INFER: FEATURE ENGINEERING AS INFERENCE WITH TRANSFERABLE REINFORCEMENT LEARNING AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Feature engineering, the transformation of raw columns into predictive features, is one of the most time-consuming steps of tabular machine learning. Automated feature engineering (AutoFE) faces an exponentially growing candidate space and expensive per-candidate evaluation, and nearly every existing method pays for a fresh search on every new dataset. We propose INFER (*INductive Feature Engineering via Reinforcement*), which learns feature engineering *once, offline*, and applies it to any new dataset by *inference*. INFER is a hierarchical deep reinforcement learning framework with two cooperating agents: an *outer* agent reads dataset meta-features and selects a small subset of promising columns, and an *inner* agent composes operators over that subset to build a new feature. Both agents are trained on a collection of datasets and then applied, with frozen weights and no retraining, to previously unseen datasets. Feature generation on a new dataset therefore costs only forward passes and a lightweight selection step, with the number of generation runs as an explicit compute budget. On 75 binary-classification datasets, under a strict nested protocol in which every candidate is generated and fitted on training rows only, INFER improves over the original features on 81–83% of datasets with both XGBoost and RandomForest. It outperforms the RL-based method DIFER under both classifiers and the state-of-the-art OpenFE under XGBoost, and matches OpenFE under RandomForest. Code: `https://github.com/infer2026/INFER`

1 INTRODUCTION

Feature engineering is one of the most consequential steps in applied machine learning. Good features expose structure that the learning algorithm cannot easily recover on its own. Creating them by hand works well, but it is slow and requires expertise; automating it promises to free practitioners from this effort and to make machine learning more accessible.

Automated feature engineering (AutoFE) remains hard in practice. The first obstacle is structural: nearly all methods, including the reinforcement learning (RL) ones, run a new search on every dataset. Whatever the search learns is discarded when the next dataset arrives, so a practitioner with a hundred datasets pays for a hundred searches. Nothing is amortized, and the cost of a method is fixed by its internal search rather than chosen by the user. The other two obstacles are the usual ones: the space of candidate features grows exponentially with the number of operators and their composition depth, and evaluating a candidate means training a model. A recent usability study of 53 AutoFE methods found that most are hard to use and that none let the user bound time or memory, which the authors consider a prerequisite for practical automation (Schäfer et al., 2025).

We take a different approach. We treat feature engineering as a skill that can be *learned once and transferred*. Our framework, INFER (*INductive Feature Engineering via Reinforcement*), trains two deep RL agents on a collection of datasets and then applies them, with frozen weights, to datasets they have never seen. The search is hierarchical. An *outer* agent reads dataset meta-features and selects a small subset of promising columns. An *inner* agent then composes operators over that subset to build a new feature. Because the agents are not retrained on the target dataset, generating features for it costs only forward passes of two small networks plus the materialization of the result-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

ing candidates. The number of generation runs is therefore an explicit compute budget. The name reflects this property: on a new dataset, feature engineering becomes policy *inference*.

Contributions.

- We propose INFER, a hierarchical two-agent DRL framework that splits AutoFE into a feature-*selection* problem, solved by an outer agent operating on dataset meta-features, and a feature-*construction* problem, solved by a recurrent inner agent that composes operators sequentially.
- INFER is *inductive*: the agents are trained once, offline, and applied to disjoint, unseen datasets without retraining. A single pair of frozen agents replaces per-dataset search on all 75 benchmark datasets and both downstream classifiers, and a dataset-level four-fold protocol ensures that every reported number comes from datasets the agents never saw during training.
- We introduce a generation-then-selection inference procedure that is *leakage-free by construction*: for every outer cross-validation fold, candidate features are generated, fitted, and selected on training rows only. This closes a gap that is easy to open in AutoFE evaluations, where target-dependent statistics are computed before the data is split.
- On 75 heterogeneous binary-classification datasets, INFER improves over the original features on 81% (XGBoost) and 83% (RandomForest) of datasets. It outperforms the RL-based method DIFER under both classifiers and OpenFE under XGBoost, and matches OpenFE under RandomForest, while moving the search off the target dataset.

2 RELATED WORK

Generate-then-select AutoFE. Much of the AutoFE literature builds a large pool of candidate features and then filters it (Ying et al., 2025). The Data Science Machine (Kanter & Veeramachaneni, 2015) introduced Deep Feature Synthesis over relational tables. Cognito (Khurana et al., 2016) applies a greedy hierarchical search over transformations. ExploreKit (Katz et al., 2016) generates candidates with unary, binary, and higher-order operators and ranks them with a classifier trained offline on meta-features, and Learning Feature Engineering (Nargesian et al., 2017) recommends transformations from feature representations. OpenFE (Zhang et al., 2023) combines a boosting-based importance estimator with two-stage pruning and reaches expert-level performance, and autofeat (Horn et al., 2020) offers accessible generation and selection; generative approaches cast transformation as learning in a latent space (Ying et al., 2024). INFER follows the generate-then-select paradigm, but the generator is a learned, transferable policy rather than a per-dataset search.

RL for feature engineering. Several works frame AutoFE as sequential decision making. Khurana et al. (2018) traverse a transformation graph under a fixed budget with Q-learning. Neural Feature Search (Chen et al., 2019) trains an RNN controller with RL, and MARLFS (Liu et al., 2019) uses one agent per feature. DIFER (Zhu et al., 2022) optimizes features in a differentiable embedding space. Recent work includes a dual-agent design in which one agent proposes features and a second decides whether to keep them (Gao et al., 2025), and a multi-agent framework for scientific data (Xiao et al., 2025). INFER differs on two axes. First, it uses a *hierarchical* decomposition that separates *which* columns to combine (outer agent) from *how* to combine them (inner agent). Second, it is *inductive*: the policy is trained once and reused. The methods above learn or refine their policy on the target dataset itself; DIFER, for example, keeps a separate controller checkpoint per dataset. We compare against DIFER directly in Section 5.

LLM-based feature engineering. CAAFE (Hollmann et al., 2023) prompts an LLM with dataset descriptions to propose features. LLM-FE (Abhyankar et al., 2026) frames the task as LLM-guided evolutionary program search, and collaborative approaches add human feedback to the loop (Li et al., 2026). These methods need textual metadata and repeated LLM inference; INFER needs neither.

RL and meta-learning background. DRL is effective for searching large discrete spaces (Dulac-Arnold et al., 2015). We use Proximal Policy Optimization (PPO; Schulman et al., 2017) for the outer agent's selection decisions and a Deep Recurrent Q-Network (DRQN; Hausknecht & Stone, 2017) for the inner agent's partially observable, step-by-step construction. Meta-learning lets a system use experience across tasks to act well on new ones (Hospedales et al., 2022); in feature generation, it enables training on existing datasets and fast generation on unseen ones (Reyes-Amezcua et al., 2022). This property underlies the inductive design of INFER.

3 THE INFER FRAMEWORK

3.1 PROBLEM FORMULATION AND REWARD DESIGN

Let $D = (X, y)$ be a supervised dataset with feature set $\mathcal{F}_D = \{f_1, \ldots, f_n\}$ and target y. Let $\mathrm{AUC}(\mathcal{F})$ be the ROC AUC of a fixed downstream classifier trained on feature set $\mathcal{F}$. We are given an operator library $\mathcal{O} = \mathcal{O}_{\mathrm{u}} \cup \mathcal{O}_{\mathrm{b}} \cup \mathcal{O}_{\mathrm{h}}$ of unary, binary, and higher-order operators (Section 3.3). Applying operators to existing features yields new candidate features. The goal is to construct a set $\mathcal{G}$ of new features such that $\mathrm{AUC}(\mathcal{F}_D \cup \mathcal{G}) > \mathrm{AUC}(\mathcal{F}_D)$, using policies that are trained once and transferred to unseen datasets.

We decompose this into two coupled decision processes. The *outer* problem is a feature-selection MDP $\mathcal{M}_O = (\mathcal{S}_O, \mathcal{A}_O, R_O)$. Its state $s_O = \phi(D)$ is a vector of dataset meta-features. An action $a_O \in \mathcal{F}_D$ adds one column to a working subset F until $|F| = k$. The policy π_O is rewarded by the improvement obtained from the feature later built from F. The *inner* problem builds one feature from F. A feature is assembled incrementally, and the validity and value of each step depend on the operators and operands accumulated so far. A single observation is therefore not a sufficient statistic. We model the inner problem as a partially observable MDP $\mathcal{M}_I = (\mathcal{S}_I, \mathcal{A}_I, R_I)$ with a recurrent policy $\pi_I(a_t \mid o_t, h_t)$. Here o_t summarizes the construction stack and the column-level meta-features $\psi(\cdot)$ of the operands, h_t is the recurrent hidden state, and an action in $\mathcal{A}_I = F \cup \mathcal{O}$ either pushes an operand onto a construction stack S or applies an operator to it. Episodes last at most $T_{\max}$ steps.

We measure the usefulness of a constructed feature by the relative error reduction (RRE) in AUC:

$$\mathrm{RRE}(\mathcal{F}, \mathcal{F}') = \frac{\big(1 - \mathrm{AUC}(\mathcal{F})\big) - \big(1 - \mathrm{AUC}(\mathcal{F}')\big)}{1 - \mathrm{AUC}(\mathcal{F})}, \tag{1}$$

where $\mathcal{F}' = \mathcal{F} \cup \{f_{\text{new}}\}$. The outer agent is driven directly by this objective. Once the inner agent has built a feature from the selected subset, the outer reward is the delayed signal $r_O = \mathrm{RRE}(\mathcal{F}_D, \mathcal{F}_D \cup \{f_{\text{new}}\})$.

The inner agent faces a long sequential construction, for which a single terminal RRE would be a very sparse signal. We therefore use a shaped step reward that stays aligned with the RRE objective. An intermediate operation that raises the AUC of the feature under construction receives $+5$. Completing a valid feature yields a terminal bonus of $+10$ if the feature improves AUC and $+1$ if it is valid but does not. An action that violates the stack grammar (e.g., a binary operator with one operand) receives -1. All other steps receive 0. This shaping speeds up learning of both the grammar and the operator compositions that raise AUC, while the delayed RRE reward keeps the outer policy tied to true predictive gain.

3.2 AGENTS AND ARCHITECTURE

Figure 1 shows the framework. The outer agent selects k columns with the highest potential for generating informative features; the inner agent applies operators to this subset until a stopping condition is met. The new feature is evaluated and appended to the dataset, and the process repeats.

Outer agent. Given $s_O = \phi(D)$, the outer agent selects k original or previously generated columns for the inner agent. We implement it with PPO (Schulman et al., 2017) in TorchRL library (Bou et al., 2024). An action mask (`MaskableDiscrete`) prevents duplicate or invalid selections, so the agent explores only valid subsets.

Inner agent. The inner agent builds a feature with a stack machine. The stack S holds intermediate operands. At each step the agent either pushes a column from F onto S or applies an operator from $\mathcal{O}$. An operator pops its operands (one for unary, two for binary, and the grouping and aggregation columns for higher-order operators) and pushes the result. The constructed feature f_{new} is the top of S when the episode ends, after at most $T_{\max}$ steps. A standard DQN assumes that the current observation is a sufficient statistic for the optimal action, which does not hold here: the value and even the validity of an action depend on what was pushed earlier. We therefore use a DRQN (Hausknecht & Stone, 2017), whose recurrent layer summarizes the construction history. The network has two fully connected ReLU layers of width 256, an LSTM layer of hidden size 256, and a linear output layer (Table 3). A `HybridReplayBuffer` combines a FIFO buffer of recent episodes with

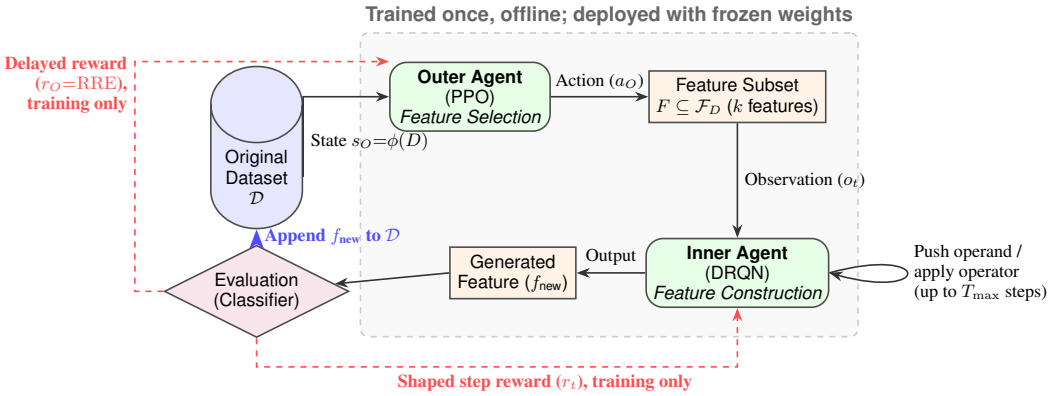

Figure 1: The INFER framework. The outer agent (PPO) selects a subset of k columns from the dataset meta-features $s_O = \phi(D)$. The inner agent (DRQN) composes operators over the subset, for up to $T_{\max}$ steps, to build a new feature f_{new}. During training (once, offline), the evaluation step returns a shaped step reward to the inner agent and the relative error reduction (Eq. 1) as a delayed reward to the outer agent (dashed red arrows). At deployment on a new dataset, the weights are frozen, the dashed arrows are inactive, and the agents generate features by inference alone.

a prioritized buffer (Schaul et al., 2016) of the best episodes by cumulative reward, so successful construction trajectories are retained while exploration continues.

3.3 OPERATORS AND META-FEATURES

The operator library has three families. *Unary* operators $\mathcal{O}_{\text{u}}$ act on one column: discretizers (e.g., equal-width binning) and normalizers (e.g., scaling to $[0, 1]$). *Binary* operators $\mathcal{O}_{\text{b}}$ are the four arithmetic operations $+, -, \times, \div$. *Higher-order* operators $\mathcal{O}_{\text{h}}$ are SQL-like aggregations: *GroupByThenMax*, *Min*, *Avg*, *Stdev*, and *Count*. Every operator with data-dependent parameters (bin edges, scaling ranges, group statistics) is *fitted* on training rows and then *applied* to held-out rows (Section 3.5).

Two families of meta-features describe the state. *Dataset meta-features* $\phi(D) \in \mathbb{R}^{129}$ contain 45 core statistics (size and class balance, cross-validated AUC and log-loss of the current feature set, information-gain summaries, and pairwise dependency tests) plus 84 precision-at-fixed-recall statistics. *Column meta-features* $\psi(\cdot) \in \mathbb{R}^{15}$ describe one column: information gain with the target, aggregate paired-t and χ^2 association scores with other columns, and a discrete/numeric indicator. For a candidate feature, ψ is computed both against the original columns (*in-between* features) and against the current operand set (*parent-based* features); each agent also receives action masks and feature indicators. Appendix D lists every descriptor. Note that several of these statistics depend on the labels; INFER is therefore *target-aware* and assumes labeled training rows, but it needs no textual or semantic metadata.

3.4 TRAINING

Training has two phases and runs once over a collection of training datasets (Algorithm 2 in Appendix B). In phase one, the inner agent learns to build useful features from randomly sampled column subsets of size k; its weights are then frozen. In phase two, the outer agent learns to select subsets, using the frozen inner agent both to build features and to score its selections via RRE. The resulting pair of agents is applied, without retraining, to datasets that were not part of training. During training, AUC and RRE are computed on an internal stratified held-out split of each training dataset.

3.5 DEPLOYMENT ON A NEW DATASET: INFERENCE ONLY

Once trained, INFER never updates its weights again. It generates features for a new dataset through repeated forward inference of the two agents, and the whole procedure is leakage-free by construction. We exploit this by decoupling *generation* from *selection* (Algorithm 1): the agents first produce

Algorithm 1 INFER inference on an unseen dataset (strict nested generation-then-selection)

Require: dataset D; frozen policies π_O, π_I; features per run m; runs N; selection strategy σ

split the rows of D into stratified outer folds
for each outer fold $(D_{\text{train}}, D_{\text{test}})$ **do**
  for $i = 1$ **to** N {generation, outer-train only} **do**
    $D^{(0)} \leftarrow D_{\text{train}}$; $C_i \leftarrow \emptyset$
    for $j = 1$ **to** m **do**
      $s_O \leftarrow \phi(D^{(j-1)})$; select F by π_O; $f_j \leftarrow$ program from F by π_I, fitted on $D^{(j-1)}$
      $C_i \leftarrow C_i \cup \{f_j\}$; $D^{(j)} \leftarrow D^{(j-1)} \cup \{f_j(D^{(j-1)})\}$
    end for
  end for
  $\mathcal{G} \leftarrow \sigma(\{C_1, \ldots, C_N\}, D_{\text{train}})$ {selection, outer-train only (inner-CV OOF)}
  apply the fitted programs in $\mathcal{G}$ to D_{test}; refit the selected classifiers on D_{train}; predict on D_{test}
end for
return metrics aggregated over the outer-test folds

a large pool of candidate features, and a lightweight step then selects the subset that is added to the dataset. Because generation involves no optimization, the number of generation runs N is a natural budget knob, and the selection step can be chosen to fit the available resources.

Candidate generation. One *generation run* produces one *candidate set* C_i of m features: the outer agent selects k columns, the inner agent builds one feature from them, and the feature is appended to the working dataset before the next one is built. Repeating this N times gives a pool $\mathcal{C} = \{C_1, \ldots, C_N\}$. With m=5 and N=50 this is a raw pool of 250 candidate programs. Each candidate is stored as a replayable *fitted program* (operators, operands, and the fitted parameters of every data-dependent operator, including dependencies on earlier generated features) rather than as a materialized column.

Strict nesting. Evaluation is nested. The rows of the target dataset are first split into stratified outer folds. *Everything else happens inside the outer-training rows*: meta-features ϕ and ψ, the policy rollouts, the fitting of discretizer bins, normalizer ranges, and GroupBy aggregates, and the selection step. The fitted programs are then applied unchanged to the outer-test rows. Each outer fold has its own candidate pool and cache namespace, so nothing computed on the full dataset or on another fold can be reused. Appendix A shows the full data flow; Section 4.2 describes how we verify it.

Selection strategies. Each candidate set C_i defines a feature set $\mathcal{F}_D \cup C_i$ and thus a classifier. All strategies operate on the out-of-fold (OOF) probabilities of these classifiers, computed by inner stratified cross-validation on the outer-training rows. **Top-K ensemble** ranks the candidate sets by OOF AUC and averages the probabilities of the top K. The value $K \in \mathcal{K} = \{1, 2, 3, 5, 10\}$ that maximizes the inner-CV AUC of the ensemble is retained for that outer fold. **LR-Stacking** discards candidate sets whose OOF AUC is below that of the original features, then trains an L_2-regularized logistic regression on the OOF probabilities of the remaining sets. If no set passes the filter, all valid sets are retained. **NNLS** applies the same filter and then solves a non-negative least-squares problem for weights that are normalized to sum to one. Sets with zero weight are dropped, which adds a further layer of selection. **Combined** runs the three strategies above and evaluates all $2^3 - 1 = 7$ non-empty subsets of them on *honest* training-side OOF predictions, produced by a second (meta) cross-validation on the outer-training rows. The subset with the highest OOF AUC is selected, and its strategies' predictions are averaged on the outer-test fold; the combination uses training-side information only and is not an oracle. Since all strategies share one candidate pool, running all of them adds little cost over one.

4 EXPERIMENTAL SETUP

4.1 DATASETS

We evaluate on 75 binary-classification datasets from OpenML (Vanschoren et al., 2014). Datasets were retained if they had between 500 and 50,000 instances, so that repeated generation is affordable but evaluation remains reliable; if they were binary, since all reported metrics are ROC AUC; and if

they were compatible with the operator library, which needs at least one numeric column. We prioritized datasets with a screening baseline AUC at or below 0.90, since near-saturated datasets yield few positive rewards and little training signal; a few were kept to probe that regime. The collection is heterogeneous: 500 to $48{,}842$ instances (median $1{,}723$), 4 to 20 columns mixing numeric and categorical attributes, and class balance ranging from even to severely skewed. Appendix G lists all datasets with their sizes.

4.2 Evaluation Protocol

Dataset-level transfer. To measure inductive transfer, we split the 75 datasets into four folds. For each fold, both agents are trained on the other three folds and then applied, with frozen weights, to the held-out fold. Every dataset therefore serves exactly once as an unseen test dataset, and no reported number comes from a dataset the agents were trained on. A separate pair of agents is trained for each downstream classifier, with that classifier used in the reward. In practical terms, every number in this paper is what a user would obtain by downloading the released weights and running them on their own data.

Row-level nesting. On each test dataset we use five stratified outer folds and five inner folds. Generation, fitting, and selection use the outer-training rows only (Section 3.5), and the same outer folds are used for every compared method. We verified the protocol with leakage-invariance tests: perturbing the outer-test rows (labels and predictors, including extreme values) leaves the generated training features, the fitted transformation parameters, and the selection decisions unchanged; perturbing the outer-training labels does change target-aware generation; and each outer fold has an independent candidate pool. All tests pass.

Metrics and statistics. We report mean and median ROC AUC over datasets, the mean RRE over the original features (Eq. 1), and the win-rate, i.e., the percentage of datasets on which a method improves over the original features. Improvements are tested with a two-sided Wilcoxon signed-rank test, Holm-corrected across the seven methods per classifier; average ranks are compared with a Friedman test; confidence intervals for the mean RRE are bootstrapped over datasets ($5{,}000$ resamples).

All agents were trained on Intel Xeon E5 CPUs with a single NVIDIA RTX 3090 GPU. We use $k{=}4$, $T_{\max}{=}10$, $m{=}5$, and $N{=}50$ throughout; Appendix C lists all hyperparameters.

4.3 Baselines and Compared Configurations

We report results with XGBoost and RandomForest as downstream classifiers. Both are strong on tabular data, so improvements require informative features; simpler learners are often improved by shallow transformations, which makes the reward less discriminative.

INFER configurations. All configurations share the same candidate pool and differ only in the selection strategy of Section 3.5: INFER (Top-K), (LR-Stacking), (NNLS), and (Combined).

External baselines. We compare against three methods chosen to span the dominant paradigms and to isolate distinct aspects of our contribution. **OpenFE** (Zhang et al., 2023) is a state-of-the-art feature-generation system; it isolates the value of a learned, transferable generator against a fresh per-dataset search. **DIFER** (Zhu et al., 2022) is the most cited RL-based AutoFE method with public code; it isolates the value of inductive transfer against a per-dataset RL search. **MAFESE** (Van Thieu et al., 2024) is a feature-*selection* library with filter, wrapper, embedded, and metaheuristic strategies; it generates no new features and isolates the value of generation itself. OpenFE and MAFESE were run with their default settings. DIFER was run with the settings of its paper (50 iterations, population 512, 400 controller epochs), a 12-hour cap per fold, and search on the training fold only. All baselines use the same outer folds and downstream classifiers. We also report a case study against the LLM-based method LLM-FE (Abhyankar et al., 2026) on the seven metadata-equipped datasets distributed with its repository (Appendix F); LLM-FE needs textual metadata that most of our benchmark lacks.

Table 1: Summary over the 75-dataset benchmark. "Mean RRE" is the mean relative error reduction over the original features (Eq. 1); "Win" is the percentage of datasets on which the method improves over the original features; "Rank" is the average rank among the eight rows (lower is better; a Friedman test rejects equal performance under both classifiers, $p < 10^{-30}$). *: significantly better than the original features (Wilcoxon signed-rank, Holm-adjusted $p < 10^{-3}$); †: significantly worse (adjusted $p < 10^{-2}$). Best value per column in bold.

Method	Mean AUC	Median AUC	Mean RRE (%)	Win (%)	Rank
XGBoost					
Original features	0.7924	0.8170	–	–	5.11
MAFESE†	0.6920	0.6961	−72.7	6.7	7.56
OpenFE	0.7946	0.8203	−0.3	58.7	4.51
DIFER†	0.7907	0.8185	−2.0	32.0	5.41
INFER (LR-Stacking)	0.7951	0.8258	+2.4	68.0	4.49
INFER (NNLS)*	0.8027	0.8283	+5.5	**84.0**	3.07
INFER (Top-K)*	0.8023	0.8286	+5.2	80.0	3.10
INFER (Combined)*	**0.8031**	**0.8310**	**+5.9**	81.3	**2.75**
RandomForest					
Original features	0.7918	0.8227	–	–	5.69
MAFESE†	0.6840	0.6953	−68.5	6.7	7.65
OpenFE$^{*[1]}$	0.8014	**0.8311**	+5.3	78.7	3.55
DIFER	0.7988	0.8191	+2.8	54.7	4.81
INFER (LR-Stacking)*	0.8057	0.8299	+5.7	82.7	3.36
INFER (NNLS)*	0.8029	0.8297	+4.6	77.3	3.96
INFER (Top-K)*	0.8054	0.8302	+5.8	**84.0**	3.76
INFER (Combined)*	**0.8061**	0.8300	**+6.0**	82.7	**3.21**

5 RESULTS

Table 1 summarizes the benchmark. Figure 2 shows head-to-head win/tie/loss counts of INFER (Combined) against each baseline, and Figure 3 its per-dataset AUC against the original features. Per-dataset numbers are in Appendix H.

INFER improves over the original features. Under XGBoost, INFER (Combined) improves the AUC on 61 of 75 datasets (81%), with a mean RRE of $+5.9\%$ (bootstrap 95% CI $[3.0, 9.7]$) and a mean AUC gain of $+0.011$. Under RandomForest it improves on 62 of 75 (83%), with a mean RRE of $+6.0\%$ (CI $[3.4, 9.3]$) and a mean AUC gain of $+0.014$. Both improvements are highly significant (Wilcoxon $p = 1.0 \times 10^{-5}$ and 3.0×10^{-8}). Top-K and NNLS behave similarly. LR-Stacking is the most conservative variant under XGBoost, where its gain is not significant (68% win, $p = 0.18$), but the strongest single strategy under RandomForest (best on 37 of 75 datasets); we return to this asymmetry in Section 6. The mean RRE is pulled up by one dataset, `Employee-Turnover-at-TECHCO` ($+0.25$ AUC under XGBoost, $+0.33$ under RandomForest); without it the mean RRE is $+4.8\%$ and $+5.1\%$, which is why we report medians alongside means. Overall, INFER (Combined) has the best mean AUC and the best average rank under both classifiers, at negligible cost over the single-strategy variants.

Comparison with the external baselines. MAFESE, a pure selection method, lowers the AUC on 70 of 75 datasets under both classifiers: improvement here requires *adding* signal, not pruning it. DIFER, which runs a per-dataset RL search, is significantly worse than the original features under XGBoost (32% win) and improves only marginally under RandomForest ($+2.8\%$ RRE, $p = 0.05$). INFER (Combined) beats DIFER on 61 of 75 datasets under XGBoost ($p = 1.3 \times 10^{-6}$) and on 50 of 75 under RandomForest ($p = 0.02$). It does so without any optimization on the target dataset, whereas DIFER re-trains its controller on every dataset at a cost of about 27 core-hours (Section 6); the learned generator wins on both quality and cost. OpenFE is the strongest competitor. Under XGBoost it does not significantly improve on the original features ($p = 0.28$), and INFER (Combined)

[1] OpenFE crashed on three RandomForest datasets (`BitcoinHeist_Ransomware`, `Customer_Churn_Classification_Nominal`, and `Employee-Turnover-at-TECHCO`; 24,780–35,000 rows), apparently because of their size. For these datasets we count OpenFE as returning the original features, i.e., its AUC equals the baseline. All other numbers are over all 75 datasets.

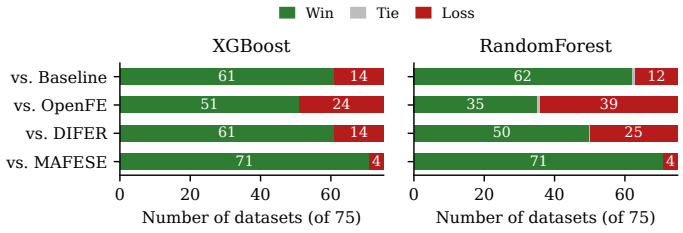

Figure 2: Win/tie/loss of INFER (Combined) against the original features and the three external baselines over the 75 datasets, under XGBoost (left) and RandomForest (right). Under RandomForest, OpenFE's three failed datasets count as ties with the original features and are therefore INFER wins.

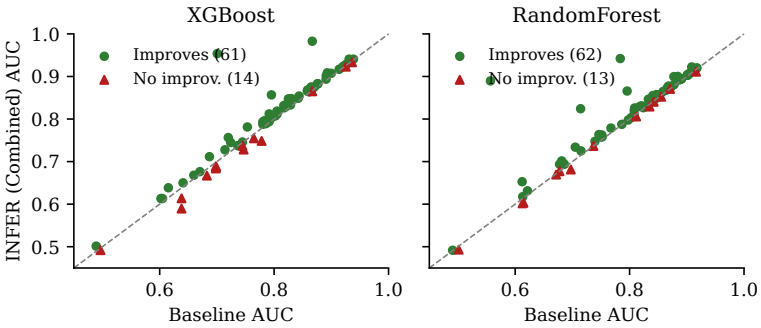

Figure 3: Per-dataset AUC of INFER (Combined) against the original features, under XGBoost (left) and RandomForest (right). Each point is one of the 75 datasets; points above the diagonal are improvements.

beats it on 51 of 75 datasets ($p = 0.01$). Under RandomForest, OpenFE is itself strong ($+5.3\%$ RRE). INFER (Combined) has the higher mean AUC and the better average rank, but the head-to-head comparison is even (35 wins, 1 tie, 39 losses; $p = 0.73$), and OpenFE has the slightly higher median. We read this as parity under RandomForest. The qualitative distinction holds under both classifiers: OpenFE runs a fresh search on every dataset, whereas the INFER agents were trained on other datasets and transfer with frozen weights.

6 Analysis and Discussion

Where the gains come from. Gains are not uniform (Figure 4). Under XGBoost they are weakest on hard datasets (baseline AUC < 0.70: 53% win-rate, mean RRE -0.6%) and large on the medium and easy tiers (89% and 88% win-rate, $+7.4\%$ and $+7.7\%$ RRE): when the original columns carry little signal, the operators have little to amplify. Under RandomForest the gains hold across all tiers (60%, 89%, and 88% win-rate; RRE $+6.3\%$, $+7.2\%$, and $+3.9\%$): constructed features help most when the learner cannot recover the interactions on its own. Gains also grow with dataset size: above the median of 1,723 rows, the win-rate is 89% (XGBoost) and 86% (RandomForest), against 74% and 79% below it.

No single selection strategy dominates. By per-dataset AUC, Top-K is the best single strategy on 37 datasets, NNLS on 24, and LR-Stacking on 14 under XGBoost; under RandomForest the order flips (LR-Stacking 37, Top-K 27, NNLS 11). This heterogeneity motivates the Combined configuration, which is not an oracle: it matches or exceeds all three single strategies on 27 (XGBoost) and 29 (RandomForest) datasets and falls below the per-dataset best on the rest.

Computational cost. INFER has two costs that must be kept apart (Table 2). The first is *one-time and offline*: training the two agents on the source datasets. This took about 1.3 GPU-hours for the inner agent and 614 CPU core-hours for the outer agent, once per downstream classifier. It is paid before any target dataset exists and is never repeated. The second is *per dataset and online*: applying the frozen agents to a new dataset. This step involves no optimization: forward passes of two small

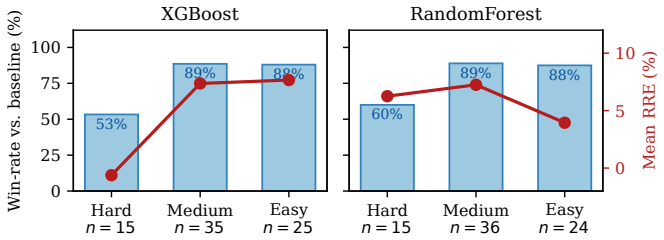

Figure 4: Win-rate over the original features (bars, left axis) and mean RRE (line, right axis) of INFER (Combined), stratified by baseline difficulty (Hard: baseline AUC < 0.70; Medium: 0.70–0.85; Easy: ≥ 0.85), under XGBoost (left) and RandomForest (right).

Table 2: CPU cost of training INFER (once, offline) versus what every method pays per new dataset (core-time, five-fold protocol; deployment is about one fifth). The baselines have no offline part.

Cost	**Core-time**	**What is paid**
One-time, offline (per downstream classifier; never repeated)		
INFER training (both agents)	1.3 GPU-h + ≈ 614 core-h	policy optimization on source datasets
Per new dataset (paid on every dataset)		
INFER generation, N=50	4.0 h (median 3.5 h)	inference + candidate materialization
INFER generation, N=10	≈ 0.8 h	as above, pool of 50 candidates
INFER selection (any N)	≈ 0.2 h	inner-CV out-of-fold selection
OpenFE	0.22 h	full boosting-based search
DIFER	27 h (median 25 h)	full RL search, controller from scratch

networks, candidate materialization, and a lightweight selection step. Under our five-fold protocol it costs about 4.0 core-hours per dataset at N=50 (median 3.5), mostly for candidate materialization. The cost scales linearly with N: at N=10 it is about 48 core-minutes per dataset. The baselines have no offline part: each pays its full search on every dataset. OpenFE is cheap (13 core-minutes per dataset), but its cost is fixed and nothing carries over. DIFER pays about 27 core-hours per dataset and re-trains its controller every time; INFER's training cost is recovered against it after about 27 datasets ($614/(27-4)$), and every dataset after that is about seven times cheaper. This is the practical consequence of the inductive design: the expensive learning happens once, and a new dataset costs only inference.

Limitations. (i) The benchmark covers binary classification with operator-compatible numeric columns; multiclass, regression, and purely categorical settings need their own operators and rewards. (ii) INFER is target-aware: its meta-features use training labels, so it assumes a labeled training set (as all compared methods do), but no textual metadata. (iii) Gains are smallest on hard, low-signal datasets, especially under XGBoost; losses are few and small (Appendix E).

7 CONCLUSION

We presented INFER, a hierarchical two-agent DRL framework for automated feature engineering. An outer agent selects promising columns from dataset meta-features, and a recurrent inner agent composes operators over them to build new features. Two properties distinguish the design. First, the agents are trained once, offline, and applied to unseen datasets with frozen weights, so feature engineering on a new dataset becomes policy inference with a tunable budget rather than a fresh search. Second, the evaluation is leakage-free by construction: every candidate is generated, fitted, and selected inside the training rows of each outer fold. Under this strict protocol on 75 datasets, INFER improves over the original features on more than 80% of datasets with both XGBoost and RandomForest, outperforms the per-dataset RL method DIFER on both quality and cost, and matches or exceeds OpenFE, the strongest per-dataset generator. The expensive part of the method is paid once; every dataset after that is cheap, and the released weights can be applied to new data as they are. Future work includes measuring the accuracy–cost curve over the generation budget N, extending the operator library and reward to multiclass and regression tasks, and removing the caching bottleneck that limits parallel training.

REPRODUCIBILITY STATEMENT

Section 3 and Algorithm 1 specify the method, and Appendix B gives the training algorithm. Appendix C lists all hyperparameters, Appendix D enumerates every meta-feature, Appendix G lists the datasets, and Appendix H gives per-dataset results for every method. The evaluation protocol, including the leakage-invariance tests, is described in Section 4.2 and Appendix A. Code, trained weights, fold assignments, and the baseline drivers are available at `https://github.com/infer2026/INFER`.

AI USE STATEMENT

Large language model (LLM) tools were used solely to assist with the writing of this manuscript, specifically for language polishing, including improving grammar, clarity, phrasing, and overall readability. These tools were not used to generate research ideas, design or conduct experiments, analyze results, or produce scientific content. All AI-assisted edits were reviewed and verified by the authors, who take full responsibility for the content of this paper.

REFERENCES

Nikhil Abhyankar, Parshin Shojaee, and Chandan K. Reddy. Llm-fe: Automated feature engineering for tabular data with llms as evolutionary optimizers, 2026. URL https://arxiv.org/abs/2503.14434.

Albert Bou, Matteo Bettini, Sebastian Dittert, Vikash Kumar, Shagun Sodhani, Xiaomeng Yang, Gianni De Fabritiis, and Vincent Moens. Torchrl: A data-driven decision-making library for pytorch. In *International conference on learning representations*, volume 2024, pp. 1778–1811, 2024.

Xiangning Chen, Bo Qiao, Weiyi Zhang, Wei Wu, Murali Chintalapati, Dongmei Zhang, Qingwei Lin, Chuan Luo, Xudong Li, Hongyu Zhang, Yong Xu, Yingnong Dang, Kaixin Sui, and Xu Zhang. Neural feature search: A neural architecture for automated feature engineering. In *2019 IEEE International Conference on Data Mining (ICDM)*, pp. 71–80. IEEE, 2019.

Gabriel Dulac-Arnold, Richard Evans, Hado van Hasselt, Peter Sunehag, Timothy Lillicrap, Jonathan Hunt, Timothy Mann, Theophane Weber, Thomas Degris, and Ben Coppin. Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679*, 2015.

Wanfu Gao, Zengyao Man, Hanlin Pan, and Kunpeng Liu. Dual-agent reinforcement learning for automated feature generation, 2025. URL https://arxiv.org/abs/2505.12628.

Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps, 2017. URL https://arxiv.org/abs/1507.06527.

Noah Hollmann, Samuel Müller, and Frank Hutter. Large language models for automated data science: Introducing caafe for context-aware automated feature engineering, 2023. URL https://arxiv.org/abs/2305.03403.

Franziska Horn, Robert Pack, and Michael Rieger. The autofeat python library for automated feature engineering and selection. In Peggy Cellier and Kurt Driessens (eds.), *Machine Learning and Knowledge Discovery in Databases*, pp. 111–120, Cham, 2020. Springer International Publishing. ISBN 978-3-030-43823-4.

Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. Meta-learning in neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(9): 5149–5169, 2022.

James Max Kanter and Kalyan Veeramachaneni. Deep feature synthesis: Towards automating data science endeavors. In *2015 IEEE international conference on data science and advanced analytics (DSAA)*, pp. 1–10. IEEE, 2015.

Gilad Katz, Eui Chul Richard Shin, and Dawn Song. Explorekit: Automatic feature generation and selection. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pp. 979–984. IEEE, 2016.

Udayan Khurana, Deepak Turaga, Horst Samulowitz, and Srinivasan Parthasrathy. Cognito: Automated feature engineering for supervised learning. In *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*, pp. 1304–1307. IEEE, 2016.

Udayan Khurana, Horst Samulowitz, and Deepak Turaga. Feature engineering for predictive modeling using reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Zhuoyan Li, Aditya Bansal, Jinzhao Li, Shishuang He, Zhuoran Lu, Mutian Zhang, Qin Liu, Yiwei Yang, Swati Jain, Ming Yin, and Yunyao Li. Human-llm collaborative feature engineering for tabular data, 2026. URL https://arxiv.org/abs/2601.21060.

Kunpeng Liu, Yanjie Fu, Pengfei Wang, Le Wu, Rui Bo, and Xiaolin Li. Automating feature subspace exploration via multi-agent reinforcement learning. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 207–215. ACM, 2019. doi: 10.1145/3292500.3330868.

Fatemeh Nargesian, Horst Samulowitz, Udayan Khurana, Elias B Khalil, and Deepak S Turaga. Learning feature engineering for classification. In *Ijcai*, pp. 2529–2535, 2017.

Ivan Reyes-Amezcua, Daniel Flores-Araiza, Gilberto Ochoa-Ruiz, Andres Mendez-Vazquez, and Eduardo Rodriguez-Tello. Macfe: A meta-learning and causality based feature engineering framework, 2022. URL https://arxiv.org/abs/2207.04010.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay, 2016. URL https://arxiv.org/abs/1511.05952.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

Bastian Schäfer, Lennart Purucker, Maciej Janowski, and Frank Hutter. How usable is automated feature engineering for tabular data?, 2025. URL https://arxiv.org/abs/2508.13932.

Nguyen Van Thieu, Ngoc Hung Nguyen, and Ali Asghar Heidari. Feature selection using metaheuristics made easy: Open source mafese library in python. *Future Generation Computer Systems*, 160:340–358, 2024. ISSN 0167-739X. doi: https://doi.org/10.1016/j.future.2024.06.006. URL https://www.sciencedirect.com/science/article/pii/S0167739X24003030.

Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. Openml: networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 15(2):49–60, June 2014. ISSN 1931-0153. doi: 10.1145/2641190.2641198. URL http://dx.doi.org/10.1145/2641190.2641198.

Meng Xiao, Junfeng Zhou, and Yuanchun Zhou. Reinforcement learning-based feature generation algorithm for scientific data, 2025. URL https://arxiv.org/abs/2507.03498.

Wangyang Ying, Dongjie Wang, Xuanming Hu, Yuanchun Zhou, Charu C. Aggarwal, and Yanjie Fu. Unsupervised generative feature transformation via graph contrastive pre-training and multi-objective fine-tuning, 2024. URL https://arxiv.org/abs/2405.16879.

Wangyang Ying, Cong Wei, Nanxu Gong, Xinyuan Wang, Haoyue Bai, Arun Vignesh Malarkkan, Sixun Dong, Dongjie Wang, Denghui Zhang, and Yanjie Fu. A survey on data-centric ai: Tabular learning from reinforcement learning and generative ai perspective, 2025. URL https://arxiv.org/abs/2502.08828.

Tianping Zhang, Zheyu Aqa Zhang, Zhiyuan Fan, Haoyan Luo, Fengyuan Liu, Qian Liu, Wei Cao, and Li Jian. OpenFE: Automated feature generation with expert-level performance. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 41880–41901. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/zhang23ay.html.

Guanghui Zhu, Zhuoer Xu, Chunfeng Yuan, and Yihua Huang. Difer: Differentiable automated feature engineering. In *Proceedings of the First International Conference on Automated Machine Learning*, volume 188 of *Proceedings of Machine Learning Research*, pp. 17/1–17. PMLR, 2022. URL https://proceedings.mlr.press/v188/zhu22a.html.

A Evaluation and Selection Flow

Figure 5 shows the complete data flow for one outer fold. The left box contains everything that happens on the outer-training rows: candidate generation and operator fitting, the inner cross-validation that produces out-of-fold probabilities per candidate set, the three selection strategies, and the meta cross-validation that produces honest out-of-fold predictions per strategy for the Combined configuration. The right box contains the only steps that touch the outer-test rows: applying the fitted programs, predicting with the refit classifiers, and computing the final metric.

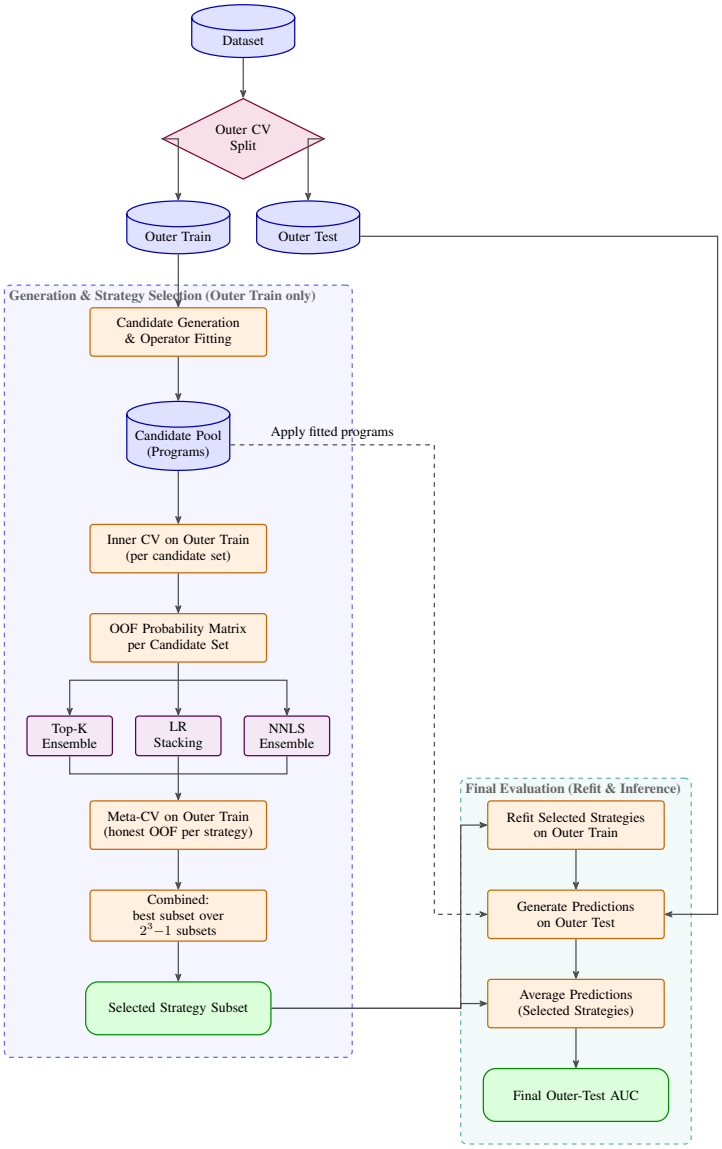

Figure 5: Evaluation and selection flow for one outer fold under nested cross-validation. Generation, operator fitting, and selection use the outer-training rows only (left box). The outer-test rows enter only at prediction time (right box).

B TRAINING ALGORITHM

Algorithm 2 INFER training (two phases)

Require: training datasets $\mathcal{D}_{\text{tr}}$; operators $\mathcal{O}$; subset size k; features per episode m; horizon $T_{\max}$; episode budgets E_I, E_O
Ensure: trained policies π_I, π_O

initialize π_I (DRQN) and π_O (PPO) with random weights
Phase 1 – train the inner agent
for $e = 1$ **to** E_I **do**
  sample $D \sim \mathcal{D}_{\text{tr}}$ and a subset $F \subseteq \mathcal{F}_D$ with $|F| = k$
  $S \leftarrow \emptyset$; $h_0 \leftarrow \mathbf{0}$; $t \leftarrow 0$
  repeat
    $t \leftarrow t + 1$; observe o_t from (S, F, ψ); $a_t \sim \pi_I(\cdot \mid o_t, h_{t-1})$
    $S \leftarrow \text{APPLY}(S, a_t)$ {push operand or apply operator}
    receive shaped step reward r_t {Section 3.1}
  until a_t terminal **or** $t = T_{\max}$
  store transitions in the hybrid replay buffer; update π_I
end for
freeze π_I
Phase 2 – train the outer agent
for $e = 1$ **to** E_O **do**
  sample $D \sim \mathcal{D}_{\text{tr}}$; $D^{(0)} \leftarrow D$; $s_O \leftarrow \phi(D^{(0)})$
  for $j = 1$ **to** m **do**
    $F \leftarrow \emptyset$
    while $|F| < k$ **do**
      $a_O \sim \pi_O(\cdot \mid s_O)$ with action masking; $F \leftarrow F \cup \{a_O\}$
    end while
    $f_j \leftarrow$ feature built from F by the frozen π_I
    $r_O \leftarrow \text{RRE}\big(\mathcal{F}_{D^{(j-1)}}, \mathcal{F}_{D^{(j-1)}} \cup \{f_j\}\big)$
    $D^{(j)} \leftarrow D^{(j-1)} \cup \{f_j\}$; $s_O \leftarrow \phi(D^{(j)})$
  end for
  update π_O (PPO)
end for
return π_I, π_O

C HYPERPARAMETERS

Table 3: Hyperparameters of the INFER agents and of the inference and selection step.

Parameter	Inner agent (DRQN)	Outer agent (PPO)
Subset size k	4	
Max. steps $T_{\max}$	10	—
Episodes E	150,000	20,000
Optimizer	Adam	Adam
Learning rate	10^{-3}	$3{\times}10^{-4}$
Batch / rollout size	64	64
Replay buffer	5,000 episodes (hybrid)	rollout (n_{steps}=64)
Replay warm-up	256 episodes	—
Discount γ	0.99	0.99
Soft update τ	10^{-2} (every 4 steps)	—
Exploration	ϵ: $0.9 \rightarrow 0.005$	stochastic (PPO)
Network	$256 \rightarrow 256 \rightarrow$ LSTM(256)	MLP $[256, 256]$
Inference and selection: m=5 features per run; N=50 runs; 5 outer × 5 inner folds; $\mathcal{K} = \{1, 2, 3, 5, 10\}$		

D META-FEATURE ENUMERATION

Dataset meta-features $\phi(D)$ (129 total). Table 4 groups the 45 core descriptors. The remaining 84 are precision-at-fixed-recall statistics: for each recall level $r \in \{0.00, 0.05, \ldots, 1.00\}$, the maximum, minimum, mean, and standard deviation of precision at recall r over the cross-validation folds.

Table 4: Core dataset meta-features (45 descriptors).

Group	Descriptors
Size and balance (8)	number of instances, classes, features, numeric attributes, and discrete attributes; ratio of numeric and of discrete attributes; minority-class percentage
Discrete-value statistics (4)	max, min, mean, stdev of the number of distinct values per discrete attribute
Classification (9)	number of folds; max, min, mean, stdev of AUC; max, min, mean, stdev of log-loss
Information gain (12)	max, min, mean, stdev of information gain over all, discrete, and numeric attributes
Feature dependencies (12)	max, min, mean, stdev of paired t-test values (numeric attributes), of χ^2 values (discrete attributes), and of χ^2 values (discrete and discretized attributes)

Column meta-features $\psi(\cdot)$ (15 per column). For each column, the 15 descriptors are listed in Table 5. One slot ("probability difference score") is reserved in the implementation and currently holds a constant.

Table 5: In-between and parent-based column meta-features (15 descriptors per column).

Group	Descriptors
Target association (1)	information gain with the target
Placeholder (1)	probability difference score (constant)
Numeric associations (4)	max, min, mean, stdev of paired t-test values between the column and the numeric attributes
Discrete χ^2 associations (4)	max, min, mean, stdev of χ^2 values between the column and the discrete attributes
All-column χ^2 associations (4)	max, min, mean, stdev of χ^2 values between the column and all attributes
Type indicator (1)	is the column discrete

E COST DETAILS AND FAILURE CASES

Table 2 reports the measured CPU cost of every method under the common five-fold protocol. Each (dataset, outer fold) unit runs on one core (classifier `n_jobs=1`, single-threaded BLAS), so wall-clock time equals core-time per unit and units are independent. With C cores, the wall-clock time of the whole benchmark is approximately $\max(75 \cdot 5 \cdot t_{\text{unit}}/C,\ U)$, where t_{unit} is the mean unit time and U the longest single unit (3.0 hours for INFER at N=50). Evaluation needs five units per dataset; deployment on a new dataset needs one, since the features are generated once on the full training data. DIFER hit our 12-hour-per-fold cap on 53 of 75 datasets, so its paper setting would cost more than the 27 core-hours we report. One-time training of a pair of agents took 1.3 GPU-hours for the inner DRQN (1.3 hours wall-clock on one GPU with three workers) and about 614 CPU core-hours for the outer PPO (30.7 hours wall-clock on 20 CPUs with 16 workers); the two phases run sequentially.

Failure cases. Losses are few and small: at most -0.016 AUC under RandomForest and -0.049 under XGBoost, mostly on low-signal problems of about 500 rows. The largest gains (`Employee-Turnover-at-TECHCO`, `monks-2`, `PieChart2`) arise where arithmetic and group-by aggregations expose interactions the learner cannot recover on its own.

F Case Study: LLM-Based Feature Engineering

LLM-FE (Abhyankar et al., 2026) formulates feature engineering as LLM-guided evolutionary program search, with prompts conditioned on dataset metadata, examples, and prior high-scoring programs. It needs textual dataset and column descriptions, which most of our benchmark lacks. We therefore compare on the seven metadata-equipped binary-classification datasets distributed with the LLM-FE repository (Table 6). Several have a very high baseline AUC (e.g., `breast-w`, ≈ 0.99), which leaves little headroom for any method. We present this as a case study, not a benchmark.

Table 6: Comparison with LLM-FE on the seven metadata-equipped datasets. Best value per row in bold. "Win vs. base" is the fraction of datasets on which each method improves over the original features.

Dataset	Original	INFER (Top-K)	INFER (Comb.)	LLM-FE
XGBoost				
adult	0.9279	**0.9299**	0.9298	0.9116
bank	0.9307	**0.9345**	0.9341	0.9289
blood	**0.6981**	0.6886	0.6869	0.6683
breast-w	0.9940	**0.9948**	0.9939	0.9917
credit-g	**0.7803**	0.7710	0.7725	0.7683
heart	0.9195	0.9228	**0.9234**	0.9220
pc1	0.8115	0.8348	**0.8383**	0.7542
Mean	0.8660	0.8681	**0.8684**	0.8493
Win vs. base	–	71%	57%	14%
RandomForest				
adult	0.9023	0.9074	**0.9079**	0.9037
bank	0.9290	0.9302	**0.9305**	0.8022
blood	0.6861	0.6854	**0.6882**	0.6328
breast-w	0.9921	0.9921	0.9921	**0.9934**
credit-g	**0.7864**	0.7850	0.7857	0.7855
heart	0.9262	0.9211	0.9213	**0.9276**
pc1	0.8296	0.8330	**0.8360**	0.8107
Mean	0.8645	0.8649	**0.8660**	0.8366
Win vs. base	–	43%	57%	43%

Two observations follow. First, INFER has the higher mean AUC under both classifiers (XGBoost: 0.868 vs. 0.849; RandomForest: 0.866 vs. 0.837). Head-to-head, it beats LLM-FE on all seven XGBoost datasets and, in the Combined configuration, on five of seven RandomForest datasets. Second, LLM-FE often *degrades* performance: it improves over the original features on one of seven (XGBoost) and three of seven (RandomForest) datasets, with one severe regression on `bank` under RandomForest (0.802 vs. a 0.929 baseline). LLM-FE also has a different cost profile: about eight minutes per dataset on our local setup, or roughly 100K–150K API tokens ($\approx$ \$0.05) with a hosted model; the original paper used four NVIDIA RTX 8000 GPUs. On these datasets, a lightweight generator that needs no metadata was both more reliable and cheaper.

G Datasets

Table 7 lists all 75 datasets with their number of rows and columns.

H Per-Dataset Results

Tables 7 and 8 report the AUC of every method on every dataset under XGBoost and RandomForest. The best value per row is in bold.

Table 7: Per-dataset AUC under XGBoost. Orig.: original features; MAF.: MAFESE; LR: LR-Stacking; Comb.: Combined. Best value per row in bold.

Dataset	Rows	Cols	Orig.	MAF.	OpenFE	DIFER	Top-K	LR	NNLS	Comb.
abalone	4177	8	0.867	0.847	0.872	0.864	0.873	0.872	**0.873**	0.873
ada_prior	4474	14	0.891	0.796	0.894	0.888	**0.894**	0.893	0.893	0.894
adult	45222	14	0.926	0.876	0.925	0.923	**0.928**	0.928	0.928	**0.928**
adult-census	30162	14	0.924	0.849	0.920	0.923	**0.927**	0.926	0.926	**0.927**
adult-test	48842	11	0.922	0.785	0.920	0.919	**0.923**	0.923	0.923	**0.923**
arsenic-female-bladder	559	4	0.805	0.690	**0.833**	0.766	0.812	0.805	0.812	0.812
arsenic-male-bladder	559	4	0.830	**0.889**	0.847	0.822	0.814	0.850	0.849	0.848
bank32nh	8192	20	0.876	0.780	0.878	0.870	**0.884**	0.882	0.883	0.883
Bank_Customer_Churn	10000	10	0.844	0.807	0.844	0.843	**0.854**	0.853	0.854	0.854
bank_R	4521	16	0.900	0.614	**0.909**	0.901	0.908	0.905	0.906	0.907
BitcoinHeist_Ransomware	24780	7	0.939	0.838	0.817	0.931	**0.941**	0.941	0.941	**0.941**
blastchar	7032	19	0.822	0.753	0.820	0.821	0.834	0.833	0.834	**0.834**
blood-transfusion-service-center	748	4	0.698	0.649	0.686	**0.701**	0.688	0.679	0.687	0.684
blood_R	748	4	**0.698**	0.613	0.678	0.660	0.684	0.684	0.695	0.689
bwin_amlb	530	13	0.602	0.552	0.603	0.608	0.612	0.581	0.608	**0.613**
churn	5000	19	0.925	0.795	**0.931**	0.922	0.923	0.926	0.923	0.923
Click_prediction_small	39948	5	0.660	0.550	0.662	0.655	0.667	**0.668**	0.668	0.668
click_prediction_small	39926	4	0.641	0.549	0.647	0.600	**0.651**	0.650	0.650	0.650
climate-model-simulation-crashes	540	20	0.936	0.535	0.937	0.909	0.937	**0.940**	0.936	0.933
compass	16644	17	0.830	0.744	**0.870**	0.832	0.847	0.847	0.847	0.847
contraceptive	1473	9	0.687	0.696	0.695	0.689	0.711	0.688	0.708	**0.712**
credit	16714	10	0.842	0.697	0.834	0.840	0.849	**0.849**	**0.849**	**0.849**
Customer_Churn_Classification_Nominal	35000	20	0.876	0.831	0.873	0.871	0.883	0.883	0.883	**0.883**
dataset_31_credit-g	1000	20	0.780	0.711	0.775	0.782	0.787	0.774	0.786	**0.788**
dataset_credit	16714	10	0.842	0.767	0.834	0.840	0.849	0.849	**0.850**	**0.850**
diabetes	768	8	0.791	0.741	0.797	0.785	0.792	0.793	**0.797**	0.794
doa_bwin	530	13	0.605	0.554	0.602	0.596	0.613	0.574	**0.613**	0.613
doa_bwin_balanced	708	13	0.783	0.690	**0.805**	0.793	0.799	0.784	0.796	0.796
E-CommereShippingData	10999	10	0.736	0.723	0.734	0.734	0.737	**0.739**	0.739	0.738
Employee	4653	8	**0.867**	0.775	0.852	0.866	0.866	0.863	0.864	0.865
Employee-Turnover-at-TECHCO	34452	10	0.701	0.536	**0.993**	0.685	0.954	0.892	0.953	0.954
eucalyptus	641	19	0.829	0.711	0.825	0.818	**0.835**	0.830	0.832	0.833
eye_movements	7608	20	0.720	0.730	0.694	0.699	0.755	0.755	0.755	**0.756**
FICO-HELOC-cleaned	9871	20	0.780	0.764	0.781	0.783	**0.793**	0.792	0.792	0.793
Fitness_Club	1480	6	0.753	**0.804**	0.765	0.754	0.779	0.769	0.779	0.781
Fraud-Detection-Updated	530	20	**0.638**	0.542	0.619	0.633	0.626	0.588	0.608	0.614
Fraud-Detection-Updated	530	20	**0.638**	0.515	0.619	0.633	0.617	0.571	0.584	0.590
German-Credit-Risk-with-Target	522	9	**0.699**	0.582	0.664	0.665	0.688	0.662	0.688	0.688
Give-Me-Some-Credit-Sampled	36028	10	0.821	0.648	0.814	0.819	0.831	0.826	0.826	**0.831**
ilpd	583	10	0.745	0.620	**0.757**	0.726	0.739	0.712	0.741	0.737
ilpd-indian-liver_R	583	9	0.747	0.637	**0.760**	0.733	0.739	0.699	0.735	0.728
ilpd-numeric	583	10	0.745	0.597	**0.762**	0.726	0.744	0.729	0.747	0.746
imdb_genre_prediction	666	7	0.802	0.694	0.799	0.790	0.802	0.804	**0.808**	0.807
in_vehicle_coupon_recommendation	12684	20	0.822	0.679	0.825	0.817	0.829	0.830	**0.830**	**0.830**
Is-this-a-good-customer	1723	13	0.670	0.600	**0.679**	0.672	0.673	0.668	0.678	0.676
jm1	10880	20	0.724	0.679	0.521	0.724	**0.745**	0.742	0.744	**0.745**
kc1	2109	20	0.786	0.642	**0.812**	0.763	0.790	0.758	0.790	0.789
kc2	522	20	**0.765**	0.738	0.754	0.751	0.758	0.729	0.752	0.755
Mammographic-Mass-Data-Set	830	5	0.864	0.835	0.851	**0.881**	0.867	0.865	0.867	0.867
mammographic_R	961	5	0.866	0.813	0.868	**0.877**	0.870	0.864	0.868	0.869
Marketing_Campaign	2216	20	0.858	0.657	0.862	0.858	**0.867**	0.860	0.867	0.866
monks-2	601	6	0.867	0.520	0.939	0.860	0.964	**0.983**	0.965	**0.983**
NewspaperChurn	15855	13	0.859	0.630	0.859	0.861	**0.868**	0.865	0.865	**0.868**
oil_spill	937	20	0.931	0.648	0.931	0.936	0.935	0.937	0.939	**0.941**
pbcseq	1113	17	0.893	0.822	0.884	0.879	0.908	0.906	**0.909**	0.909
pc1	1109	20	0.817	0.700	**0.844**	0.833	0.830	0.834	0.828	0.832
pc2	5589	20	0.778	0.768	**0.800**	0.793	0.745	0.789	0.779	0.749
pc3	1563	20	0.818	0.722	**0.841**	0.821	0.829	0.753	0.826	0.826
php7KLval	1000	20	**0.682**	0.565	0.665	0.675	0.668	0.670	0.669	0.667
PieChart1	705	20	0.865	0.653	0.868	0.844	0.873	0.848	0.870	**0.875**
PieChart2	745	20	0.795	0.643	0.798	0.747	0.827	**0.862**	0.837	0.857
PieChart3	1077	20	0.805	0.643	0.827	**0.830**	0.821	0.786	0.816	0.819
pima_R	768	8	0.791	0.727	0.797	0.782	0.805	0.812	**0.815**	0.812
PizzaCutter1	661	20	0.861	0.705	0.849	0.858	0.863	**0.867**	0.867	0.864
PizzaCutter3	1043	20	0.825	0.683	0.834	0.839	0.840	0.812	0.845	**0.848**
pm10	500	7	0.615	0.560	0.590	**0.872**	0.636	0.638	0.642	0.638
pollen	3848	5	0.489	0.485	0.490	0.494	**0.503**	**0.503**	0.492	0.501
qsar	1055	20	0.914	0.870	**0.921**	0.900	0.916	0.915	0.916	0.917
rl	4970	12	0.871	0.757	0.890	**0.891**	0.876	0.874	0.876	0.876
seismic-bumps	2584	18	0.714	0.625	0.722	**0.729**	0.729	0.712	0.722	0.728
South_Asian_Churn_dataset	2000	9	0.796	0.727	0.795	0.794	0.804	0.800	**0.804**	0.804
UCI_churn	3333	19	0.920	0.803	0.918	0.918	0.921	0.922	0.922	**0.923**
wine	2554	11	0.891	0.783	0.900	0.885	0.901	0.896	0.898	**0.902**
WMO-Hurricane-Survival-Dataset	4951	20	0.497	**0.506**	0.504	0.482	0.498	0.491	0.497	0.492

Table 8: Per-dataset AUC under RandomForest. "–" marks the three datasets on which OpenFE failed. Abbreviations as in Table 7.

Dataset	Rows	Cols	Orig.	MAF.	OpenFE	DIFER	Top-K	LR	NNLS	Comb.
abalone	4177	8	0.872	0.831	**0.876**	0.872	0.875	0.875	0.875	0.875
ada_prior	4474	14	0.887	0.792	0.891	0.889	**0.892**	0.891	0.891	0.892
adult	45222	14	0.902	0.875	0.911	**0.914**	0.904	0.903	0.904	0.904
adult-census	30162	14	0.901	0.844	**0.909**	0.908	0.902	0.902	0.903	0.903
adult-test	48842	11	0.890	0.773	0.900	**0.908**	0.894	0.893	0.893	0.894
arsenic-female-bladder	559	4	0.825	0.677	**0.831**	0.791	0.827	0.826	0.828	0.828
arsenic-male-bladder	559	4	0.841	0.841	**0.881**	0.875	0.880	0.846	0.848	0.855
bank32nh	8192	20	0.878	0.785	**0.881**	0.876	0.881	0.880	0.880	**0.881**
Bank_Customer_Churn	10000	10	0.850	0.803	0.851	0.851	0.856	0.856	**0.857**	0.856
bank_R	4521	16	0.903	0.609	**0.919**	0.898	0.903	0.902	0.905	0.904
BitcoinHeist_Ransomware	24780	7	0.901	0.825	–	**0.913**	0.901	0.904	0.901	0.904
blastchar	7032	19	0.818	0.751	0.827	0.818	0.829	0.829	**0.830**	**0.830**
blood-transfusion-service-center	748	4	0.686	0.643	0.708	**0.718**	0.695	0.692	0.696	0.694
blood_R	748	4	0.684	0.600	0.705	**0.711**	0.696	0.696	0.696	0.696
bwin_amlb	530	13	0.615	0.533	0.596	**0.619**	0.613	0.602	0.604	0.604
churn	5000	19	0.917	0.796	**0.920**	0.918	0.920	0.920	0.919	0.919
Click_prediction_small	39948	5	0.622	0.550	0.627	**0.666**	0.631	0.631	0.631	0.631
click_prediction_small	39926	4	0.613	0.553	**0.619**	0.605	0.618	0.617	0.618	0.618
climate-model-simulation-crashes	540	20	0.891	0.519	**0.931**	0.896	0.911	0.882	0.882	0.895
compass	16644	17	0.884	0.784	0.859	0.889	0.897	**0.900**	0.897	**0.900**
contraceptive	1473	9	0.681	0.695	0.697	0.680	**0.702**	0.698	0.698	0.702
credit	16714	10	0.845	0.697	0.846	0.844	0.848	0.848	**0.848**	**0.848**
Customer_Churn_Classification_Nominal	35000	20	0.868	0.829	–	0.874	0.877	0.877	0.877	**0.877**
dataset_31_credit-g	1000	20	0.786	0.708	0.788	0.786	0.789	**0.790**	0.788	0.787
dataset_credit	16714	10	0.845	0.753	0.846	0.844	0.848	**0.848**	**0.848**	**0.848**
diabetes	768	8	0.824	0.741	0.825	0.829	0.828	0.829	**0.830**	0.829
doa_bwin	530	13	0.613	0.546	**0.624**	0.612	0.604	0.605	0.605	0.602
doa_bwin_balanced	708	13	0.808	0.721	**0.848**	0.816	0.821	0.820	0.817	0.821
E-CommereShippingData	10999	10	0.737	0.720	0.738	**0.739**	0.737	0.738	0.738	0.737
Employee	4653	8	0.851	0.772	0.850	**0.862**	0.851	0.852	0.850	0.852
Employee-Turnover-at-TECHCO	34452	10	0.557	0.540	–	0.613	0.867	0.889	0.863	**0.890**
eucalyptus	641	19	0.842	0.719	**0.845**	0.835	0.839	0.840	0.840	0.841
eye_movements	7608	20	0.705	**0.814**	0.708	0.714	0.731	0.735	0.732	0.734
FICO-HELOC-cleaned	9871	20	0.798	0.753	0.795	0.796	**0.799**	0.797	0.797	0.798
Fitness_Club	1480	6	0.767	**0.800**	0.779	0.771	0.780	0.779	0.779	0.779
Fraud-Detection-Updated	530	20	0.678	0.548	0.668	0.657	**0.700**	0.678	0.688	0.694
Fraud-Detection-Updated	530	20	**0.678**	0.512	0.668	0.657	0.676	0.672	0.669	0.677
German-Credit-Risk-with-Target	522	9	0.672	0.563	**0.682**	0.678	0.671	0.672	0.672	0.669
Give-Me-Some-Credit-Sampled	36028	10	0.820	0.648	0.825	0.819	**0.830**	0.827	0.828	0.830
Give-Me-Some-Credit-Sampled-Preprocessed	36028	10	0.820	0.624	0.825	0.819	0.830	0.830	0.830	**0.831**
ilpd	583	10	0.751	0.616	**0.772**	0.755	0.761	0.764	0.761	0.763
ilpd-indian-liver_R	583	9	0.752	0.650	**0.782**	0.750	0.755	0.761	0.756	0.758
ilpd-numeric	583	10	0.746	0.596	0.764	0.754	0.762	**0.765**	0.761	0.763
imdb_genre_prediction	666	7	0.809	0.698	0.817	0.819	0.824	0.826	0.825	**0.826**
in_vehicle_coupon_recommendation	12684	20	0.808	0.677	**0.816**	0.810	0.811	0.810	0.810	0.811
Is-this-a-good-customer	1723	13	0.715	0.606	0.689	0.714	0.726	**0.730**	0.728	0.725
jm1	10880	20	0.751	0.652	0.717	0.748	0.758	0.757	0.758	**0.758**
kc1	2109	20	0.824	0.640	**0.838**	0.823	0.826	0.828	0.826	0.827
kc2	522	20	0.811	0.752	**0.815**	0.802	0.814	0.809	0.807	0.806
Mammographic-Mass-Data-Set	830	5	0.859	0.831	0.862	**0.874**	0.863	0.867	0.866	0.865
mammographic_R	961	5	0.864	0.813	0.865	**0.877**	0.866	0.867	0.869	0.868
Marketing_Campaign	2216	20	0.871	0.650	0.863	0.859	0.870	**0.872**	0.870	0.871
monks-2	601	6	0.784	0.517	0.940	0.791	0.909	**0.942**	0.905	**0.942**
NewspaperChurn	15855	13	0.853	0.630	**0.865**	0.854	0.857	0.856	0.857	0.858
oil_spill	937	20	0.909	0.581	**0.938**	0.908	0.931	0.919	0.922	0.922
pbcseq	1113	17	0.877	0.844	0.861	0.867	0.900	**0.901**	0.900	0.900
pc1	1109	20	0.855	0.700	**0.867**	0.852	0.859	0.854	0.853	0.852
pc2	5589	20	0.714	0.389	0.813	0.676	0.809	0.816	0.732	**0.824**
pc3	1563	20	0.844	0.717	0.853	0.837	**0.853**	0.851	0.851	0.851
php7KLval	1000	20	0.697	0.560	**0.710**	0.686	0.683	0.682	0.679	0.681
PieChart1	705	20	0.866	0.646	0.883	0.862	0.872	**0.884**	0.879	0.872
PieChart2	745	20	0.796	0.572	0.779	0.814	0.847	0.843	0.847	**0.866**
PieChart3	1077	20	0.832	0.627	**0.841**	0.836	0.834	0.838	0.833	0.834
pima_R	768	8	0.823	0.733	0.824	0.814	0.829	**0.831**	0.830	0.830
PizzaCutter1	661	20	0.847	0.687	0.865	**0.866**	0.854	0.864	0.858	0.857
PizzaCutter3	1043	20	0.833	0.690	0.842	0.831	0.843	0.846	0.844	**0.847**
pm10	500	7	0.612	0.535	0.618	**0.865**	0.648	0.650	0.653	0.653
pollen	3848	5	0.491	0.494	**0.499**	0.487	0.484	0.492	0.487	0.492
qsar	1055	20	0.918	0.880	**0.921**	0.911	0.920	0.920	0.920	0.920
rl	4970	12	0.835	0.767	0.890	**0.915**	0.812	0.829	0.812	0.829
seismic-bumps	2584	18	0.740	0.618	0.734	0.743	0.750	**0.751**	0.748	0.748
South_Asian_Churn_dataset	2000	9	0.803	0.729	**0.810**	0.804	0.805	0.801	0.801	0.803
UCI_churn	3333	19	0.916	0.809	0.917	**0.918**	0.911	0.914	0.913	0.911

(continued)

Dataset	Rows	Cols	Orig.	MAF.	OpenFE	DIFER	Top-K	LR	NNLS	Comb.
wine	2554	11	0.905	0.806	0.899	0.902	0.907	**0.907**	0.907	0.907
WMO-Hurricane-Survival-Dataset	4951	20	0.501	0.499	**0.512**	0.507	0.495	0.501	0.493	0.493