurveRSC supplies the prediction task, observed-label relation, peer provider, candidate search, learner fitting, temporal policy and challenger selection. The source adapter declares availability and exports the correct phase of the database. The benchmark harness owns held-out answers and scoring.

This division keeps reusable representation primitives in GraphReduce while keeping supervised selection and task-label semantics in KurveRSC. A peer-label provider is different from GraphReduce's `context` family: the latter computes within-source peer comparisons on ordinary rows, whereas the former uses mature historical outcomes from other entities.

4 Relational compression and claims about flattening

4.1 What the published criticism says

The criticism deserves a precise response. Fey et al.'s ICML 2024 position paper argues that manual relational feature engineering is laborious and can select suboptimal features; it proposes end-to-end learning on temporal heterogeneous graphs 10. KumoRFM-2's introduction specifically criticizes "generic, task-agnostic feature aggregation schemes" that "fail to exploit task-specific relational structure" 15, Section 1. Kumo's explanatory documentation uses the heading "Why flat tables fail" and identifies multi-hop relationships, temporal sequences, topology and combinatorial interactions as information that ordinary flattening can discard 29.

Those are substantial objections to a fixed or manually restricted representation. They are not a theorem that every representation materialized as an entity-by-feature matrix must be predictively inferior. In particular, the KumoRFM-2 passage distinguishes the expense of feature engineering from the limitations of task-agnostic aggregation; it does not establish a universal noncompetitiveness result for task-adaptive relational programs.

4.2 A tabular interface can contain a relational representation

KurveRSC operates on tables, paths, timestamps, historical outcomes and peer memberships before producing the learner's matrix. The task's validation objective selects which relational program survives. Ordered state, conditional histories, CDF sketches, repeated-table paths and peer outcomes provide concrete examples of relational information that can reach that matrix. The graph operators need not be differentiable for their selection to depend on the task.

The substantive distinction is therefore **how relational information is compressed and adapted to the task**, rather than simply whether the final learner consumes rows and columns. A fixed mean/count table and a selected relational program can have the same output shape while retaining very different predictive information. Task-level program selection is still different from learning the aggregation weights end to end; our results do not make those mechanisms equivalent.

Compression also need not preserve every source detail to preserve a prediction. If a representation $\phi_q(\mathcal{D}, e, t)$ satisfies

$$P(Y \mid \mathcal{D}, e, t) = P(Y \mid \phi_q(\mathcal{D}, e, t)),$$

then it retains the task's conditional outcome distribution despite discarding other database information. This is a sufficiency criterion, not a claim that the current finite feature families satisfy it. Work on invariant set functions shows why aggregation must be judged by its transformations and assumptions; work on GNN expressivity likewise shows that learned neighborhood aggregation has representational limits 20, 21. Neither result proves universality for GraphReduce's bounded scalar features or an equivalence to a relational GNN.

4.3 What the current results challenge

The fresh 21-task scorecard places this compression-plus-learner procedure first by point Elo on the captured 15-method panel, which includes relational GNN and foundation-model systems. That is empirical counterevidence to a blanket claim that relational compression followed by a conventional tabular learner cannot be competitive. The particular panel and uncertainty matter: the lead is 11.5 Elo and the bootstrap intervals overlap. The completed author-run reproduction is not yet a confirmed public leaderboard submission or evidence of superiority on every task.

The conclusion also has independent context. RDBLearn and TabPFN-Rel materialize relational features for a tabular foundation learner 17, 16. The RelArena report explicitly presents this approach as evidence that such representations remain competitive with specialized relational architectures 16. KurveRSC adds evidence for task-selected relational programs with a conventional CPU learner, rather than establishing the general idea of automated relational feature construction for the first time.

Our captured panel does not contain a matched KumoRFM-2 result. Consequently, these experiments do not establish a direct win over KumoRFM-2. A comparison would need matched database availability, task coverage, splits and scoring, with external pretraining, training time and inference cost disclosed. RelArena also separates reproduced leaderboard rows from paper-reference results with different provenance 4. Nor do our hours-long searches refute Kumo's claims about the convenience or latency of in-context prediction, or establish billion-row scalability. The defensible conclusion concerns the competitiveness of the architecture on this measured task panel.

4.4 Selection fidelity is a separate source of headroom

The unresolved search problem is meaningful precisely because it is distinct from a representational ceiling. Let $J_{\text{full}}(q; p)$ be a common full-fidelity temporal-validation score for graph/plan q, holding fitting procedure p fixed. Use AUC or negative MAE so larger is better. For a declared candidate set $\mathcal{Q}$ and selected $\hat{q}$, define

$$q^* = \arg\max_{q \in \mathcal{Q}} J_{\text{full}}(q; p), \qquad \text{Regret}_{\text{selection}} = J_{\text{full}}(q^*; p) - J_{\text{full}}(\hat{q}; p).$$

A positive value means that the existing candidate space already contains a better full-fidelity validation configuration. It would motivate a selector improvement without requiring another feature family. A second failure mode occurs within a graph shape: the plan discovered from a capped cohort can omit rare categories or interactions that a richer, still time-eligible discovery sample would expose. A graph surviving promotion therefore does not prove its frozen plan captured every useful source pattern.

These are testable opportunities, not measured guarantees of extra Elo. The current report has not established an exhaustive oracle for every task, and a validation winner need not improve future test performance. The next controlled audit should separately measure candidate retention and plan-discovery loss: compare the same programs and fitting policies at 10K, 50K and full fidelity; measure winner retention and selection regret; and then compare capped versus full-source plan discovery for the same retained shapes. These comparisons must use training/validation data, preserve temporal boundaries, and leave the final test for locked evaluation. The objective is better allocation of evidence to existing relational hypotheses under the runtime budget, rather than an unbounded expansion of the graph lattice.

5 Representation and temporal contract

Let a task predict outcome $y_{e,t}$ for entity e at time t, with label horizon h. Let τ be the information boundary of the current validation or forecast phase. A graph program q produces

$$x_q(e, t; \tau) = \text{GraphReduce}(q, \mathcal{D}_{\text{visible}}(t, \tau), \mathcal{H}_{\text{observed}}(t, \tau)).$$

The program records path instances, depth, families, annotations, payload budgets, target-history participation, categorical decisions and temporal semantics. A second policy p specifies eligible label cohorts, estimator mode, readout, blend and member identities. The fitted predictor is $f_{q,p}$. KurveRSC selects these in bounded stages. It does not exhaustively optimize every pair in the full Cartesian product of graph and readout choices.

5.1 Source visibility and label maturity

Event time and observation time are separate declarations. With ordinary strict event semantics, a source row contributes only before its query cutoff; an explicit inclusive declaration admits rows known at the cutoff. An optional `available_at` clock additionally constrains arrival. Both the row's query boundary and the phase's frozen availability boundary apply. Timestamp precision is retained; subday events do not require rounding a query down to a day.

For labels, the default portable contract sets availability to $t + h$. A caller may supply a later or row-specific observation clock. The complete portable entry point rejects labels declared observable before their horizon finishes. An initial-fit label must already be mature before the first observed-refit cohort; refit labels must be mature by the corresponding forecast origin.

Historical-target joins use the label's observation clock when supplied. Such explicit known-by-time clocks are inclusive; the legacy bare `Labels` contract without an observation clock uses a strict historical timestamp. Peer-history contributions are stricter: availability must be strictly before the query cutoff. These distinctions are part of the input contract, not choices selected on test performance. The active peer provider bounds its pool before applying each query's maturity filter, as detailed below.

Every retrospective origin receives its own eligible source snapshot. Replaying an old query against an uncensored future database would leave automatic category selection, text bases and peer construction exposed to future data, even if a final aggregate applied a timestamp predicate. Phase freezing covers those preparation steps as well as learner rows.

5.2 Held-out boundaries and routing

The complete portable context receives observed training and validation labels, and forecast requests containing entity/time keys without a target column. It does not load a benchmark task registry to recover missing information. Inner selection and outer forecast contexts are distinct; a forecast's later timestamp does not authorize source arrivals beyond the caller's frozen phase.

Predictions carry entity/time keys through each stage and are joined to the original request order. An array of the right length is insufficient evidence of correct alignment. Structural keys route joins, peer exclusion, folds and output alignment; they are excluded from learner payloads.

Row timestamps cannot reconstruct overwritten database attributes. The caller must supply historical versions or snapshots for mutable fields. Declaring a current mutable table `timeless=True` does not establish historical correctness. Likewise, excluding held-out task labels does not establish that an external pretrained encoder has never encountered related public source text.

6 Relational signal construction

6.1 Historical targets and peer outcomes

KurveRSC can attach the full unsampled observed training-label pool as the reserved `__kurversc_target_history__` relation, with entity keys, an observation timestamp and a numeric target value. The history pool is independent of the 10K/50K search cohorts. During inner validation it contains training outcomes; later prediction may additionally use observed validation outcomes subject to their availability. Ordinary GraphReduce reductions then

expose prior counts, means, recency and other permitted history summaries for an entity. A candidate can omit this relation entirely.

Peer context captures a different question: how have other entities sharing a declared relational neighborhood behaved? Memberships are derived from the schema. The active provider first bounds the observed history pool with a target-blind hash of entity/time identities. Each query then filters that pool for maturity, uses latest eligible outcomes, and excludes its own entity's contribution. The frozen recipe requests both ordinary and recency-weighted summaries, with at most 50,000 historical rows. Ordinary limits are eight paths, 128 neighbors and 10,000-row query batches; large histories use four paths and 32 neighbors.

Sampling before query-local maturity can reduce usable historical support: rows unavailable to an older query can occupy slots in the bounded pool. A separate maturity-first sampler exists in `peer_sampling.py`, but it is not wired into the active frozen graph provider. This report does not attribute that experimental sampling correction to the reported default. The optional latest-completed peer view is a separate later challenger with its own contract.

For example, the signal can distinguish a user's own past activity from the past outcomes of users connected to the same objects. This is supervised, time-aware relational signal, so its maturity and leave-own-entity-out rules are essential. It is not equivalent to sending an entity ID to CatBoost.

6.2 Feature families and bounded numeric distributions

GraphReduce family	Information retained
`base`	Counts and type-aware reductions; recency, relationship diversity/repetition, bounded predicate history, exposure-normalized activity and numeric distribution sketches
`temporal`	Numeric magnitude, variation and change inside historical time windows
`state`	Latest values, ordered lags and summaries of the last N events, suitable for irregular event cadence
`sequence`	Activity rates, recent shares, concentration, bursts and active spans
`conditional`	Counts, shares, presence and changes under categorical/annotated conditions
`episode`	Event opportunity through row counts and distinct-event counts, including windows
`context`	Peer-group size and numeric deviations on ordinary child rows before reduction
`semantic`	Caller-declared expressions and predicates; available in GraphReduce, outside automatic generic family discovery

Automatic annotation may expose bounded categorical predicates and gated numeric values. Root/parent columns also remain eligible inputs, with applicable annotations such as parent age. This should not be confused with fitting an AutoGluon model on every parent field; that is not part of this recipe.

CDF means cumulative distribution function. For selected numeric column v, GraphReduce fits up to seven quantile cutpoints on an eligible training sample of at most 20,000 rows. At entity grain it emits the finite-value coverage and bounded empirical fractions

$$\widehat{F}_{e,t}(c_j) = \frac{\sum_{r \in R(e,t)} \mathbf{1}[v_r \text{ finite}]\mathbf{1}[v_r \le c_j]}{\sum_{r \in R(e,t)} \mathbf{1}[v_r \text{ finite}]}.$$

Empty support remains missing. Constant, binary and entirely unobserved columns do not create these sketches. Cutpoints are frozen as SQL literals in the plan; validation and forecast values do not refit them. This preserves distinctions such as a broad distribution versus a concentrated distribution with the same mean. It remains a bounded sketch, not an arbitrary empirical distribution or learned row-level message-passing representation.

CDF features belong to `base`, not a new family. The current complete recipe sets `base_cdf=True`. Native configuration and matched diagnostic interfaces can compare CDF on/off, but this frozen all-21 recipe does **not** double every graph into an on/off pair. That is an important difference between available configuration knobs and the actual reported defaults.

6.3 Bounded repeated-table paths

A relational walk may revisit a base table while the executable compute graph remains acyclic. KurveRSC creates separate table instances for separate paths. An occurrence limit of one reproduces acyclic path discovery; a limit of two admits bounded self/peer paths, for example entity to association to another entity of the same type. The staged search includes these topology alternatives as bounded challengers. Depth remains at most four. It does not traverse unbounded cycles, nor run a Graph Neural Network over every database row.

6.4 Payload budgets and categorical refinement

The reference source-column budget is four columns per family/node, with up to 32 derived features per selected source column and one continuation function per propagated derived column. Structural join keys, dates and context keys remain outside the payload-column budget. A deterministic metadata ranking uses coverage, entropy, spread, cardinality and identifier redundancy; the feature audit records both utility rank and actual selection order. Disabling `intelligent_column_selection` preserves configured/source order for ablation.

Categorical ties can change which frequent values reach a bounded annotation. After stronger graph shapes survive promotion, a small refinement stage tests seeded deterministic category tie decisions for up to two parent shapes. Seeds 0 through 7 define the available tie variants; a total stage budget of eight limits evaluations. These are categorical tie seeds, distinct from CatBoost seeds 0, 1 and 2 used in procedure validation.

The staged recipe uses modern high-cardinality handling (`skip`, with text categorical fallback disabled), alongside a published-compatible anchor in the search machinery. It does not globally force every graph to reproduce GraphReduce 1.10.14's top-category behavior. Compatibility semantics, annotation operations and ordering are recorded in the chosen configuration and plan. Graph-level `auto_text_features` remains disabled; the bounded text readouts described below are tested after graph promotion.

7 Bounded graph search with increasing fidelity

The system must retain useful higher-order candidates long enough to evaluate them fairly, while avoiding a full depth/family/history/topology Cartesian search. The frozen `staged-retention` profile does this with candidate ceilings, structural coverage, and promotion of uncertainty-competitive configurations.

7.1 Fidelity changes more than the number of learner rows

The low-cost adapter constructs deterministic, cutoff-balanced task cohorts and their connected source rows. Cheap child sources can also be capped. A label cohort increase therefore changes the eligible connected subgraph as well as the estimator evidence. This is why a 10K screen is an approximation to a full-source evaluation, not an equivalent fit with fewer trees.

Plan discovery and promoted evaluation use separate sources. Promotion replays the candidate's frozen operations/schema instead of silently discovering a different candidate as history expands. Full reranking uses uncapped eligible source tables for its selected cutoffs. Source completeness at this stage does not remove the separate 100,000-row budget per CatBoost fit.

The initial search uses the smallest available window with adequate evidence: normally at least three and at most 12 training cutoffs, seeking at least 256 labels and, for binary tasks, at least 24 examples in each class. Short histories use the available support. This window is selected from training evidence and does not change the independently configured initial production-frame count.

7.2 Reference stage ceilings

Stage	Candidate ceiling	Evidence or purpose
Screening	32	10,000-row screening fidelity; balanced structural coverage
Individual challengers	24	Bounded additions such as history, state, context, topology and deeper paths
Combined challengers	12	Bounded combinations supported by earlier evidence

Stage	Candidate ceiling	Evidence or purpose
Confirmation	40	50,000-row fidelity; retain score and confidence-frontier contenders
Historical fidelity promotion	8	Stronger temporal support with the candidate's plan frozen
Categorical refinement	8	Seeded tie variants after promotion, rather than multiplying the initial lattice
Full temporal rerank	12	Full-source evaluation over up to three cutoff folds

These are upper bounds, not promised counts of unique graph configurations. Candidates repeat across fidelities; inapplicable shapes are pruned; cache hits do not require another fit. The sum of these stage ceilings is 136 evaluations before accounting for repeats, folds, estimator members and later procedure audits. It must not be described as 136 unique graphs or 136 total learner fits.

Depths 1 through 4 are available subject to family/topology and capability limits. Expensive family combinations retain depth restrictions. Annotation on/off, historical targets, bounded repeated-table paths and compact versus richer family combinations receive coverage without exhaustive pairing of every axis. Static feature preflight rejects configurations estimated to exceed the 10,000-feature search ceiling. Estimated width is a planning constraint, not a rigorous bound on SQL working memory.

7.3 Full-fidelity comparison and complexity

The latest fold defines the recent leader and an uncertainty-aware safe set. The selector rejects a candidate when recent evidence clears both a practical loss threshold and its uncertainty comparison. Within the surviving set, a stability-adjusted walk-forward objective breaks ties. For fold scores s_k, the objective rewards average quality and penalizes fold-score uncertainty; the reference stability multiplier is 0.25.

With the complexity selector enabled, a graph can replace that winner only as a practical uncertainty-equivalent alternative with at least a twofold feature reduction. The complexity substitution loss is capped at 0.01 AUC or 1% relative MAE, using paired fold differences where available. This avoids letting a large uncertainty estimate justify an arbitrarily large accuracy loss for simplicity. The frozen registered metadata records `rerank_practical_recent_ties=False`; the complexity cap is not a blanket 1% bound on every stability-based choice. Nested-depth safety is also disabled in the complete graph command.

An additional MAE-only production-finalist check can compare the selected graph with at most one already-surviving graph having at most half as many features. It replays the complete initial fit plus observed-validation refit over two or three available origins and three seeds. Replacement requires unanimous seed selection and per-fold practical safety. Target-blind capacity preflights retain the incumbent when a complete comparison cannot fit the declared budget; partial evidence cannot promote a challenger.

8 Matching selection to final fitting

8.1 Temporal support is selected after the graph

The graph-stage incumbent keeps `full_training_frames=1` and `temporal_estimator_mode="frame_ensemble"`. Historical search confidence does not automatically add old equally weighted models to final inference.

After graph selection, the basic temporal menu is:

Policy	Fitting procedure
`incumbent`	One initial frame ensemble, followed by eligible observed-validation member fits
`pooled12`	One estimator over up to 12 selected historical cohorts
`floor1000`	Pooled support expanded to meet a minimum of 1,000 eligible labels where available
`ordered_same_support`	Regression-only ordered-median readout on matched support
`ordered12`	Regression-only ordered-median readout on broader historical support

The declared temporal blend grid adds 0.125 and 0.5 challenger weights using already-fitted paired predictions. A pooled fit divides a global 100,000-row budget across its cohorts. A frame ensemble instead gives each member its own bounded training sample, so adding members can increase total fit cost and training coverage. These are different procedures and must be compared as such.

The base CatBoost policy remains 300 iterations, depth 6, learning rate 0.05 and L2 regularization 10. Search uses held-out evaluation/early stopping where declared. Final fitting uses the selected procedure without consulting forecast targets. Some bounded specialist/teacher heads have their own fixed budgets; it would be inaccurate to describe the complete system as one 300-tree model.

8.2 Retrospective origins, seeds and paired confidence

Later procedure comparisons use up to three chronological origins and seeds 0, 1 and 2. Available origins must provide complete matched predictions for a candidate to be admitted. Validation recreates the same initial support, label-maturity rules, observed-validation refit, sampling, feature schema and member aggregation that inference will use.

Entity-clustered paired uncertainty accounts for repeated entities. The temporal gain calculation gives half the weight to the latest origin and shares the other half among older origins. Shared entity bootstrap multipliers preserve dependence across time. A positive recent deployment-seed gain, seed agreement, confidence and older-origin safety govern admission; unsupported comparisons retain the incumbent. Later readout gates additionally require the complete available origin/three-seed grid and, where specified, positive gains in every case. Multiplicity allowances are fixed by stage, not chosen after observing official scores.

These safeguards reduce selection risk. They do not establish a no-regression guarantee under temporal shift, nor correct for every research hypothesis explored before freezing the pipeline. Three retrospective seeds are evidence for selection; the deployed procedure uses seed 0 rather than a test-selected seed or an automatic three-seed prediction ensemble.

8.3 What is ensembled at prediction time?

Graph finalists compete to select **one graph program**; their predictions are not automatically averaged. Inside that graph, the admitted fitting policy may use multiple estimators. A frame-ensemble policy averages its initial-frame and eligible observed-validation member predictions. A pooled policy fits one estimator to its pooled sample. An admitted blend combines the declared challenger with the incumbent using the locked weight.

Thus, “three training frames” is neither “three graphs” nor necessarily exactly three models at prediction: observed-validation member fits can add members. The saved fit audit records actual membership. It is the authoritative answer for a particular task.

9 Optional signal and readout stages

Optional means **conditionally applicable and validation-admitted**. In the complete staged recipe these stages are checked automatically; a user need not manually select a known winner. They remain separate from the bare native `fit()` defaults. They also remain separate from the graph-family lattice, preventing each new downstream hypothesis from multiplying all graph candidates.

Stage	Candidate signal or procedure	Principal safeguard
Parent text	Train-only bounded TF-IDF with regularized linear readouts and 25%/50% blends	Eligible root text, matched temporal evidence
Exposure quality	Schema-derived timestamped exposure/event/object paths, ordinary binary responses and other-entity priors	Cutoff checks, own-entity exclusion, matched support and regularization variants
Extra support	Three/five initial-frame ensembles with the same observed-validation procedure	Exact support availability and full procedure comparison
Target domain	Nonnegative projection or integer-count projection where mature labels support it	Training-only domain detection and validation admission

Stage	Candidate signal or procedure	Principal safeguard
Zero-aware regression	Separate zero/positive behavior with a distribution-aware readout	Actual sampled-cohort support; unavailable optional head retains comparator
Object history	Phase-frozen object breadth and history	Source metadata, mature labels, bounded complete audit
Recurrence	Activity timelines, dormant-return patterns and additional historical support	Same forecasting/refit lifecycle
Joint semantic recurrence	Frozen parent-text embeddings with cross-fitted supervised propensity and recurrence information	Train-only projection, out-of-fold supervision, independent replay
Conditional count	Count-specific downstream challenger after an admitted integer projection	Complete grid, fixed weights and native replay
Production sequence	Sequence algebra and precise subday sequence heads	Qualified binary native lifecycle; preserve already-admitted procedures
Complementary views	Bounded peer history, object co-occurrence coordinates and ordinary-response annotations in separate heads	Fixed three-view mean, locked blend grid, native parity and capacity preflight

The first signal menu has eight default recipes: two text blends, four exposure variants and two frame-support variants. Experimental partial-exposure blends available to standalone audits are not silently part of that frozen eight-arm menu. The recovered-readout stage has nine named candidates grouped by target domain, zero awareness, objects, recurrence and semantic recurrence. Subsequent count, sequence and complementary-view stages have narrower applicability.

Capacity is explicit. Recovered optional readouts use a 500-million-feature-cell limit, with semantic limits of 500,000 rows and 200,000 documents. Sequence and complementary views use 125 million cells and 250,000 rows; complementary views add at most 112 columns. The MAE finalist check has its own one-billion-cell optional-fit and three-billion-cell audit-matrix limits. These are not caps on all graph materialization or total task RSS.

Later stages preserve already-admitted procedures when their validation/inference adapter cannot reproduce that procedure exactly. For example, the current sequence stage qualifies an incumbent frame-ensemble lifecycle, while the complementary-view stage qualifies a direct pooled lifecycle. Explicitly inapplicable or over-capacity candidates yield an exact incumbent fallback. Unexpected numerical, provenance or control-replay failures remain errors; they are not silently reported as a successful weaker model.

9.1 External text representation disclosure

The semantic challenger pins `sentence-transformers/all-MiniLM-L6-v2` to the following revision:

`1110a243fdf4706b3f48f1d95db1a4f5529b4d41`

It uses frozen 384-dimensional CPU embeddings and a training-only basis of at most 32 dimensions. Encoder weights are not fine-tuned on task labels. Its public training mixture includes StackExchange data; this is an external pretrained representation with possible source-domain overlap, not a database-trained encoder with a guaranteed chronological pretraining boundary. Pinned model card.

10 Generalization to new databases

The staged interface takes schema and task declarations, rather than a table of benchmark-specific winning graphs. `TaskContext.from_tables()` accepts source frames, primary/foreign keys, event and arrival clocks, observed labels, a prediction horizon, inner/outer boundaries and target-free requests. String and composite keys are canonicalized consistently for routing. The benchmark compatibility constructor, `from_protocol_splits()`, preserves the caller's already-censored tables, source order and temporal metadata so the benchmark handoff does not silently change the representation.

```
context = kurversc.TaskContext.from_tables(
    tables=source_tables,
```

```
    task=kurversc.PredictionTask(
        "accounts", "account_id", "prediction_time", "churn", "30d",
        kind="classification",
        label_available_at="outcome_observed_at",
    ),
    training=observed_training,
    validation=observed_validation,
    requests=forecast_requests,
    inner_cutoff=selection_boundary,
    outer_cutoff=forecast_boundary,
    dataset_name="new-database",
)
bundle = context.export("artifacts/churn-input")
result = kurversc.run_staged_task(
    bundle, output="artifacts/churn-run", task_hours=23.75,
    minimum_available_memory_gib=8,
)
predictions = result.predictions
```

Here `source_tables` is a mapping of names to `SourceTable` objects with explicit key and time declarations. The exported bundle is checksummed and does not contain forecast answers. The packaged wheel includes the transitive stage implementations, so portable inference does not depend on a research checkout or a RelArena registry. The frozen complete recipe supports binary classification and numeric regression at deployment seed 0; this is narrower than every mode exposed by the native library.

Generality is architectural and conditional: discovery follows schema metadata, and inapplicable optional heads fall back. It does not establish universal performance or complete support for every source dialect, key domain, delayed arrival pattern or task type. Some optional timeline/view providers have additional eligibility limits. The portable context still handles the core task, with reasons recorded for optional exclusions.

11 Production deployment through Kurve

The Kurve product provides the production layer around relational signal compression. It connects existing warehouses, discovers schema relationships, manages compute graphs, generates point-in-time training and production tables, and supports model training and prediction 34, 35.

Kurve supports Snowflake, Databricks and Amazon Redshift, with deployment through a Snowflake Native App or within customer-controlled cloud and on-premises infrastructure. Relational computation can remain in the existing data platform, and outputs feed the customer's existing ML and MLOps workflows. These deployment options support sovereign AI by keeping data, models and execution under the operator's control 34, 35.

The frozen DuckDB-based open-source implementation evaluated here is the reproducible benchmark reference. Warehouse integrations and production operationalization are available through the Kurve product; this report does not separately benchmark those product deployments.

12 Execution, memory and reproducibility

12.1 Preserving values while reducing unnecessary work

The frozen implementation captures historical frames on disk and concatenates them in bounded batches. This reduces simultaneous full-frame retention while preserving cohort order, source sampling and model-visible values. Normalized feature preparation constructs columns without a whole-frame deep-copy cycle; temporal fitting uses shallow views where safe. Once normalization finishes, small routing/target headers are detached so a Series view does not keep a large raw numeric block alive.

Text heads select their few categorical columns before filtering training rows, and exposure queries project only the required routing/time columns. These changes remove unrelated payload copies rather than reducing feature counts or training budgets. Bootstrap acceleration caches score order and entity layouts while retaining the selector's declared resampling rule. Completed contract- matching trials and readout cases can be resumed without repeating

their fits.

GraphReduce additionally narrows thread use for very wide eager grouped queries with many distinct aggregates. Temporarily using one SQL thread for this specific operation avoids multiplying large per-thread aggregation states; the previous setting is restored afterward. Qualification compared numerical features, normalized learner inputs and predictions before accepting this execution repair. It is included in the private GraphReduce candidate bytes, not established by the version string alone.

These measures improve avoidable overhead. They do not make a wide graph cheap. Peak memory can still include phase source tables, intermediate SQL states, DuckDB buffers, wide normalized matrices, fitted models and temporary arrays. One-frame streaming is not a claim that only one frame-sized allocation exists anywhere in the complete procedure.

12.2 Parallelism and resource admission

Task-level and graph-search parallelism are different knobs. The frozen run uses sequential graph candidates and sequential temporal seed workers within a task, with resource-aware parallel admission across tasks. The server permits up to eight task workers against a 64-thread scheduling budget and reserves 64 GiB for host headroom. Actual concurrent task count depends on their future memory reservations, rather than the currently observed low RSS alone.

Reservations derive from completed resource observations with headroom. They are not a 64-GiB per-task kill ceiling. A DuckDB working-memory setting governs that engine's spilling; it does not bound total Python/native RSS, and graph- search commands can explicitly run without an engine memory cap. CPU thread slots are scheduling reservations, not affinity limits.

Each task has one original 23.75-hour cumulative wall deadline. Preparation and interrupted elapsed time count; resume does not reset it. A resource stop preserves completed checkpoints without claiming a completed prediction. The current fresh campaign has no automatic pressure retries. This enforces a budget below 24 hours; it does not prove that every task will successfully complete on arbitrary hardware within that budget.

12.3 Identity and artifacts

Checkpoints bind to task/context, configuration, feature audit, fidelity, implementation and runtime identities. Identical version labels are insufficient for reuse during editable development. Completed artifacts include frozen plans, feature schemas, temporal support/member audits, applicability records, per-origin/seed prediction evidence, locked selection/admission files and keyed final predictions. A changed context or procedure cannot silently resume an old run.

Fresh reproduction starts with empty task model/search outputs. Sharing raw datasets or verified pretrained weights is allowed; importing an old graph, prediction or remembered official score is a different operation. The current all-21 campaign made no such substitutions. Every current-system score below comes from that same fresh campaign and frozen source identity.

13 Evaluation and current evidence

13.1 Protocol

The benchmark uses the 21 RelBench v1 entity classification/regression tasks in RelArena, official temporal splits and the official task evaluator. Classification reports test ROC-AUC; regression reports test MAE. Current-system task results have deployment seed 0. The core selection path does not receive held-out answers; the harness scores saved predictions after the selected procedure is locked. RelBench, RelArena.

Research involved repeated inspection of benchmark results while developing the recipe. A fresh deterministic rerun tests implementation reproducibility; it is not an untouched new statistical test set. The aggregate comparison also uses reported competitor results from the captured panel rather than rerunning all methods in a matched hardware/time experiment.

13.2 Fresh 21-task scorecard

All current-system entries below are completed official scores from the same fresh campaign. Full-precision values, result/prediction hashes and the shared source identity accompany the report in the score supplement.

Task	Metric	Published	Staged	Client	Winner
Amazon / item-churn	AUC	0.827256	0.829162	0.828653	Kurve
Amazon / item-ltv	MAE	42.273080	42.575857	45.270515	Kurve
Amazon / user-churn	AUC	0.710347	0.711315	0.711698	Client
Amazon / user-ltv	MAE	14.020545	14.005336	13.877122	Client
Avito / ad-ctr	MAE	0.03344026	0.03281736	0.03125771	Client
Avito / user-clicks	AUC	0.658406	0.647785	0.691631	Client
Avito / user-visits	AUC	0.674721	0.673024	0.672453	Kurve
Event / user-attendance	MAE	0.25866836	0.23748723	0.24165168	Kurve
Event / user-ignore	AUC	0.832628	0.843941	0.784037	Kurve
Event / user-repeat	AUC	0.779679	0.806016	0.760495	Kurve
F1 / driver-dnf	AUC	0.746484	0.740482	0.727809	Kurve
F1 / driver-position	MAE	3.863205	3.960206	3.757398	Client
F1 / driver-top3	AUC	0.664957	0.803766	0.791702	Kurve
H&M / item-sales	MAE	0.03216765	0.03173097	0.03647878	Kurve
H&M / user-churn	AUC	0.696916	0.711248	0.706021	Kurve
Stack / post-votes	MAE	0.06324286	0.06273966	0.06273299	Client
Stack / user-badge	AUC	0.877626	0.877817	0.883443	Client
Stack / user-engagement	AUC	0.903027	0.907724	0.907662	Kurve
Trial / site-success	MAE	0.40384204	0.39135221	0.37761499	Client
Trial / study-adverse	MAE	41.270229	35.212072	34.401374	Client
Trial / study-outcome	AUC	0.712131	0.755106	0.764556	Client

Against the published KurveRSC column, the fresh procedure improves 16 tasks and declines on five. The largest clear changes include driver-top3, study-outcome, study-adverse and user-attendance. User-clicks, driver-position, user-visits, driver-dnf and item-ltv are worse than their captured published KurveRSC scores. Thus “no regression” is not an accurate description of the net change from the published system. The fresh repeat checks answer a narrower question: whether the frozen current pipeline reproduces its earlier current- pipeline scores after packaging and execution repairs.

Against the captured client, the current scorecard is 11 wins and 10 losses. Some wins are extremely narrow, notably user-engagement. Post-votes remains a loss at full precision despite close displayed values. A pairwise win is a point comparison, not proof of statistical separation.

13.3 Aggregate RelArena statistics

The official `relarena.evaluation.compute_leaderboard` replaces the captured published KurveRSC rows with these 21 scores while retaining the other 14 methods, the same tasks and the constant-global anchor at 1000. This is a recalculation on a fixed panel, not a claim that a public leaderboard has already accepted the candidate.

Rank	Method	Elo	Reported + / -	Mean rescaled loss
1	KurveRSC staged	1948.6	+101.5 / -76.2	0.073390
2	TabPFN-Rel client (Sep 28)	1937.1	+105.8 / -67.5	0.073884
3	TabPFN-Rel local (Sep 28)	1893.5	+92.6 / -62.2	0.075685
4	RT-J	1839.7	+145.5 / -118.9	0.153949
5	RT-PluRel	1792.1	+106.5 / -96.8	0.138969

The point Elo lead over the September 28 client is 11.5. The reported Elo uncertainty ranges overlap substantially. RelArena's leaderboard intervals describe task-bootstrap variation, rather than variation across training seeds. Leaderboard protocol. Mean rescaled loss also favors KurveRSC on this final panel: 0.073390 versus 0.073884 for the client. First on these aggregate point estimates does not imply superiority on every task. Head-to-head wins alone do not determine Elo against the complete panel.

13.4 Fresh reproduction status

All 21 tasks and final scorer/resource jobs completed, with no failed tasks, automatic pressure retries or historical score substitutions. Twenty tasks have an earlier comparable current-pipeline execution: 19 scores match exactly; H&M item-sales improves from 0.03234723 to 0.03173097 MAE. User-badge is this pipeline's first completed result and is not counted as an independent repeat.

The release review uses KurveRSC 0.2.0 and GraphReduce 1.10.18. All 177 installed Python modules (including the generated stage namespace initializer) match the score-producing frozen payload before version stamping. Full classification and regression runs through the current RelArena registered class and actual core split types reproduce the frozen wheels' keyed predictions exactly on a 48-request unregistered schema. These checks reuse the qualified dependencies; they are not a cold resolution of every optional dependency, a cross-platform bitwise guarantee or an independent current-harness rerun of all 21 tasks.

14 Measured computational cost

14.1 Hardware and measurement definitions

The fresh server has 495.77 GiB usable RAM, 64 logical CPUs on 32 physical cores of an Intel Xeon Platinum 8488C, and a 992.15-GiB filesystem. The pipeline runs on CPU. Timings below cover the full task, including search, materialization, temporal checks, fitting and scoring under the campaign's concurrent load.

The principal RAM measurement is the kernel's full-task cgroup-v2 `memory.peak`, covering descendants and charged file cache/kernel memory. A separate one-second sampled tree RSS sum can double-count shared pages and has different accounting. Peak busy CPUs come from sampled cgroup CPU-time increments per wall second. Disk is a 60-second sampled allocated task-owned footprint, including outputs, temporary directories, logs and metrics; short-lived peaks may be missed.

The measured DuckDB working-memory ceilings are 123 GiB for 19 tasks, 224 GiB for user-badge and 256 GiB for Amazon user-churn. These explicit resource allocations are recorded per task in the supplement and can be set through `KURVERSC_DUCKDB_MEMORY_LIMIT`. They govern SQL aggregation/spilling, not total process RSS, source-row sampling or feature/learner budgets.

Task	Hours	RAM GiB	RSS GiB	Busy CPUs	Disk GiB
Amazon / item-churn	3.25	132.87	123.53	8.03	21.18
Amazon / item-ltv	5.68	182.17	161.33	8.03	22.52
Amazon / user-churn	3.65	203.72	194.17	8.07	21.64
Amazon / user-ltv	5.60	152.40	143.01	10.52	21.02

Task	Hours	RAM GiB	RSS GiB	Busy CPUs	Disk GiB
Avito / ad-ctr	0.72	67.24	66.71	9.09	1.43
Avito / user-clicks	1.14	52.03	51.42	8.69	1.59
Avito / user-visits	1.75	89.02	88.37	8.62	1.55
Event / user-attendance	2.10	116.77	116.20	8.02	0.55
Event / user-ignore	1.45	33.04	31.37	8.06	0.70
Event / user-repeat	1.58	32.54	32.69	8.03	0.58
F1 / driver-dnf	5.68	6.70	6.98	8.06	0.24
F1 / driver-position	4.61	4.14	4.51	8.03	0.19
F1 / driver-top3	3.55	2.88	3.34	8.04	0.10
H&M / item-sales	4.90	26.53	24.40	8.03	2.10
H&M / user-churn	3.39	44.95	44.87	8.06	4.08
Stack / post-votes	5.64	61.63	58.24	11.90	11.80
Stack / user-badge	14.17	333.51	319.19	8.01*	13.35*
Stack / user-engagement	4.82	91.00	86.89	8.05	11.52
Trial / site-success	4.37	91.67	89.96	8.05	2.50
Trial / study-adverse	8.94	26.09	24.54	8.02	2.63
Trial / study-outcome	2.69	26.90	25.54	8.03	2.65

An asterisk marks partial CPU/disk coverage. User-badge's CPU/disk observer attached 5.38 hours after task start, so those two figures are partial-coverage observations. Its RAM peak and runtime cover the whole task. The other 20 tasks were observed from near their start. Shared raw RelBench files occupy approximately 17.49 GiB; cached encoder weights 0.09 GiB; and the frozen runtime 1.87 GiB. These shared allocations are outside the per-task disk column and should not be multiplied blindly by task count.

These are observed requirements on this host, not certified minimum hardware. All 21 task resource receipts are finalized. The resource supplement preserves full precision, measurement coverage and completion status. The largest observed task is user-badge at 333.51 GiB and 14.17 hours. A comparable 512-GiB-class host with headroom is the measured deployment class, rather than a certified minimum. Independent tasks can run concurrently on larger machines using memory/CPU/storage reservations; preserve the qualified eight learner threads, one graph-search worker and one seed worker per task.

14.2 Cost relative to the earlier system

The staged procedure adds work after graph selection: materialized history, phase-frozen temporal audits, multiple seeds, optional heads and independent native replay. Smaller screening frames do not remove these costs. Candidate counts, cumulative fit counts, task wall time and total campaign wall time are different quantities. The final same-freeze timings establish the current full-task costs. A matched earlier-system execution is still needed to quantify incremental overhead or a clean optimization speedup, since hardware load and runtime lineage differ.

15 Separate new-schema check: Olist seller churn

The frozen packaged procedure was also run on the curated seven-table Brazilian e-commerce example from the pinned Prior Labs cookbook notebook. The task predicts whether a seller active in the preceding 30 days receives no order in $(t, t + 30 \text{ days}]$. The curated tables and column allowlists follow the notebook; post-purchase fields excluded by the example remain excluded. All 13,382 labels across 20 anchors were independently checked against the notebook SQL. The model context excludes held-out answers.

Quantity	Observed value
Training	11,135 examples across 18 cutoffs
Validation	1,159 sellers, May 15, 2018
Test	1,088 sellers, June 15, 2018
KurveRSC held-out ROC-AUC	0.787511565
Notebook's recorded TabPFN-Rel ROC-AUC	0.784, rounded; API not rerun
Reproduced per-entity baseline	0.688185633
Successful complete-pipeline runtime	46.16 minutes
Sampled process-tree RSS peak	3.84 GiB on a 48-GiB Mac
Selected graph	Depth 2; base, temporal, sequence; 400 features
Selected temporal policy	50% pooled-12 learner / 50% incumbent frame ensemble

The small point advantage over the rounded notebook reference is not a fresh paired TabPFN comparison or a significance result. Source timestamp maxima in 10,907 cached historical rows and 1,159 validation rows passed the cutoff check, with intraday precision retained. Olist is outside the 21-task Elo panel.

An earlier local attempt with pandas 3.0.5 failed on a read-only mask after 19.44 minutes. The successful cold attempt aligned its isolated environment to the server's pandas 2.3.3, reused no fitted graph/model, and retained the original task deadline. Total wall time including that failed attempt was 69.77 minutes. This is evidence for the qualified runtime; it also identifies an unresolved compatibility boundary in the package's broader dependency range.

The successful run spends about eight minutes on graph search, 9.85 minutes on the temporal audit, 22 minutes on optional signal comparisons, and 3.84 minutes on recovered-readout checks. Therefore, even perfect graph proposals alone would remove only part of this workflow's runtime. The executable experiment is provided as an example.

16 Related work and methodological context

16.1 Automated relational feature construction

Deep Feature Synthesis composes transformations along relational paths; One Button Machine also automates relational joining and feature discovery [8, 9]. These precedents are essential: automating relational feature creation is not itself new. KurveRSC's present focus is a temporally valid, bounded selection procedure over whole relational programs, with the downstream fitting policy included in the final comparison. GraphReduce supplies the executable relational operations [2]. Its output matrix is an explicit interface between representation and estimation, not evidence that cross-table reasoning was absent.

16.2 Relational deep learning and foundation learners

GraphSAGE learns sampled neighborhood aggregation [11]. RelGNN introduces schema-aware composite message passing, while RelGT uses attention and relational positional/temporal structure [12, 13]. RelBench supplies the shared predictive tasks [3]. These methods are relevant alternatives because they learn representations within the relational architecture rather than choosing SQL reduction programs.

KumoRFM and KumoRFM-2 extend relational representation learning through pretraining and in-context prediction [14, 15]. Their cross-task amortization and deployment objectives are different from fitting a task-specific compression program and learner. TabPFN and the TabPFN-3 report establish the tabular foundation-model lineage [18, 19]. The subsequent TabPFN-3.5 technical report [30] also reports relational-harness results; it is a distinct

paper with a distinct model/version scope. RDBLearn and TabPFN-Rel connect tabular learners to relational materialization 17, 16. They are independent support for the representation/learner boundary. This report does not turn comparisons with a particular captured TabPFN-Rel client into claims against every later model or pretraining regime.

16.3 Aggregation, readouts and supervision

Deep Sets studies invariant functions over sets, and Xu et al. analyze the distinguishing power of neighborhood-aggregation GNNs 20, 21. Together they motivate asking which distinctions survive a particular representation. They do not imply that a small fixed set of SQL moments preserves every graph property. Conversely, using learned aggregation does not by itself remove all compression or expressivity constraints.

CatBoost supplies the main mixed-type learner 5. The semantic challenger uses a frozen sentence representation from the Sentence Transformers lineage 26, with the actual MiniLM weights and training mixture identified separately 6. Distribution-aware readouts are also conceptually separate from new relational inputs: absolute-error loss targets a conditional median, making quantile modeling relevant to the regression heads 27. An improvement from a better readout is not automatically evidence that another graph path was discovered.

16.4 Multi-fidelity selection, statistics and execution

Successive Halving, Hyperband and BOHB provide important precedents for allocating more resources to promising configurations 22, 23, 24. The present staged policy is not an implementation of their algorithms or a beneficiary of their theoretical guarantees. Relational fidelity can alter connected source cohorts, categories and discovered operations, rather than merely extending a single learner's training curve. A frozen plan improves comparison consistency but also freezes discovery omissions. This makes measured rank fidelity and winner retention especially important.

Bootstrap methods provide general resampling context 28. The entity-clustered temporal gate, completeness requirements and multiplicity allowances here are the implemented procedure's own choices, rather than a guarantee supplied by that reference. RelArena defines the benchmark comparison and aggregate statistics 4, 16, and DuckDB provides the embedded analytical execution layer 25. Reporting both predictive outcomes and resource cost is necessary when comparing task-specific search with pretrained inference.

17 Limitations and next work

The principal unresolved questions concern selection fidelity and cost. The budgeted search can still discard a graph that would win at full fidelity; a frozen plan discovered from capped source cohorts can miss rare categories or fine temporal interactions. CDFs, scalar reductions and bounded optional heads cannot represent arbitrary joint neighbor structure. Optional capacity limits also mean that an available signal may be evaluated on one schema and excluded on a larger one. Further controlled studies must measure those failure modes rather than assume the largest graph is optimal.

The current evidence does not isolate each addition's causal value. A clean ablation should preserve data clocks, search budgets, final fitting and runtime while changing one representation or selector component. The completed fresh run supports implementation reproducibility under the frozen environment; independent maintainer reproduction remains outstanding. Repeated benchmark development and overlapping Elo uncertainty also limit claims of general superiority.

Exact intermediate sharing and additional replay-equivalent execution improvements are nearer-term opportunities. A longer-term direction is to amortize representation search across databases.

18 Next direction: a foundation model for graph planning

18.1 Predict a useful program from a task-conditioned schema

A planner pretrained across many databases could propose GraphReduce programs for an unseen schema, reducing the need to rediscover useful depth, paths, feature families and column budgets for every task. This is a plausible research direction because the output is a structured, capability-constrained program, and KurveRSC already generates comparative supervision while searching. It is not evidence that the ideal graph is identifiable from a

schema alone. Two databases with the same schema and marginal statistics can have different relationships between historical inputs and the target.

The input should therefore combine the schema with the prediction task and cutoff-safe profiles: keys, types, cardinality, missingness, relationship coverage and fan-out, event cadence, time coverage, target kind, root entity, horizon and available training support. A resource profile should identify the learner, backend, memory and time budget. Column and task descriptions could supply an additional semantic channel. Optional supervised probes must use only planning-time training data, and their cost must be counted.

The output should be a valid program or a short ranked set of programs, with predicted uncertainty and resource demand. Candidate generation and compilation would still enforce legal paths, temporal boundaries and backend capabilities. Confidence can determine whether to use the first proposal, evaluate a small set, or fall back to bounded search. Here **zero-shot planning** means no new graph search or planner retraining on the destination database; the downstream supervised learner still fits its observed labels. Evaluating proposed graphs on the new task is a few-probe variant and should be reported separately.

18.2 Existing scaffold and the data repository it needs

The repository already contains an initial implementation for collecting `PlannerEpisode` traces from explicit `DatabaseProfile`, `TaskProfile` and `RuntimeProfile` objects. Its first model is a CatBoost meta-ranker, not a trained foundation planner or the selector used for this report. It tests the profile, supervision and database-grouped split contracts before a possible schema-aware graph transformer. The planner training guide documents the collector, JSONL format and training entry point.

The main bottleneck is a **repository of relational search outcomes**, rather than a repository of schemas alone. Each teacher episode should retain the task and cutoff, pre-execution profiles, candidate program, fidelity, validation predictions or scores, uncertainty, cost, width, failure reason and source/runtime identity. Low- and full-fidelity observations must remain distinct. A candidate that search never evaluated is censored, not a demonstrated loser. Winner-only traces would teach the new model to repeat the current selector's omissions. Matched full-fidelity comparisons and exhaustive small-schema audits are needed to calibrate that bias.

Seven original RelBench databases and 21 related tasks are a starting point, not enough independent schema diversity to establish a foundation model. A practical corpus can combine three sources:

Source	Contribution	Required care
Public relational databases	Genuine schemas, fan-out, temporal structure and task outcomes	Deduplicate related sources; verify licenses and construct valid task/time contracts
Synthetic relational generators	Controlled variation of topology, cadence, sparsity, rare categories and target mechanisms	Hold out generator families and check transfer to real databases
Permissioned operational traces	Deployment diversity and measured backend/resource costs	Collect locally by default; share only with permission and a defined data policy

Concrete public starting points exist. RelBench v2 expands the benchmark and interfaces with ReDeLEx for access to more than 70 real-world databases; it also incorporates 4DBInfer coverage 31. 4DBInfer provides multi-table datasets, prediction tasks and an evaluation toolbox 32. The CTU Prague repository supplies SQL databases with searchable schema metadata 33. These collections still need standardized task definitions, availability clocks and newly measured graph outcomes before they become planner training episodes. Their overlap also prevents treating their database counts as additive. Single-table collections alone cannot teach relational path selection.

The existing trace contract stores aggregate profiles rather than raw rows or categorical values. That reduces the sharing burden, but does not make schema and aggregate metadata automatically anonymous. A sovereign deployment can keep both traces and the trained planner locally; participation in a shared corpus is a separate decision.

18.3 Evaluation before replacing production search

All tasks, cutoffs and candidates from one database must stay in one partition; related or duplicated source databases must also be grouped. Final evaluation should include entirely unseen schemas and source domains. Report full-

fidelity validation regret, top-k retention of the audited winner, infeasible proposals, profiling cost, total runtime and memory. Lock the policy before evaluating held-out prediction targets. Table names and benchmark identifiers must not become shortcuts for remembered winning configurations.

The first experiment should compare the existing metadata ranker against a fixed budgeted search on held-out databases. A schema-aware model with table, relationship and column pointer heads is justified only if broader traces improve that baseline. Subsequent outputs can cover temporal support and readout policies as well as the graph. This matters for end-to-end latency: graph search accounted for only about eight of the Olist run's 46 minutes, so predicting its graph alone would leave most of that procedure's measured cost.

The objective is a transferable, uncertainty-aware prior over useful relational programs. Replacing every new-database search with an unverified claim of an optimal graph would skip the selection-fidelity question this report identifies.

19 Appendix A: Frozen runtime and reproduction identity

The qualified Linux runtime uses Python 3.11.17, RelBench 2.1.2, CatBoost 1.2.10, DuckDB 1.2.2, pandas 2.3.3, NumPy 1.26.4, scikit-learn 1.6.1, PyArrow 20.0.0, PyTorch 2.13.0+cpu and Transformers 4.57.6. The preparation artifact contains 114 pinned dependency requirements alongside three private candidate wheels. All 114 entries were checked against `importlib.metadata.version` in the frozen task environment, with zero mismatches. The raw enumerated distribution inventory is recorded separately; it also contains vendored metadata and is not itself a resolved lockfile. Install from the exact candidate artifacts and dependency lock before using the complete staged entry point; broad compatible version ranges are not a reproduction specification.

graphreduce-1.10.17-py3-none-any.whl candidate wheel.

`dbf00d70a340e3707f6315ffb350caf746a040b738a0c1b60d60120c913450fc`

kurversc-0.1.1-py3-none-any.whl candidate wheel.

`04d066bb09185bed1bc687e1bcf30bfcc0349afdcc5800540f391e1e6a890692`

relarena-0.0.1-py3-none-any.whl candidate wheel.

`6dc821806d37409f1d0128940cd37d0650c71f9988a18a243254176593be8ca2`

Source archive.

`c5478149c985f1448822681aecf0f9abd5ff6fe18a8b46c1af97ac2f75595fd8`

114-requirement dependency lock.

`8e37df1ed0cb6f0d4a61f47a5d0ac431c9e4bad69bde88691f2fe14576380b2c`

Fresh run source identity.

`7b3c8832547a490d180cbc849b6feb039e3e3ada66c01225da5b51713ec931ae`

Captured 15-method / 21-task panel.

`85a1b1caecd2b878aadf81b64bda8da205efb9874f395b30654bc075b3c4a5fa`

The public portable entry point is `kurversc.run_staged_task`. The submitted current-core adapter is `relarena.models.kurversc.staged.KurveRSC20261022System.run`, with alias `kurversc-latest`. The frozen campaign used its original registered `relarena.models.kurversc.model.KurveRSCSystem.run` bridge to the same procedure. Fresh outputs, source identity, effective stage contract, dependency lock, encoder hashes, request keys and full-stage completion receipts belong in the eventual reproduction bundle. The current root source checkout may contain unrelated research scripts; only the packaged transitive stage closure and the frozen identities define this report's procedure.

20 Appendix B: Implementation map

Paths below refer to source modules packaged into the frozen candidate. Stage modules are relocated under `kurversc._staged` in the wheel. Their research-era filenames/docstrings do not override the effective frozen contract: several stages originally developed as opt-in experiments are included automatically in `run_staged_task()` today.

Responsibility	Source
Task/schema/availability contract	task_context.py, task_source.py
Supervised complete entry point	staged.py, run_portable_staged.py
Complete stage composition	run_reproducible_staged_cloud.py
Bounded graph profile	run_kurversc_selected_relarena.py
Graph candidates and selection	search.py, core.py
Historical labels and schema expansion	core.py, relbench.py
Column audit	feature_audit.py
Active peer eligibility and summaries	peer_context.py, peer_history.py
Temporal support, fitting and gate	temporal_policy.py, benchmark_temporal_policy.py
Production MAE finalist check	benchmark_production_finalists_v2.py
Signal and recovered readouts	signal_challengers.py, benchmark_recovered_readouts.py
Sequence and complementary views	benchmark_production_sequence.py, run_bounded_view_validation_20261020.py
Learner boundary and identities	modeling.py, checkpointing.py, prediction_provenance.py
Stage packaging	staged_build.py
Full-task resource observer	observe_submission_resources_20261025.py

21 Appendix C: Evidence

The compact machine-readable supplements are part of this report:

- Scores and per-task lineage.
- Completed fresh-task resource measurements.
- Snapshot identities, stage contract and aggregate results.
- Frozen dependency requirements.

Author-side copied scorer receipts, the captured 315-row comparison panel and full resource observer snapshot are retained separately under `results/technical-report-20261027/evidence/`. Those supporting files preserve the exact panel and run lineage; they are not new model-selection inputs.

22 References

The bibliography covers the implementation, benchmark protocol, competing architectures, feature-construction precedents, representation theory, resource-allocation methods and readout/statistical foundations. The BibTeX file and reviewed metadata accompany the report. Web and software references were checked on October 26, 2026.

1. W. Madrigal. *KurveRSC: Validation-Guided Relational Signal Compression with a Downstream Learner in the Loop.* Kurve AI, original technical report, 2026. Primary source.
2. W. Madrigal. *GraphReduce.* Open-source software. Primary source. Results in this report require the privately patched candidate identified in Appendix A.
3. J. Robinson, R. Ranjan, W. Hu, K. Huang, J. Han, A. Dobles, M. Fey, J. E. Lenssen, Y. Yuan, Z. Zhang, X. He and J. Leskovec. *RelBench: A Benchmark for Deep Learning on Relational Databases.* NeurIPS Datasets and Benchmarks, 2024. Primary source.
4. Prior Labs. *RelArena: Open and Reproducible Benchmarking for Relational Learning.* Open-source benchmark and source-provenance documentation. Primary source. This report uses a separately checksummed captured panel; baseline_results/SOURCES.md distinguishes reproduced results from paper-reference rows.
5. L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush and A. Gulin. *CatBoost: Unbiased Boosting with Categorical Features.* NeurIPS, 2018. Primary source.
6. Sentence Transformers. *all-MiniLM-L6-v2: Model Card and Weights.* Hugging Face model repository. Primary source. Pinned revision identified in the URL and the text-representation disclosure; includes external training-mixture documentation.

7. Prior Labs. *Relational Predictions with TabPFN-Rel.* TabPFN cookbook notebook. Primary source. Pinned commit 32b8db2; full commit is retained in the URL and experiment driver.
8. J. M. Kanter and K. Veeramachaneni. *Deep Feature Synthesis: Towards Automating Data Science Endeavors.* IEEE International Conference on Data Science and Advanced Analytics, 2015. Primary source. DOI: 10.1109/DSAA.2015.7344858.
9. H. T. Lam, J.-M. Thiebaut, M. Sinn, B. Chen, T. Mai and O. Alkan. *One Button Machine for Automating Feature Engineering in Relational Databases.* arXiv:1706.00327, 2017. Primary source.
10. M. Fey, W. Hu, K. Huang, J. E. Lenssen, R. Ranjan, J. Robinson, R. Ying, J. You and J. Leskovec. *Position: Relational Deep Learning - Graph Representation Learning on Relational Databases.* ICML, PMLR 235:13592-13607, 2024. Primary source.
11. W. L. Hamilton, R. Ying and J. Leskovec. *Inductive Representation Learning on Large Graphs.* NeurIPS, 2017. Primary source.
12. T. Chen, C. Kanatsoulis and J. Leskovec. *RelGNN: Composite Message Passing for Relational Deep Learning.* ICML; arXiv:2502.06784, 2025. Primary source.
13. V. P. Dwivedi, S. Jaladi, Y. Shen, F. Lopez, C. I. Kanatsoulis, R. Puri, M. Fey and J. Leskovec. *Relational Graph Transformer.* ICLR; preprint first released in 2025, 2026. Primary source.
14. M. Fey, V. Kocijan, F. Lopez, J. E. Lenssen and J. Leskovec. *KumoRFM: A Foundation Model for In-Context Learning on Relational Data.* Kumo.AI white paper, 2025. Primary source. Archived original; the former live PDF URL now redirects.
15. V. Hudovernik, F. Lopez, V. Kocijan, A. Nitta, J. E. Lenssen, J. Leskovec and M. Fey. *KumoRFM-2: Scaling Foundation Models for Relational Learning.* arXiv:2604.12596, 2026. Primary source. Section 1 contains the quoted criticism of generic task-agnostic aggregation.
16. A. Hayler et al.. *Advancing Open and Reproducible Relational Learning: RelArena-alpha, TabPFN-Rel and RPI.* arXiv:2608.16319, 2026. Primary source.
17. Y. Zhang, L. Xu, Q. Gan, D. Wipf and M. Wang. *RDBLearn: Simple In-Context Prediction Over Relational Databases.* arXiv:2602.18495, 2026. Primary source.
18. N. Hollmann, S. Muller, L. Purucker, A. Krishnakumar, M. Korfer, S. B. Hoo, R. T. Schirrmeister and F. Hutter. *Accurate Predictions on Small Data with a Tabular Foundation Model.* Nature 637:319-326, 2025. Primary source. DOI: 10.1038/s41586-024-08328-6.
19. L. Grinsztajn et al.. *TabPFN-3: Technical Report.* arXiv:2605.13986, 2026. Primary source.
20. M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. Salakhutdinov and A. J. Smola. *Deep Sets.* NeurIPS, 2017. Primary source.
21. K. Xu, W. Hu, J. Leskovec and S. Jegelka. *How Powerful are Graph Neural Networks?.* ICLR, 2019. Primary source.
22. K. Jamieson and A. Talwalkar. *Non-stochastic Best Arm Identification and Hyperparameter Optimization.* AISTATS, PMLR 51:240-248, 2016. Primary source.
23. L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh and A. Talwalkar. *Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization.* Journal of Machine Learning Research 18(185):1-52, 2018. Primary source.
24. S. Falkner, A. Klein and F. Hutter. *BOHB: Robust and Efficient Hyperparameter Optimization at Scale.* ICML, PMLR 80:1437-1446, 2018. Primary source.
25. M. Raasveldt and H. Muhleisen. *DuckDB: An Embeddable Analytical Database.* ACM SIGMOD, 2019. Primary source. DOI: 10.1145/3299869.3320212.
26. N. Reimers and I. Gurevych. *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks.* EMNLP-IJCNLP:3982-3992, 2019. Primary source. DOI: 10.18653/v1/D19-1410.
27. R. Koenker and G. Bassett, Jr.. *Regression Quantiles.* Econometrica 46(1):33-50, 1978. Primary source. DOI: 10.2307/1913643.
28. B. Efron. *Bootstrap Methods: Another Look at the Jackknife.* The Annals of Statistics 7(1):1-26, 1979. Primary source. DOI: 10.1214/aos/1176344552.

29. NVIDIA Kumo. *Relational Deep Learning.* Explanatory research documentation, section Why flat tables fail. Primary source. Accessed October 26, 2026; current explanatory documentation is distinguished from the original ICML paper.

30. B. Jäger et al.. *TabPFN-3.5: Technical Report.* arXiv:2609.17895; revised September 22, 2026, 2026. Primary source. Distinct from the earlier TabPFN-3 technical report.

31. J. Gu et al.. *RelBench v2: A Large-Scale Benchmark and Repository for Relational Data.* ICLR 2026; arXiv:2602.12606v2, 2026. Primary source. Corpus expansion and access to external relational databases; these resources are proposed planner inputs, not additional tasks in the reported 21-task scorecard.

32. M. Wang et al.. *4DBInfer: A 4D Benchmarking Toolbox for Graph-Centric Predictive Modeling on Relational DBs.* arXiv:2404.18209, 2024. Primary source. Primary paper and associated multi-table benchmark/toolbox.

33. J. Motl and O. Schulte. *The CTU Prague Relational Learning Repository.* arXiv:1511.03086; revision March 26, 2025, 2015. Primary source. Repository and schema metadata; valid prediction tasks and planning outcomes must still be constructed.

34. Kurve AI. *Kurve Product and Deployment.* Product overview and deployment options. Primary source. Primary product source for warehouse integrations, Snowflake Native App, customer-controlled deployment and existing ML/MLOps workflows; product statements are distinct from benchmark results.

35. Kurve AI. *Kurve Product Documentation.* Deployment, metadata graphs and compute graphs. Primary source. Primary documentation for private deployment and operational graph workflows.