Revisiting Chebyshev Polynomial and Anisotropic RBF Models for Tabular Regression

Luciano Gerber [*1] and Huw Lloyd[†1]

[1]*Department of Computing and Mathematics, Manchester Metropolitan University, Dalton Building, Chester Street, Manchester M1 5GD, United Kingdom*

June 16, 2026

Abstract

Smooth-basis models such as Chebyshev polynomial regressors and radial basis function (RBF) networks are well established in numerical analysis. Their continuously differentiable prediction surfaces suit surrogate optimisation, sensitivity analysis, and other settings where the response varies gradually with inputs. Despite these properties, smooth models seldom appear in tabular regression, where tree ensembles dominate. We ask whether they can compete, benchmarking models across 55 regression datasets organised by application domain.

We develop an anisotropic RBF network with data-driven centre placement and gradient-based width optimisation, a ridge-regularised Chebyshev polynomial regressor, and a smooth-tree hybrid (Chebyshev model tree); all three are released as scikit-learn-compatible packages. We benchmark these against tree ensembles, a pre-trained transformer, and standard baselines, evaluating accuracy alongside generalisation behaviour.

The transformer ranks first on accuracy across a majority of datasets, but its GPU dependence, inference latency, and dataset-size limits constrain deployment in the CPU-based settings common across applied science and industry. Among CPU-viable models, smooth models and tree ensembles are statistically tied on accuracy, but the former tend to exhibit tighter generalisation gaps. We recommend routinely including smooth-basis models in the candidate pool, particularly when downstream use benefits from tighter generalisation and gradually varying predictions.

Keywords: tabular regression · Chebyshev polynomials · radial basis functions · generalisation gap · model selection · benchmark

1 Introduction

Practitioners choosing a regression model for tabular problems tend to reach for tree ensembles (e.g., random forests, gradient-boosted trees), which have dominated benchmark leaderboards on predictive accuracy [17, 21]. This paper examines whether two smooth-basis models from numerical analysis, Chebyshev polynomial regressors and radial basis function (RBF) networks, can match tree ensembles on predictive accuracy while offering complementary strengths in generalisation, smoothness, and interpretability. Both are well established in function approximation but seldom applied to tabular regression, despite offering practical advantages alongside accuracy. They are directly usable as surrogates in gradient-based optimisation [18, 29],

[*]L.Gerber@mmu.ac.uk

[†]Now retired.

sensitivity analysis, and other settings where predictions must vary gradually with inputs. Moreover, Chebyshev models yield explicit polynomial coefficients that can be inspected or constrained [46], while RBF models are weighted sums of localised basis functions whose centres and widths carry geometric meaning [10]. These models also train and predict on commodity hardware without GPU acceleration, a practical consideration for the many applied-science and engineering settings where specialised compute is unavailable or uneconomical.

Tabular benchmarks have established a clear picture of predictive accuracy across model families [34, 43], but generalisation gap – the difference between training and test performance – is rarely reported. As an evaluation axis, it can signal excess model capacity and sensitivity to the specific training samples [6], and models that achieve similar held-out scores can differ substantially on it. We argue that generalisation behaviour deserves routine evaluation alongside accuracy, and assess both across diverse application domains.

We revisited these smooth-basis models under contemporary benchmarking standards and with modern optimisation techniques. For RBF networks, we developed an anisotropic formulation (`erbf`) with data-driven centre placement, width initialisation, and gradient-based optimisation of per-dimension widths. These design choices aim to mitigate the instabilities of traditional simultaneous centre-and-width optimisation. For Chebyshev regressors (`chebypoly`), we combined optional pairwise interaction terms with ridge regularisation, synthesising known numerical-analysis building blocks into a practical tool for tabular regression. For settings where the presence of discontinuities is uncertain, we also evaluated a hybrid: Chebyshev model trees (`chebytree`), which use tree splits to identify regime boundaries while fitting Chebyshev polynomials within each leaf. We benchmarked them against established baselines (ridge regression, `ridge`, and a single decision tree, `dt`, representing opposite ends of the bias-variance spectrum), an Explainable Boosting Machine (`ebm`) [35] as a representative of smooth additive models, tree ensembles (Random Forest, `rf`, and XGBoost, `xgb`), and a pre-trained tabular foundation model (TabPFN, `tabpfn`) [27][1] as a representative of the rapidly developing family of transformer-based tabular learners. The comparison spans 55 regression datasets from four domain strata (engineering and simulation; behavioural and social; physics, chemistry, and life sciences; economics and pricing).

Contributions

1. **Multi-axis benchmark.** We evaluated nine models across 55 datasets on predictive accuracy, generalisation gap, and computational cost. By reporting generalisation gap as a standard evaluation axis – not merely a diagnostic – we show that models that are indistinguishable on accuracy can differ markedly in overfitting behaviour.
2. **Model implementations.** The primary modelling contribution is `erbf`, an anisotropic RBF network with a three-stage training pipeline: data-driven centre placement, width initialisation, and decoupled gradient-based width optimisation in log-space. We also provide a Chebyshev polynomial regressor (`chebypoly`) and a Chebyshev model tree (`chebytree`); these synthesise established numerical-analysis components into practical tabular regression tools. All three are sklearn-compatible estimators, available on PyPI (`erbf`, `poly-basis-ml`).

Summary of findings TabPFN ranks first on accuracy across a majority of datasets but requires GPU inference, is constrained by dataset size, and incurs high prediction latency. Among CPU-viable models, the six competitive models (`erbf`, `chebytree`, `xgb`, `chebypoly`, `ebm`, `rf`) are statistically indistinguishable on accuracy (Friedman test with Nemenyi post-hoc comparison, $\alpha = 0.05$), but smooth models tend to exhibit tighter generalisation gaps, with the smooth model showing the smaller gap in 87% of pairwise comparisons at matched accuracy ($|\Delta \bar{R}^2| \leq 0.02$). The common default of gradient-boosted trees may not be optimal: we recommend including

[1] Our experiments use TabPFN v2.5 weights (`tabpfn` 6.3.1), distributed under a non-commercial licence.

smooth-basis models in the candidate pool, particularly when generalisation robustness or smooth prediction surfaces matter for downstream use.

The remainder of this paper is organised as follows. Section 2 reviews the model families, their theoretical foundations, and positions the work relative to existing tabular benchmarks and alternative approaches. Section 3 describes the model architectures, benchmark design, and evaluation protocol. Section 4 presents results on accuracy, generalisation gap, and computational cost. Section 4.6 develops a practical model selection framework. Section 5 discusses limitations and implications, and Section 6 concludes.

2 Background and Related Work

2.1 Radial basis function networks

Radial basis function (RBF) networks approximate a target function as a weighted sum of localised kernels centred at selected points in the input space [9, 10]. Centre selection strategies range from random subsets and k-means clustering to greedy procedures that maximise explained variance [13].

Classical RBF networks have seen limited adoption in applied machine learning. Two commonly cited difficulties are that simultaneously optimising centres and widths is non-convex, and that isotropic widths (a single width per basis function) adapt poorly to anisotropic structure in the data [10, 52].

The generalisation from isotropic to anisotropic basis functions has been explored under several names. Zhou et al. [52] provide a comprehensive survey of *hyper basis function* neural networks (HBFNNs), which replace the Euclidean norm with a weighted or Mahalanobis norm, allowing each unit to stretch along different axes. Our `erbf` model adopts diagonal per-dimension widths (a special case of the HBFNN framework) but contributes a decoupled three-stage training pipeline that sidesteps the joint optimisation difficulties described above; the full design is described in section 3.

2.2 Chebyshev polynomials and polynomial regression

Chebyshev polynomials of the first kind form an orthogonal basis on $[-1, 1]$ with numerical properties that make them attractive for regression: the resulting design matrix has far better conditioning than the monomial basis at the same degree, enabling stable fitting of higher-order expansions [46].

Despite this strong theoretical foundation, Chebyshev polynomials have seldom been used as a basis for supervised regression in machine learning. Two well-established alternatives that emphasise smoothness or interpretability are piecewise or additive in structure: MARS [19] fits adaptive hinge-function splines and generalised additive models [24] sum univariate smooth functions. These methods enforce smoothness within each component or segment but not globally across the prediction surface. Our approach combines a Chebyshev expansion with ridge regularisation [26] to provide a globally smooth alternative: the formulation is linear in its parameters once the basis is constructed, so fitting reduces to a single regularised least-squares solve, yielding a smooth, globally defined predictor. The `chebypoly` implementation is described in section 3.

2.3 Model trees

Model trees combine recursive partitioning with local regression models at the leaves. Quinlan [40] introduced M5, which fits linear models in each leaf node; Wang and Witten [49] refined

the approach as M5′ with simplified pruning. The appeal of model trees lies in their ability to capture regime changes through the tree structure while modelling smooth relationships within each regime. Our `chebytree` extends this idea by replacing linear leaf models with low-degree Chebyshev polynomial fits (section 3).

2.4 Smoothness, stability, and generalisation

Smoothness serves as an implicit regulariser across much of statistical learning. Poggio and Girosi [37] showed that when function approximation is cast as a variational problem penalising derivatives of the fitted surface, the optimal solution takes the form of an RBF or kernel expansion, and ridge regression [26] promotes smoother fits by penalising coefficient magnitude. In a foundational result, Bousquet and Elisseeff [6] linked algorithmic stability – insensitivity to perturbations of individual training examples – to generalisation, bounding leave-one-out error for sufficiently stable learners. We invoke this framework as motivation rather than as a formal guarantee: verifying the Lipschitz conditions it requires for each learning algorithm lies beyond the scope of an empirical benchmark. Belkin et al. [4] demonstrated that this classical picture requires revision in the interpolation regime, where over-parameterised models can generalise well despite fitting training data exactly, but for the moderately parameterised models in our benchmark, the stability-generalisation link remains directly applicable.

With these foundations in place, we now review the broader landscape of tabular regression benchmarks and alternative model families that motivate our experimental design.

2.5 Tabular benchmarks and evaluation

Influential large-scale comparisons consistently rank tree ensembles (random forests, 8; gradient-boosted trees, 14) as the strongest general-purpose learners for tabular data [17, 21, 43]. Recent work adds nuance: McElfresh et al. [34] show that neural architectures occasionally surpass boosted trees when hyperparameters are tuned on equal footing, and TabPFN [27] achieves state-of-the-art accuracy on small-to-moderate datasets through a pre-trained transformer. On the methodological side, Demšar [15] established the widely adopted Friedman and Nemenyi tests for multi-classifier comparison, and Cawley and Talbot [12] demonstrated the necessity of nested cross-validation to avoid selection bias. TabArena, the evaluation benchmark introduced alongside TabPFN 2.5 [22], extends this tradition to industry-scale datasets with up to 100 000 training samples, though its evaluation focuses on accuracy rankings. Our benchmark adopts nested cross-validation with Optuna-based inner tuning [2] and reports generalisation gap alongside accuracy, an axis on which model families diverge even when accuracy is matched.

Several recent model families share our interest in smooth, interpretable, or non-tree alternatives for tabular data, each emphasising different trade-offs.

2.6 Additive and shape-constrained models

Explainable Boosting Machines (EBMs) [35] fit generalised additive models via boosted shallow trees; Neural Additive Models (NAMs) [1] use per-feature neural networks; Kolmogorov–Arnold Networks (KANs) [33] place learnable spline activations on graph edges. These families prioritise interpretability through additivity, but their structural constraints limit the representation of higher-order multivariate interactions beyond pairwise terms. Our models instead prioritise global smoothness through multivariate basis functions: `chebypoly` augments the per-feature Chebyshev basis with multiplicative pairwise interaction terms, while `erbf`'s multi-dimensional Gaussian kernels couple all input dimensions implicitly through each basis function. We include `ebm` in our benchmark as a representative of this additive family, bridging the gap between smooth and tree-based approaches in our comparison.

2.7 Kernel methods and Gaussian processes

Kernel ridge regression and support vector regression scale as $O(n^3)$ or $O(n^2)$ in time and memory without approximation, making them impractical for many of the datasets in our benchmark. Gaussian processes [41] provide principled smoothness control through the kernel length-scale and deliver calibrated uncertainty estimates, but suffer the same cubic scaling; inducing-point methods [45] alleviate the cost but introduce additional hyperparameters and approximation error whose impact varies across datasets. Our `erbf` sidesteps this cost structure by fixing K centres ($K \ll n$) and fitting per-centre widths via analytical gradients, yielding $O(nKd)$ complexity with no kernel matrix inversion.

2.8 Deep learning for tabular data

FT-Transformer [20], TabNet [3], and NODE [38] adapt attention, gating, and differentiable oblivious decision trees, respectively, to tabular inputs. These architectures can match or exceed tree ensembles on sufficiently large datasets; they occupy a different part of the design space from our models, trading smoothness guarantees and CPU-based training for greater representational flexibility. TabPFN [27] takes a distinct approach: pre-trained on millions of synthetic datasets drawn from a learned prior over data-generating processes, it performs in-context learning at inference time without task-specific training or hyperparameter tuning, achieving strong accuracy on small-to-moderate datasets. We include it in our benchmark as a neural comparator. The field continues to evolve rapidly: tabular foundation models such as TabICL [39] and TabPFN 2.5 [22] now match or exceed tree ensembles on classification benchmarks.

3 Methods

This section describes the three models we contribute to the benchmark: an anisotropic RBF network (`erbf`), a Chebyshev polynomial regressor (`chebypoly`), and a hybrid Chebyshev model tree (`chebytree`). `erbf` and `chebypoly` produce globally smooth (continuously differentiable) prediction surfaces; `chebytree` produces piecewise-smooth surfaces, with smooth polynomial fits within each tree leaf. For each, we outline the mathematical formulation and the algorithmic refinements that make it competitive on tabular regression. We also describe the Explainable Boosting Machine (`ebm`), included as a comparator that bridges tree-based and smooth additive modelling. Benchmark design and the evaluation protocol are deferred to section 3.6. Software packages, API details, and installation instructions are given in sections C and C.4.

3.1 Anisotropic RBF Network (`erbf`)

An `erbf`[2] places K Gaussian basis functions in the feature space and predicts as a weighted sum of their activations:

$$f(\mathbf{x}) = \sum_{k=1}^{K} w_k \, \phi_k(\mathbf{x}) + b, \qquad \phi_k(\mathbf{x}) = \exp\left(-\frac{1}{2} \sum_{j=1}^{d} \frac{(x_j - c_{kj})^2}{\sigma_{kj}^2} \right), \tag{1}$$

where $\mathbf{c}_k \in \mathbb{R}^d$ is the centre, $\boldsymbol{\sigma}_k \in \mathbb{R}^d_{>0}$ is the width vector, w_k is the output weight, and b is a bias. The key distinction from isotropic RBF networks is that each basis function has a separate

[2]The "e" in `erbf` stands for *ellipsoidal*: each basis function's iso-activation contour forms an axis-aligned ellipsoid when widths differ across dimensions.

width along each feature dimension ($K \times d$ width parameters in total), allowing anisotropic adaptation to local data structure.

Classical RBF training optimised centres and widths simultaneously, typically via gradient descent or metaheuristics, creating a high-dimensional non-convex problem prone to poor local minima. These training difficulties, together with the limited expressiveness of isotropic widths, contributed to RBF networks falling out of mainstream use. The pipeline described below reduces the severity of the non-convex optimisation landscape by fixing centres before width optimisation begins, while anisotropic widths adapt to per-feature relevance.

The conceptual insight motivating this decomposition is that centre placement and width adaptation serve distinct purposes: centres determine *where* the model allocates representational capacity, while widths determine *how* that capacity responds to local data structure. Conflating these in a single optimisation loop (the classical approach) couples two sources of non-convexity: the loss surface contains both the combinatorial difficulty of choosing good locations and the continuous difficulty of shaping the basis functions around them. Lipschitz-guided centre placement addresses the first problem by exploiting target variation directly: regions where the function changes rapidly need finer coverage, so sampling probability proportional to local Lipschitz estimates concentrates centres where they matter most, without requiring gradient-based search over discrete placements. With centres fixed, width optimisation becomes a continuous problem in $\mathbb{R}^{K \times d}$ that L-BFGS-B [11] can solve efficiently via analytical gradients. The decoupling also mitigates a plausible failure mode of joint optimisation: the optimiser may drive widths toward zero around poorly placed centres to fit local noise, producing sharp peaks that overfit. Fixing centres first constrains width optimisation to refine a geometrically sensible initial configuration rather than compensate for bad placement.

Training proceeds in three decoupled stages.

3.1.1 Stage 1: Centre placement

Two strategies are available. The default, Lipschitz-guided placement, is supervised: we estimate the local Lipschitz constant at each training point as

$$L_i = \max_{j \in \text{kNN}(i)} \frac{|y_i - y_j|}{\|\mathbf{x}_i - \mathbf{x}_j\| + \epsilon}, \tag{2}$$

using five nearest neighbours, where ϵ is a small constant for numerical stability; extreme values are clipped at the 99th percentile for robustness. The K centres are then sampled without replacement from the training set, with selection probability proportional to L_i. This concentrates centres in high-gradient regions where the target function changes rapidly, allocating modelling capacity where it is most needed. The k-NN-based Lipschitz estimate is inherently noisy, particularly in high-dimensional spaces where neighbour distances become less informative. In our benchmark, per-fold feature selection limits effective dimensionality to at most 50 features, and the 99th-percentile clipping guards against outlier estimates. Crucially, the Lipschitz estimate serves only to initialise centre placement; the subsequent width optimisation (Stage 3) compensates for imprecise initial placement. We verify this robustness empirically (see section D.3). The alternative, K-means clustering, is unsupervised and distributes centres according to feature-space density without accounting for target variation.

3.1.2 Stage 2: Width initialisation

Good width initialisation substantially reduces the burden on subsequent optimisation. Two methods are available:

Local ridge (supervised): for each centre $\mathbf{c}_k$, a ridge regression is fitted to its k nearest training points ($k = \min(100, \lfloor n/K \rfloor)$, floored at 10), yielding per-feature coefficients β_j. The initial width is then

$$\sigma_{kj} = \tau \sqrt{\frac{\mathrm{Var}(x_j)}{|\beta_j|}}, \tag{3}$$

where $\mathrm{Var}(x_j)$ is the variance of feature j among the centre's neighbours and $\tau = 1.5\sqrt{d}$ is a dimensional scaling factor (same rationale as for local variance below). Features with strong local predictive power (large $|\beta_j|$) thus receive narrow widths (high sensitivity), while locally irrelevant features receive wide widths (smoothing). *Local variance* (unsupervised): widths are set proportional to the local standard deviation of each feature among the centre's neighbours, scaled by $\sqrt{d}$ so that the Gaussian activation remains approximately $e^{-1/2}$ at the typical neighbour distance regardless of dimensionality (the exponent sums d squared terms, so per-dimension widths must grow with $\sqrt{d}$ to compensate). This purely geometric initialisation ignores target values and is appropriate when features contribute roughly equally.

3.1.3 Stage 3: Width optimisation

With centres fixed, widths are optimised by minimising the mean squared error via L-BFGS-B, operating in log-space ($\theta_{kj} = \log \sigma_{kj}$) to enforce positivity. The width optimisation sub-problem remains non-convex; however, the initialisation from Stage 2 provides a warm start that empirically leads to consistent solutions across random seeds. We present a convergence analysis across multiple initialisations in section D.1. This log-space parameterisation also reduces the scale sensitivity of the width parameters – a width change from 0.01 to 0.02 receives the same gradient magnitude as a change from 1.0 to 2.0 – but does not constitute regularisation in the formal sense. Analytical gradients are computed in $O(n \cdot K \cdot d)$ per iteration; the activation kernel is JIT-compiled via Numba [30] for efficiency.

3.1.4 Output weights

Given the activation matrix $\mathbf{\Phi} \in \mathbb{R}^{n \times K}$, the output weights are obtained by ridge regression,

$$\mathbf{w} = (\mathbf{\Phi}^\top \mathbf{\Phi} + \alpha \mathbf{I}_K)^{-1} \mathbf{\Phi}^\top \mathbf{y}, \tag{4}$$

where $\mathbf{I}_K$ is the $K \times K$ identity matrix.

3.1.5 Automatic scaling of K

Setting the number of centres to its automatic mode applies the empirical heuristic

$$K = \mathrm{clip}\big(\max(40, 2d),\ 20,\ \min(200, n/10)\big), \tag{5}$$

scaling roughly with dimensionality while respecting sample-size and computational constraints.

3.1.6 Hyperparameters

The key hyperparameters are the number of centres K (with an automatic mode described above, or a fixed value), the ridge penalty α for the output weights, the centre placement strategy (Lipschitz-guided or K-means), and the width initialisation method (local ridge or local variance). Full implementation details and search spaces are given in section C and table 5.

3.1.7 Training summary

Algorithm 1 in the appendix summarises the three-stage pipeline.

3.2 Chebyshev Polynomial Regressor (`chebypoly`)

The Chebyshev polynomial regressor expands each input feature into a univariate polynomial basis and fits a linear model in the expanded feature space. We use Chebyshev polynomials of the first kind, T_n, defined by the three-term recurrence

$$T_0(x) = 1, \quad T_1(x) = x, \quad T_{n+1}(x) = 2x\,T_n(x) - T_{n-1}(x), \tag{6}$$

which form an orthogonal basis on $[-1, 1]$. Standard monomial bases $(1, x, x^2, \ldots)$ suffer from severe ill-conditioning at moderate degrees, whereas Chebyshev polynomials remain well-conditioned and achieve near-minimax approximation error [46].

3.2.1 Feature expansion

Given d input features and a maximum degree c, we construct a design matrix whose columns are the Chebyshev evaluations

$$T_0(x_1),\ T_1(x_1),\ \ldots,\ T_c(x_1),\ T_1(x_2),\ \ldots,\ T_c(x_d),$$

yielding $1 + d \cdot c$ basis terms (the constant T_0 is included only for the first feature to avoid redundant intercepts). Inputs are first mapped to $[-1, 1]$ using a min-max scaler, and values outside the training range are clipped for numerical stability.

3.2.2 Interaction terms

Univariate Chebyshev terms capture per-feature nonlinearity but miss bivariate relationships. When interaction terms are enabled, we augment the basis with pairwise product interactions $x_i \cdot x_j$ between scaled features, adding up to $d(d-1)/2$ terms. These are raw products of the scaled features rather than Chebyshev cross-terms; because $T_1(x) = x$, each coincides with the lowest tensor cross term, $x_i x_j = T_1(x_i)\,T_1(x_j)$, so the default basis remains a subset of the orthogonal tensor-product basis (section D.2 gives the full construction and a conditioning comparison against the monomial basis). To limit basis size when dimensionality is high ($d > 30$) we restrict interaction pairs to the top half of features ranked by variance. At the higher interaction complexity setting, each product term is further expanded by appending its Chebyshev evaluation $T_2(x_i \cdot x_j)$, a non-tensor term that departs from strict orthogonality; at the default setting, only the raw products are used.

3.2.3 Regularisation and prediction

The expanded design matrix is solved by ridge regression. We tested elastic net regularisation [53] but found that coordinate descent at low α values incurs substantial computational overhead without improving accuracy; ridge proved sufficient. Predictions are clipped to $\pm 3\sigma$ of the training target to suppress extrapolation artefacts.

3.2.4 Hyperparameters

Two hyperparameters govern the core basis expansion: the polynomial degree c, which sets the expressiveness of the univariate basis, and the ridge penalty α. Two further parameters

control interaction terms: whether to include pairwise product features, and the degree to which those product terms are expanded. Full implementation details and search spaces are given in section C and table 5.

3.3 Chebyshev Model Tree (`chebytree`)

The Chebyshev model tree is a piecewise-smooth hybrid: a decision tree partitions the feature space into axis-aligned regions, and each leaf fits an independent Chebyshev polynomial regressor. This combines the tree's capacity for detecting regime boundaries (thresholds, interaction effects, discontinuities) with the smooth local fits provided by Chebyshev polynomials.

3.3.1 Architecture

A decision tree regressor is grown with tunable maximum depth and minimum leaf size. At prediction time, each sample is routed to its leaf by the tree structure, and the leaf-local Chebyshev polynomial generates the prediction. Leaves with insufficient samples fall back to the tree's own constant prediction to avoid overfitting degenerate local fits. Each leaf model is fitted independently using ridge regression on the Chebyshev-expanded features of the leaf's training subset.

3.3.2 Design rationale

Because each leaf contains only a subset of the training data, leaf polynomials should generally use a lower degree than a global Chebyshev fit to avoid overfitting. Interaction terms are omitted from leaf models both to reduce overfitting risk and to keep the per-leaf basis size tractable. The minimum leaf size can be expressed as a fraction of the training set, ensuring leaves remain large enough for stable polynomial fits.

3.3.3 Hyperparameters

The model exposes four hyperparameters: the per-leaf polynomial degree, the tree depth, the minimum leaf fraction, and the ridge penalty α. At the extremes of this space, a shallow tree with low-degree polynomials yields a simple piecewise-linear model, while a deep tree with high-degree polynomials produces many nonlinear local surfaces. Full implementation details and search spaces are given in section C and table 5.

3.4 Explainable Boosting Machine (`ebm`)

The Explainable Boosting Machine [35] fits a generalised additive model via cyclic gradient boosting of shallow trees, learning one feature at a time and optionally detecting pairwise interactions. Each feature's contribution is a learned shape function, yielding globally additive and locally smooth predictions. We include `ebm` in the benchmark as a bridge between tree-based training and smooth prediction surfaces: it uses tree splits internally to construct shape functions but produces continuous, interpretable output that sums per-feature contributions. This positions it at the intersection of the tree-based and smooth-model families that the benchmark compares. Hyperparameters tuned in the inner loop are the number of histogram bins (`max_bins`), the learning rate, and the minimum number of samples per leaf (`min_samples_leaf`). For tractability under nested cross-validation we disable pairwise interaction detection (`interactions`$=0$) and evaluate `ebm` as a purely additive model; this is a conservative configuration that may understate its full capability. Unlike the other models in

our benchmark, `ebm` does not require feature scaling. The implementation is provided by the InterpretML `interpret` package.

3.5 Datasets

We assembled 55 regression datasets from seven sources: OpenML [47], UCI [16], PMLB [42], MoleculeNet [51], Hugging Face Datasets, scikit-learn generators, and custom synthetic functions. Dataset sizes range from $n = 442$ to 581,835 and dimensionality from $d = 2$ to 1,024; table 10 in the appendix provides the full list with metadata.

3.5.1 Stratification

We grouped the datasets into four strata by application domain, as an exploratory lens for examining whether model performance varies with the type of problem. We group by domain rather than by an objective smoothness measure deliberately: estimating the smoothness of a dataset's target function remains an open problem, and the available measures [e.g., 5, 34] depend on modelling and parameter choices – a probe model or smoother, a neighbourhood size, a frequency cutoff – that carry their own inductive biases, so they would relocate the subjectivity involved rather than remove it. The assignments therefore reflect our judgement about which domains are more likely to produce smooth target functions, or to involve thresholds or discrete structure; they are not a validated classification, and individual datasets may not conform to the stratum-level expectation.

S1: Engineering/Simulation (13 datasets). Systems typically governed by physical laws or smooth equations. Representative datasets include `airfoil_noise`, `power_plant`, `friedman1`, and Feynman physics problems.

S2: Behavioural/Social (10 datasets). Human decision-making, preferences, and demand, often involving implicit thresholds (ratings, satisfaction scores, usage counts). Nine of the ten datasets have non-continuous targets – ordinal ratings and satisfaction scores, or integer counts. Representative datasets include `wine_quality`, `Bike_Sharing_Demand`, and `student_performance`.

S3: Physics/Chemistry/Life Sciences (16 datasets). Natural-science measurements – likely often smooth, but may contain phase transitions or abrupt property changes. Three molecular datasets (`esol`, `freesolv`, `lipophilicity`) are distributed as SMILES strings (see preprocessing below). Representative datasets include `superconduct`, `esol`, `lipophilicity`, and `diabetes`.

S4: Economics/Pricing (16 datasets). Pricing, insurance, and real estate – domains that often involve explicit rules, tax brackets, and policy thresholds. Representative datasets include `california_housing`, `diamonds`, `house_sales`, and `medical_charges`.

3.6 Benchmark Design and Evaluation Protocol

We applied the evaluation protocol detailed below for nine regression models on the 55 datasets described in section 3.5. The models span six groups: two smooth-basis models (`chebypoly`, `erbf`); a smooth-tree hybrid (`chebytree`); a smooth additive model (`ebm`); two tree ensembles (`rf`, `xgb`); a pre-trained tabular transformer (`tabpfn`); and two baselines at opposite ends of the bias-variance spectrum, linear ridge regression (`ridge`) and decision tree (`dt`).

3.6.1 Preprocessing pipeline

All models receive identically preprocessed data. The pipeline has two stages: dataset-level steps applied once before cross-validation splitting, and fold-level steps applied within each outer CV fold to prevent information leakage from feature selection.

3.6.2 Dataset-level preprocessing

These steps are deterministic and do not use target values, so applying them before splitting introduces no leakage.

1. *Featurisation* (dataset-specific). Three molecular datasets distributed as SMILES strings are converted to 16 standard RDKit [31] physicochemical descriptors.
2. *Quality filtering* (all datasets, unconditional). Features with more than 50% missing values are dropped, as are quasi-constant features (mode frequency >95%).
3. *Sample reduction.* Datasets exceeding $n = 50{,}000$ are subsampled without replacement, keeping tuning tractable and satisfying the input constraints of `tabpfn`.

3.6.3 Nested cross-validation

Evaluation follows a nested cross-validation (CV) design to obtain unbiased estimates of generalisation performance while tuning hyperparameters.

- *Outer loop:* 5-fold CV provides held-out test folds for final evaluation.
- *Inner loop:* within each outer training set, hyperparameter search takes place with an adaptive number of inner folds (3-fold CV when $n \geq 1{,}000$, 5-fold otherwise). The model is then refit on the full outer training set using the best configuration before evaluation on the held-out outer test fold.

For models that support early termination (here, XGBoost), 15% of the outer training set is held out as a validation set for early stopping.

3.6.4 Fold-level preprocessing

The following steps are fitted on training data only within each outer CV fold and applied to both training and test splits, preventing information leakage from feature selection.

- *Feature prefiltering.* For datasets with $d \geq 25$: features with $|\rho_{\text{Spearman}}| < 0.05$ (computed on the training fold) are removed except those rescued by MI (features in the bottom 30% by Spearman rank but in the top 30% by MI are retained), preserving non-monotonic relationships that a linear correlation filter would discard.
- *Dimensionality reduction.* If $d > 50$ after prefiltering, the top-k features are selected by mutual information (computed on the training fold) to keep computational costs tractable.
- *Imputation.* Median for numeric features, mode for categorical features.
- *Categorical encoding.* Categorical features are converted to numeric via target encoding (`TargetEncoder` with `smooth='auto'` via `sklearn`).
- *Standardisation* (model-dependent). `ridge` and `tabpfn` receive standardised features; tree-based models do not; Chebyshev and `erbf` models apply internal scaling.

Because feature rankings are computed within each fold, the selected feature sets may vary across folds for the eight datasets where prefiltering or dimensionality reduction is applied.

3.6.5 Hyperparameter tuning

Hyperparameters are optimised with Optuna [2] using the TPE sampler (`multivariate=True`) and a `MedianPruner` for early termination of unpromising trials ($n_{\text{startup}} = 3$, $n_{\text{warmup}} = 1$). Trial budgets are roughly proportional to search-space dimensionality: `ridge` 20, `dt` 25, `rf` 25, `xgb` 50, `erbf` 30, `chebypoly` 30, `chebytree` 30, `ebm` 15; `tabpfn` uses no tuning (zero-shot). Search spaces use log-uniform distributions for regularisation parameters, integer ranges for structural parameters (e.g., tree depth, number of centres), and categorical choices for algorithmic variants (e.g., centre placement, width initialisation). Full per-model search-space definitions are provided in section B.

3.6.6 Evaluation metrics

We report adjusted R^2 (hereafter $\bar{R}^2$) as the primary accuracy metric and the generalisation gap, defined as training minus validation R^2, as a measure of overfitting. For all models, predictions are clipped to $[y_{\min} - 3\sigma,\ y_{\max} + 3\sigma]$ (computed from the training fold targets) to prevent catastrophic extrapolation from inflating error metrics. Each metric is aggregated as both mean $\pm$ standard deviation and median across the five outer folds.

3.6.7 Ranking and statistical testing

Models are ranked per-dataset by $\bar{R}^2$; models that could not run on a dataset receive worst rank (k, the number of models in the comparison). We aggregate via mean rank and top-2 counts (rank-1 and rank-2 placements) across all datasets. Ranks – and the best/second highlighting in the results tables – are computed from the full-precision $\bar{R}^2$, not the rounded values shown. A few datasets on which several models fall within a negligible margin are therefore ordered by differences below the noise floor, and two table entries that round to the same displayed figure may be highlighted differently; the rankings are nonetheless robust to this. Treating models whose $\bar{R}^2$ differ by less than a tolerance as tied – assigning them the average rank – leaves the conclusions unchanged across every tie tolerance we examined, from 0 to 0.02: `erbf` remains the top-ranked model, and the six competitive models stay grouped together and clearly separated from `dt` and `ridge`. Statistical significance of rank differences is assessed with the Friedman test followed by Nemenyi post-hoc comparison at $\alpha = 0.05$ [15], automated via the `autorank` package [25]. The Nemenyi test inherently controls the family-wise error rate across all pairwise comparisons, serving the same purpose as a Bonferroni or Holm correction; no additional multiple-comparison adjustment is therefore required. The Nemenyi critical difference depends only on the number of models and datasets: CD $= 1.635$ for the full nine-model comparison ($k = 9$, $n = 54$) and CD $= 1.416$ for the eight CPU-viable models ($k = 8$, $n = 55$).

3.6.8 Computational setup

Outer CV folds are parallelised across CPU cores via `joblib`. To prevent thread thrashing, BLAS thread counts are pinned to 1 (OMP, MKL, and OpenBLAS environment variables) and model-internal parallelism is likewise disabled (`rf` and `xgb`: `n_jobs=1`). Wall-clock times for tuning, training, and prediction are recorded per fold to enable cost-accuracy trade-off analysis.

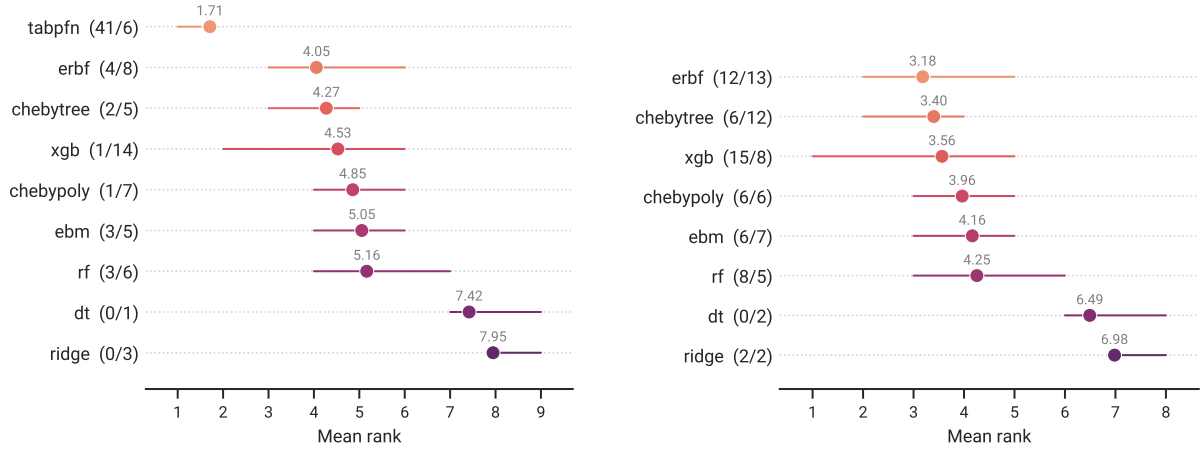
4 Results

This section reports results on accuracy and generalisation gap, together with computational cost, for the eight CPU-viable models. `tabpfn`, which occupies a distinct category as a pre-trained GPU-dependent transformer, is discussed separately in section 4.5. Statistical comparisons use the Friedman test with Nemenyi post-hoc analysis described in section 3.6.

4.1 Predictive Accuracy

Figure 1b summarises predictive accuracy across 55 datasets for the eight CPU-viable models (full numerical results in table 12 in the appendix). The top six models – `erbf`, `chebytree`, `xgb`, `chebypoly`, `ebm`, and `rf` – are statistically tied (all pairwise rank differences within the Nemenyi critical difference, CD $= 1.416$), with mean ranks between 3.18 and 4.25. We refer to these six as the *competitive* models hereafter; `dt` and `ridge` form a separate lower group. The Nemenyi post-hoc test identifies two non-overlapping non-significant groups: {`erbf`, `chebytree`, `xgb`, `chebypoly`,

ebm, rf} and {dt, ridge}; every competitive model is significantly better than both dt and ridge. erbf achieves the best mean rank (3.18) with 12 R^2 wins, followed by chebytree (3.40; 6 wins) and xgb (3.56; 15 wins).



(a) All models

(b) CPU-viable models

Figure 1: Mean rank for predictive accuracy ($\bar{R}^2$); parentheses show rank-1 wins / rank-2 places. Bars show the interquartile range of per-dataset ranks; CD bars indicate the Nemenyi critical difference (section 3.6): models whose mean ranks differ by less than CD are statistically indistinguishable. Full numerical results in tables 11 and 12 (appendix).

Figure 2 shows the full distribution of $\bar{R}^2$ across datasets for each model; the six competitive CPU-viable models have largely overlapping distributions.

4.1.1 Stratum-level analysis

Table 1 breaks down mean rank and $\bar{R}^2$ by domain stratum, examining whether model performance varies with the expected characteristics of the data-generating process (see section 3.5 for strata definitions).

Table 1: Per-stratum mean rank and $\bar{R}^2$ (all nine models). Model order as in Figure 1a. Within each stratum, the best CPU-viable model per column is in **bold** and the second underlined; rank and $\bar{R}^2$ are emphasised independently. TabPFN is shown for reference but excluded from the emphasis.

	S1 (13)		**S2** (10)		**S3** (16)		**S4** (16)	
Model	Rank	$\bar{R}^2$	Rank	$\bar{R}^2$	Rank	$\bar{R}^2$	Rank	$\bar{R}^2$
tabpfn	1.38	0.886	3.60	0.502	1.25	0.715	1.25	0.851
erbf	**3.38**	**0.860**	5.10	0.450	**3.81**	**0.651**	4.19	0.808
chebytree	4.00	0.845	4.40	0.456	4.69	0.625	4.00	0.812
xgb	4.23	0.820	5.40	**0.459**	4.94	0.645	**3.81**	**0.821**
chebypoly	4.46	0.774	4.50	0.457	4.44	0.627	5.81	0.788
ebm	6.23	0.744	**4.00**	0.456	4.88	0.622	4.94	0.797
rf	5.69	0.821	4.50	0.457	5.06	0.630	5.25	0.792
dt	7.00	0.780	7.10	0.420	8.38	0.551	7.00	0.763
ridge	8.62	0.630	6.40	0.346	7.56	0.541	8.75	0.636

Because the strata are a domain-informed lens rather than a validated classification, the

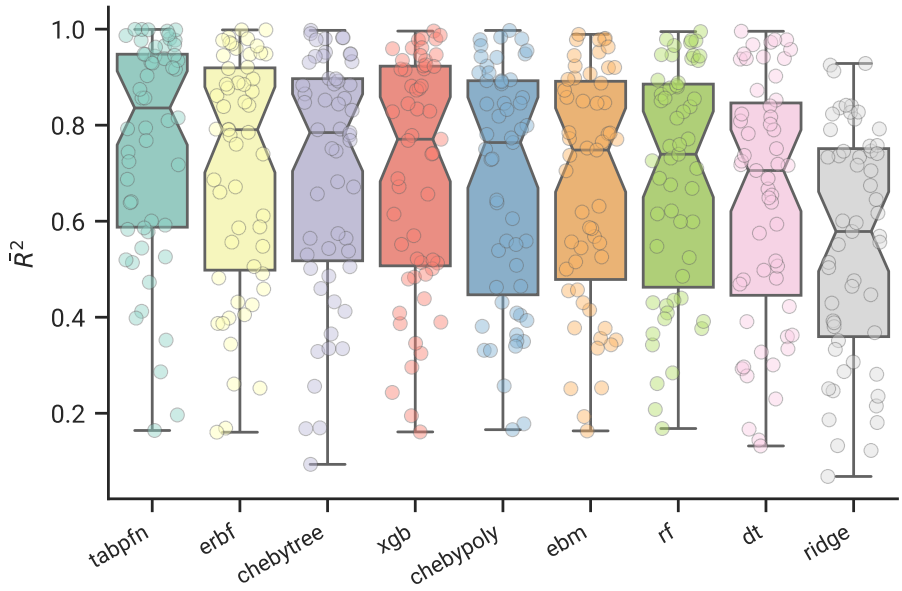


Figure 2: Distribution of $\bar{R}^2$ across 55 datasets per model (ordered by mean rank). Notched boxes show median and IQR; dots are individual datasets. The competitive cluster has largely overlapping distributions, with the main separation occurring in the lower tail.

per-stratum results should be read as consistency checks against a priori expectations rather than independent validation.

Among CPU-viable models, the pattern is consistent with those expectations. `erbf` ranks best in S1 and S3 – the strata where physical or chemical processes suggest smoother target functions – while `xgb` leads in S4, where threshold-based pricing and policy rules could favour split-based models. S2 (behavioural/social data) shows lower accuracy across the board ($\bar{R}^2 \approx$ 0.35–0.46). The competitive models are tightly bunched in mean $\bar{R}^2$ here (within about 0.01, with `xgb` highest at 0.459); `ebm` leads on mean rank (4.00 vs. 4.40–5.40 for the others), possibly reflecting the suitability of its additive structure for modelling human-behavioural variables.

The rank differences, however, are modest (typically less than one rank position between the top CPU-viable models), and the number of datasets per stratum (10–16) limits statistical power. The stratification should be read as a directional guide rather than a sharp prescription: it suggests which function class (smooth or piecewise-constant) may be more likely to suit a given domain, without guaranteeing that a particular model will win. Grouping S1 and S3 into a "smooth" block (29 datasets) and S2 and S4 into a "threshold" block (26 datasets), `erbf` ranks first among CPU-viable models in the smooth block, while `xgb` and `chebytree` share the lead in the threshold block. The hybrid `chebytree` is competitive across all four strata (rank 4.00–4.69), a pattern consistent with its capacity to combine tree-based regime detection with smooth local fits.

4.1.2 Effect of target continuity

We classify a target as non-continuous when it is an intrinsically discrete quantity – an ordinal rating or satisfaction score, an integer count, or a quantised low-resolution measurement – rather than a real-valued measurement. Thirteen of the 55 datasets meet this criterion: six ordinal targets (4–17 levels), four integer counts (10–869 values), and three quantised measurements (35–61 distinct values). All thirteen take distinct values in at most 5% of their samples, corroborating their low effective resolution; because the criterion is intrinsic rather than purely

cardinality-based, a large dataset whose continuous target merely repeats at coarse resolution (*nyc-taxi*, 0.3% distinct values from over 580,000 samples) is treated as continuous. Table 2 compares mean ranks on continuous versus non-continuous targets.

Table 2: Mean rank by target type (CPU-viable models, excluding `tabpfn`). Non-continuous targets are intrinsically discrete (ordinal ratings, integer counts, or quantised low-resolution measurements; see text for the criterion). Best in **bold** per column among competitive models, second underlined.

Model	Continuous (42)	Non-continuous (13)	All (55)
`erbf`	**2.88**	4.15	**3.18**
`chebytree`	3.50	**3.08**	3.40
`xgb`	3.36	4.23	3.56
`chebypoly`	4.19	3.23	3.96
`ebm`	4.29	3.77	4.16
`rf`	4.17	4.54	4.25
`dt`	6.43	6.69	6.49
`ridge`	7.19	6.31	6.98

`erbf` shows the largest rank degradation on non-continuous targets: from 2.88 on continuous data to 4.15 (shift of $+1.27$), with only one win on non-continuous datasets. In contrast, `chebypoly` and `chebytree` both *improve* on non-continuous targets (shifts of -0.96 and -0.42 respectively), with `chebytree` ranking best (3.08).

A plausible architectural explanation is as follows. `erbf`'s localised Gaussian-like basis functions are a natural match for smooth targets but a poor one for step-like structure, which would require many narrow, overlapping bumps. `chebypoly`'s global polynomial basis can approximate piecewise structure through oscillation across the domain, reducing its sensitivity to discrete boundaries. `chebytree` is the most natural fit: its tree routing can partition the feature space into regions of roughly homogeneous target behaviour, while the smooth leaf polynomials handle variation within each partition. We note, however, that these are post-hoc rationalisations of the rank patterns; the benchmark does not isolate the mechanism. On the 42 continuous-target datasets, `erbf`'s mean rank of 2.88 leads the next-best model (`xgb` at 3.36) by 0.48, compared with 0.22 overall, and 11 of its 12 wins fall in this subset.

4.2 Generalisation Gap

The generalisation gap (training R^2 minus test R^2) quantifies how much of the training-set fit fails to transfer to held-out data, serving as a proxy for excess capacity beyond what the generalisable signal requires (section 1). Stability theory provides a motivating link: models whose predictions change little when a single training example is replaced tend to exhibit tighter generalisation bounds [6].

Gap magnitudes are not, however, directly comparable across families: different model families control capacity through fundamentally different mechanisms – early stopping and shrinkage in `xgb`, ridge penalisation in `chebypoly`, basis-function count and width regularisation in `erbf`, pruning depth in `dt`. The matched-accuracy analysis below partially controls for this by restricting comparisons to model pairs that achieve similar held-out performance on each dataset: when accuracy is matched, the remaining gap difference is less likely to be an artefact of differing capacity-control regimes. We do not claim that gap constitutes a formal stability measure; rather, it provides a practical diagnostic that complements accuracy.

Figure 3 summarises the results (full numerical results in table 13 in the appendix). Note that `ridge`'s near-zero gap (median 0.004) is most likely a consequence of underfitting (high

bias) rather than model quality.

Smooth and hybrid models (hereafter collectively 'smooth models' unless explicitly distinguished) exhibit substantially tighter generalisation gaps than tree ensembles. Among competitive models, `chebypoly` achieves the best gap rank (3.47), followed by `ebm` (4.40), `erbf` (4.47), and `chebytree` (4.82); `xgb` has the largest gap among competitive models (mean rank 8.31). These are the nine-model rankings shown in table 13 and fig. 3 (full numerical results in the appendix). `chebypoly` achieves 19 gap wins, followed by `chebytree` and `ebm` (10 each) and `erbf` (9). A tighter gap is consistent with lower training-sample sensitivity (section 4.3 provides empirical support for this association). Despite its tree-based internals, `ebm`'s gap behaviour aligns more closely with the smooth models than with the tree ensembles; its additive structure and cyclic boosting may yield a smoother prediction surface, which could account for this. When accuracy is tied (within 0.02 $\bar{R}^2$), smooth models (including `ebm`) win on gap in 87% of 167 matched comparisons with tree ensembles (`xgb`, `rf`). These matched-accuracy counts are descriptive summaries rather than formal hypothesis tests; they complement the Friedman/Nemenyi analysis above by illustrating the direction and consistency of gap differences when accuracy is controlled for. The pattern is stable across accuracy-matching thresholds spanning an order of magnitude (table 14 in section F.3).

The few gap reversals are concentrated in S4 economic datasets (e.g., *Brazilian_houses*), where pricing tiers and policy thresholds may create genuinely piecewise-constant structure. A tree with a small number of well-placed splits can capture such structure at low effective complexity, yielding similar training and test performance; smooth models approximating the same discontinuities may require more basis functions, increasing effective complexity and with it the gap.

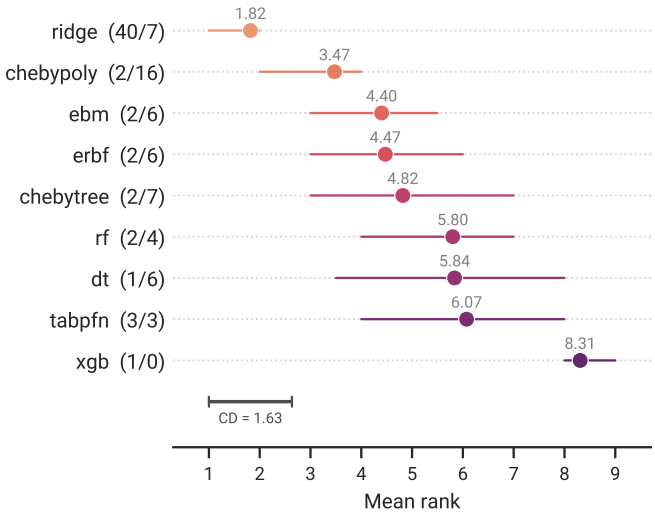
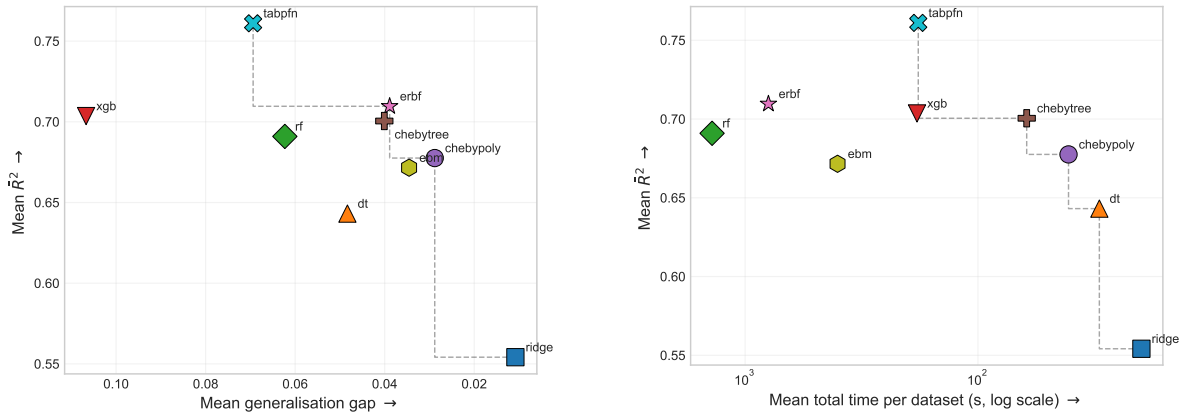


Figure 3: Mean rank for generalisation gap ($R^2_{\text{train}} - R^2_{\text{test}}$); parentheses show rank-1 wins / rank-2 places; bars show the interquartile range. Lower rank = smaller gap. Smooth models (`chebypoly`, `erbf`), `ebm`, and the hybrid `chebytree` cluster at the top; `xgb` ranks last among competitive models. `ridge`'s top position reflects underfitting.

4.2.1 Accuracy–generalisation trade-off

Figure 4a plots each model in ($\bar{R}^2$, gap) space, treating lower gap as better for a given level of accuracy (fig. 4b shows the corresponding accuracy-cost trade-off). Among CPU-viable models, the Pareto front runs from `ridge` (lowest gap, lowest accuracy) through `chebypoly` and `erbf`; the

smooth-basis models occupy the best trade-off positions, achieving competitive accuracy with substantially lower generalisation gaps than `xgb` or `rf`.



(a) Accuracy vs. generalisation gap

(b) Accuracy vs. computational cost

Figure 4: Pareto trade-off diagrams. Each point represents one model at its mean $\bar{R}^2$ plotted against (a) mean generalisation gap and (b) mean total time per dataset including tuning (log scale). Step lines trace the Pareto front; arrows on axes indicate the direction of improvement.

4.3 Prediction stability under retraining

To assess whether the generalisation gap reflects sensitivity to training-sample composition, we measured prediction stability via bootstrap retraining. For each model–dataset pair, we retrained the tuned model 100 times on bootstrap resamples of the training fold, generating prediction distributions for each test instance. Because this refits each tuned model 100 times per dataset, we restrict the analysis to the 29 datasets on which the repeated refitting was computationally feasible; the most demanding large-scale datasets were excluded. Prediction stability is quantified by the prediction coefficient of variation (CoV): for each test instance we compute the CoV of its 100 bootstrap predictions, and summarise each model–dataset pair by the median CoV across test instances (the median rather than the mean, because the CoV is inflated for instances whose mean prediction is near zero). Table 3 reports the generalisation gap and the median prediction CoV per model, each averaged across the 29 datasets.

Table 3: Generalisation gap and median prediction CoV per model, each averaged across the 29 datasets. Models ordered by gap magnitude. Lower values indicate tighter generalisation and more stable predictions, respectively.

Model	Mean gap	Median CoV
`ridge`	0.009	0.021
`chebypoly`	0.025	0.034
`chebytree`	0.030	0.062
`ebm`	0.031	0.060
`erbf`	0.033	0.065
`dt`	0.044	0.095
`rf`	0.050	0.038
`xgb`	0.086	0.072

Within-dataset Spearman correlation between gap and median prediction CoV is positive in

93% of datasets (mean $\rho = 0.61$; pooled across all model–dataset pairs, $\rho = 0.477$, $p < 0.0001$), providing empirical support that models with smaller generalisation gaps tend to produce more stable predictions under retraining. Excluding `rf` strengthens the association ($\rho = 0.71$ within-dataset, 0.50 pooled), indicating that `rf` is the principal source of departure from the general pattern.

`rf` is the exception, and the single decision tree `dt` isolates the cause. Both are regularised tree learners, and their mean gaps are similar (0.044 for `dt`, 0.050 for `rf`); yet `rf`'s median prediction CoV is less than half that of `dt` (0.038 versus 0.095). The structural difference is bagging: `rf` averages many trees fitted to bootstrap resamples, which cancels much of the per-tree variance, so the ensemble's predictions change little when the training sample is resampled [7, 32]. Gap and prediction CoV therefore capture different quantities for a bagged ensemble – the gap reflects the fit of the constituent trees, the CoV the stability of their average – which is why `rf` departs from the general gap–stability association. `xgb` does not show the pattern: it fits trees sequentially to residuals rather than averaging independent trees, so it has no comparable variance-cancelling mechanism, and its gap and prediction variability move together. On 9 of the 29 stability datasets `rf` ranks in the top two for gap but the bottom half for median CoV, consistent with the decoupling being systematic rather than driven by a few outliers.

The gap is therefore a useful stability diagnostic for non-ensemble models, but understates the prediction stability of bagged ensembles. Smooth-basis models achieve both tighter gaps and lower prediction variability, plausibly through a different mechanism: the smoothness of the fitted function may constrain sensitivity to the training sample (excluding `ridge`, whose low values on both measures reflect underfitting rather than model quality).

4.4 Computational Cost

Among the competitive CPU-viable models, `chebypoly` and `chebytree` are the cheapest to tune (40 s and 61 s mean, respectively), since their fitting reduces to ridge regression in the expanded basis. `xgb` (182 s) is moderate; `ebm` (391 s), `erbf` (777 s), and `rf` (1 343 s) are substantially more expensive, driven by cyclic boosting iterations, L-BFGS-B width optimisation, and full forest fitting per Optuna trial, respectively. Once tuned, `ebm` offers the fastest inference among competitive models (2 ms per 1 000 instances), followed by `xgb` and `erbf` (9 ms each) and `chebypoly` (12 ms); all CPU models predict in under 21 ms per 1 000 instances except `rf` (157 ms). Full timing breakdowns are in tables 15 and 16 in the appendix.

The accuracy-cost Pareto front (fig. 4b) shows `ridge`, `dt`, `chebypoly`, and `chebytree` at the fast end; `ebm` occupies an intermediate position with moderate tuning cost but very fast prediction. `erbf` and `rf` fall off the front due to their higher tuning cost.

4.4.1 Scalability check

The main benchmark uses preprocessed data (at most 50 features and 50 000 samples) to keep computational cost manageable for `erbf` and `rf` tuning at high dimensionality. These caps bind on only six of the 55 datasets – five subsampled to 50 000 instances and two reduced to 50 features – so the great majority enter tuning at full size. To test whether the Chebyshev models remain competitive at full scale, we ran the five scalable models (`ridge`, `dt`, `xgb`, `chebypoly`, and `chebytree`) on eight datasets without any feature selection or sample subsampling, including datasets with up to $n = 581{,}835$ samples and $d = 1{,}024$ features (table 4).

On this small sample, `chebytree` matches `xgb` closely (mean $\bar{R}^2$ 0.765 vs. 0.769) and wins on three of the eight datasets; `chebypoly` scores lower (0.728) but remains ahead of `ridge` and `dt`. These results suggest that both Chebyshev models can scale to full-size data without feature selection or subsampling, though the limited number of datasets precludes strong conclusions.

Table 4: Scalability check: $\bar{R}^2$ per dataset for scalable models on full-scale data (no feature selection or subsampling). Datasets ordered by mean $\bar{R}^2$ across models. Best per dataset in **bold**.

Dataset	S	n	d	xgb	chebytree	chebypoly	ridge	dt
medical_charges	S4	163K	3	0.978	**0.980**	0.979	0.825	0.978
diamonds	S4	54K	6	**0.946**	0.945	0.943	0.926	0.942
superconduct	S3	21K	79	**0.910**	0.900	0.837	0.733	0.825
friedman1_d100	S1	2K	100	**0.926**	0.878	0.905	0.710	0.676
geo_music	S2	1K	117	0.647	0.670	**0.758**	0.756	0.473
qsar_tid_11	S3	6K	1024	0.611	**0.634**	0.523	0.538	0.265
allstate_claims	S4	188K	130	0.547	**0.549**	0.482	0.494	0.428
nyc_taxi	S4	582K	9	**0.588**	0.567	0.398	0.308	0.513
Mean				0.769	0.765	0.728	0.661	0.637

xgb's rank does not improve here relative to the main benchmark, which suggests that the preprocessing caps do not systematically favour smooth models.

The distribution of winning hyperparameter configurations across the 55 datasets is reported in section F.5.

4.5 Comparison with TabPFN

The preceding subsections focused on the eight CPU-viable models. tabpfn [27] occupies a distinct position: it is a pre-trained transformer that requires no per-dataset hyperparameter tuning and runs on GPU, differing fundamentally from the fitted models evaluated above. Our benchmark centres on models that train and predict on commodity hardware without GPU acceleration – the CPU-based settings common across applied science and engineering – so tabpfn's reliance on GPU inference places it outside that primary scope. We therefore collect its results here as a separate reference point rather than interleaving them with the CPU-viable analysis.

4.5.1 Accuracy

When included in the nine-model ranking (fig. 1a; full numerical results in table 11), tabpfn achieves 41 rank-1 placements out of 54 datasets and a mean rank of 1.71, placing it ahead of the competitive CPU-viable group.[3] Cross-dataset standard deviation of $\bar{R}^2$ is slightly lower for tabpfn (0.20) than for the six competitive CPU models (0.22–0.23). The stratum breakdown (table 1) shows tabpfn leading in all four strata except S2 (behavioural/social data, rank 3.60 vs. 4.00 for the next-best CPU model), where its advantage is marginal and only 9 of 10 datasets are available.

4.5.2 Generalisation gap

tabpfn's gap (median 0.044, mean rank 6.02; fig. 3) falls between dt and xgb, closer to the tree ensembles than to the smooth models. As a pre-trained model, tabpfn has not been fitted to training data in the conventional sense, so its gap reflects the mismatch between its learned prior and each dataset rather than overfitting as usually understood. The accuracy–gap Pareto front (fig. 4a) places tabpfn at the high-accuracy end with a moderate gap.

[3] One dataset (food_delivery_time) is excluded from the nine-model comparison because tabpfn triggered a GPU out-of-memory error; its rank on that dataset is set to worst (9).

4.5.3 Operational constraints

`tabpfn` requires no tuning but incurs the highest prediction cost among all models tested (3 609 ms median per 1 000 instances, compared with under 21 ms for all CPU models except `rf` at 157 ms). Its GPU memory consumption scales as $O(n^2)$ due to attention over training instances, with a practical ceiling of 50 000 samples; one dataset in our benchmark (`food_delivery_time`, $n \approx 45$K) triggered a CUDA out-of-memory error. These constraints restrict its applicability to moderate-sized problems where GPU inference is available.

Tabular deep learning is an active and rapidly evolving area; a systematic comparison across architectures (e.g., TabPFN, FT-Transformer, GRANDE) would be a valuable but separate study. Our inclusion of `tabpfn` as a single representative provides a reference point for the accuracy frontier while keeping the benchmark's primary focus on CPU-viable model families.

4.6 Practical Model Selection

The preceding subsections evaluated models along individual axes – accuracy, generalisation gap, and computational cost. Here we draw these threads together into practical guidance for model selection. No single model dominates all criteria, and the best choice depends on which trade-offs matter most for a given application; all competitive families should be benchmarked on the target problem. What follows are empirically grounded priors to inform that process.

Domain knowledge can inform priors about which family to favour. The stratum analysis (section 4.1.1) offers a directional guide: smooth models tend to rank higher on datasets from engineering, simulation, and physical-science domains (S1, S3), while tree ensembles tend to lead on economic and pricing datasets with threshold-driven structure (S4). These are tendencies, not guarantees (section 3.5).

When models achieve comparable accuracy on a given problem – which, given the aggregate tie, can often be the case – other criteria become decisive. The matched-accuracy gap advantage (87% of pairwise comparisons, section 4.2) and the accuracy-gap Pareto front (fig. 4a) both favour smooth models. Depending on the application, a practitioner may also choose to accept a small accuracy trade-off in favour of smooth prediction surfaces (for surrogate optimisation or sensitivity analysis), structural interpretability, or lower computational cost.

On computational cost (section 4.4), `chebypoly` and `chebytree` are the fastest competitive models, placing them on the accuracy-cost Pareto front (fig. 4b). `erbf` and `rf` fall off this front due to expensive tuning, though `erbf`'s fast inference (table 15) means the tuning cost is a one-time investment amortised over subsequent predictions.

Scalability further differentiates the models. The scalability check (table 4) showed that `xgb`, `chebytree`, and `chebypoly` all run at full scale without preprocessing, though the Chebyshev basis size grows combinatorially with dimensionality. `erbf`'s $O(nKd)$ per-iteration cost makes it the most expensive smooth model to tune at high n or d.

Smooth models also offer a degree of structural interpretability not shared by tree ensembles. `chebypoly` yields explicit polynomial coefficients from which partial derivatives and sensitivity indices can be computed; `chebytree` combines tree routing with local polynomial coefficients; `erbf` provides geometric interpretability via localised basis functions whose widths indicate local feature relevance. `xgb` and `rf` are more commonly interpreted through post-hoc tools (SHAP, partial dependence), though these tools are equally applicable to smooth models. These are structural properties of the model architectures; a comparative evaluation of interpretability is beyond the scope of this benchmark. As noted in section 1, smooth prediction surfaces are additionally valuable when outputs feed into gradient-based optimisation or sensitivity analysis.

In summary, when generalisation gap and interpretability are priorities, `erbf` and `chebypoly`

offer the strongest profiles among CPU-viable models. When tuning budget is limited, `chebytree` and `chebypoly` provide competitive accuracy at the lowest cost. When scalability to high-dimensional data is the binding constraint, `xgb` and `chebytree` both proved viable at full scale. Where GPU inference is available, `tabpfn` can extend the accuracy frontier further (section 4.5).

5 Discussion

5.1 Scope and limitations

Several considerations qualify the conclusions of this study.

The benchmark is restricted to regression. Classification tasks, which dominate many tabular benchmarks, may favour different model families.

Our evaluation uses a fixed nested cross-validation protocol with Optuna-based hyperparameter tuning. Trial budgets (20–50 per model, proportional to search space dimensionality) were chosen to keep resource usage comparable across models; more trials could shift relative rankings, particularly for `erbf`, whose width optimisation involves a non-convex inner loop that benefits from broader search.

The main preprocessing pipeline applies mutual-information-based feature selection (capping at 50 features) and sample subsampling (capping at 50 000 instances) to keep nested-CV tuning tractable across all model families; the same thresholds conveniently satisfy `tabpfn`'s input constraints. Although these caps are applied uniformly, their effect on different model families is not necessarily symmetric. The scalability check (table 4) partially addresses this by running scalable models on full-scale data without preprocessing, but a complete comparison at full scale for all models would require substantially more computation. Relatedly, we applied a uniform preprocessing pipeline across all datasets rather than tailoring feature engineering to each domain; domain-specific feature construction (e.g. task-specific molecular fingerprints instead of generic RDKit descriptors) might improve all models. Results on individual datasets should therefore be interpreted as reflecting the model-pipeline combination, not the model alone.

ERBF's training complexity is $O(nKd)$ per L-BFGS-B iteration, where n is the number of samples, K the number of centres, and d the dimensionality. In our experiments, ERBF trained comfortably on datasets up to approximately 10 000 samples and 50 features on a standard multi-core CPU, with mean tuning time of 777 s compared to 182 s for XGBoost. For larger problems, the absence of mini-batch support and the growth of the kernel matrix with K become limiting factors; extending ERBF with stochastic optimisation or inducing-point approximations is a natural direction for future work.

All train-from-scratch models in the benchmark operate in axis-aligned feature coordinates: trees splits act on single features, ERBF uses diagonal (per-dimension) widths, and Chebyshev interaction terms are products along coordinate axes. Data with strong oblique structure (relationships aligned along linear combinations of features) may disadvantage all families equally; preprocessing such as PCA rotation could benefit all models.

The generalisation gap metric conflates multiple sources of train–test divergence – capacity, regularisation strength, and optimisation dynamics – so cross-family comparisons of gap magnitude should be interpreted with caution; the matched-accuracy analysis (section 4.2) partially mitigates this but does not eliminate it.

We evaluated 55 datasets from four domain strata, a number that is large by the standards of tabular regression benchmarks but still insufficient to make definitive claims about specific application domains. The stratum-level analysis (section 4.1.1) is exploratory rather than confirmatory.

5.2 Future work

Extending the evaluation to classification is the most natural next step. Formal assessment of prediction-surface regularity, for example via local Lipschitz analysis, could complement the generalisation gap analysis presented here. For ERBF specifically, scaling the $O(nKd)$ per-iteration cost via mini-batch optimisation or sparse activations remains an open direction. Hybrid architectures that combine tree-based routing with local smooth fits, or target transformations that map discrete outcomes to a continuous latent space, could address ERBF's sensitivity to non-continuous targets.

5.3 Broader implications

The accuracy-gap trade-off documented in section 4.6 has implications beyond the specific models and datasets studied here. Our exploratory stratum analysis (section 4.1.1) offers a tentative counterpoint to the established narrative that tree ensembles dominate tabular regression. Smooth models rank highest in the engineering and natural-science strata (S1, S3), a pattern consistent with a priori expectations that these domains involve smoother target functions; even in the economics and behavioural strata (S2, S4) – where threshold-driven targets might favour split-based models – tree ensembles hold only a narrow lead or none at all. The strata are based on domain judgement rather than formal criteria, and the per-stratum sample sizes are small, so these patterns should be read as suggestive rather than conclusive; in particular, they cannot serve as independent confirmation that smooth models excel on smooth data, since the stratification itself encodes that hypothesis. They do, however, hint that the dominance of tree ensembles may be less universal than benchmark leaderboards – which draw heavily from commercial and administrative domains – suggest. More broadly, evaluating generalisation gap alongside accuracy is informative because models that achieve similar held-out scores can differ substantially in their sensitivity to the specific training sample. Stability theory motivates the expectation that lower sensitivity leads to tighter generalisation bounds [6], and our bootstrap retraining analysis (section 4.3) provides empirical support for this association.

The practical benefits of smooth prediction surfaces extend beyond the benchmark setting. In surrogate-based engineering design, an optimiser that follows the surrogate's gradient may become trapped or oscillate when the surface contains artefactual discontinuities at tree-split boundaries; the optimiser chases model structure rather than genuine improvements in the objective – a practical consideration that motivates the use of smooth surrogates in model-based optimisation [18, 29]. In consumer-facing applications such as loan calculators or insurance quotations, users expect that small changes to inputs produce proportionate changes in outputs; a split boundary that causes a large price jump for a negligible input change undermines trust. These settings do not require certified smoothness guarantees, but they benefit from prediction surfaces that vary gradually.

The evaluation framework developed here – combining accuracy, generalisation gap, and computational cost – is applicable to any regression model and could serve as a template for future tabular benchmarks. The generalisation advantage we observe may extend to other model families that produce smooth prediction surfaces, though this remains to be tested.

6 Conclusion

We have benchmarked two smooth-basis model families (Chebyshev polynomial regressors and anisotropic RBF networks) and a smooth-tree hybrid (Chebyshev model trees) against tree ensembles and a pre-trained transformer across 55 tabular regression datasets, evaluating

predictive accuracy and generalisation behaviour.

The main finding is that smooth-basis models match tree ensembles on predictive accuracy while tending to generalise better: the six competitive CPU-viable models are statistically indistinguishable on accuracy, but smooth models exhibit tighter generalisation gaps in the large majority of pairwise comparisons at matched accuracy (section 4.6).

These results do not argue that smooth models should replace tree ensembles universally; model selection depends on the balance of aspects including accuracy, generalisation, cost, and interpretability for each application (section 4.6). Rather, they suggest that smooth models deserve routine inclusion in the candidate pool, and that the practitioner's default of gradient-boosted trees, while reasonable, could leave potential gains in other evaluative axes on the table. When two models achieve comparable accuracy, for example, one could favour the one with the tighter generalisation gap.

6.1 Reproducibility

All experiments use fixed random seeds (outer CV seed 42, Optuna seed 0) and deterministic nested cross-validation splits. The model implementations are available on PyPI (`pip install erbf poly-basis-ml`), and the full benchmark code–including analysis scripts, dataset loaders, and result summaries–is available at `https://github.com/gerberl/poly-erbf-benchmark`. The benchmark was run with Python 3.12, scikit-learn 1.6, XGBoost 3.1, Optuna 4.6, and NumPy 2.3; a complete environment specification (`environment.yml`) is included in the repository. Datasets are drawn from OpenML, UCI, PMLB, MoleculeNet, Hugging Face Datasets, scikit-learn generators, and custom synthetic functions (sources and IDs listed in section F.6). Experiments were conducted on a workstation with an Intel Core i9-9900X CPU (10 cores, 20 threads, 3.50 GHz), 128 GB RAM, and an NVIDIA TITAN RTX GPU (24 GB VRAM), running Linux 6.1. Hyperparameter search spaces are fully specified in section B, and per-dataset results are reported in section F.6.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Funding

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

CRediT authorship contribution statement

Luciano Gerber: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Data curation, Writing (original draft, review, editing), Visualization.
Huw Lloyd: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Data curation, Writing (original draft, review, editing), Visualization.

Data availability

All datasets used in this study are publicly available. OpenML datasets are identified by their OpenML IDs listed in the appendix. Synthetic datasets are generated using functions provided in the benchmark code repository. The benchmark code, analysis scripts, and full results are available at https://github.com/gerberl/poly-erbf-benchmark.

Declaration of generative AI and AI-assisted technologies in the manuscript preparation process

During the preparation of this manuscript, the authors used generative AI tools (including large language models) to support tasks such as drafting, language refinement, code development, and exploratory analysis. All outputs were critically reviewed, verified, and edited by the authors, who take full responsibility for the content of the published article.

References

[1] Rishabh Agarwal, Levi Melnick, Nicholas Frosst, Xuezhou Zhang, Ben Lengerich, Rich Caruana, and Geoffrey E. Hinton. Neural additive models: Interpretable machine learning with neural nets. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

[2] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2623–2631, 2019.

[3] Sercan Ö. Arik and Tomas Pfister. TabNet: Attentive interpretable tabular learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 6679–6687, 2021.

[4] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, 2019.

[5] Ege Beyazit, Jonathan Kozaczuk, Bo Li, Vanessa Wallace, and Bilal Fadlallah. An inductive bias for tabular deep learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

[6] Olivier Bousquet and André Elisseeff. Stability and generalization. *Journal of Machine Learning Research*, 2:499–526, 2002.

[7] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[8] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[9] David S. Broomhead and David Lowe. Radial basis functions, multi-variable functional interpolation and adaptive networks. *Royal Signals and Radar Establishment Memorandum*, (4148), 1988.

[10] Martin D. Buhmann. *Radial Basis Functions: Theory and Implementations*. Cambridge University Press, 2003. doi: 10.1017/CBO9780511543241.

[11] Richard H. Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5): 1190–1208, 1995.

[12] Gavin C. Cawley and Nicola L. C. Talbot. On over-fitting in model selection and subsequent selection bias in performance evaluation. *Journal of Machine Learning Research*, 11: 2079–2107, 2010.

[13] Sheng Chen, Colin F. N. Cowan, and Peter M. Grant. Orthogonal least squares learning algorithm for radial basis function networks. *IEEE Transactions on Neural Networks*, 2(2): 302–309, 1991.

[14] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794, 2016.

[15] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

[16] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL https://archive.ics.uci.edu/ml.

[17] Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, and Dinani Amorim. Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research*, 15(1):3133–3181, 2014.

[18] Alexander I. J. Forrester and Andy J. Keane. Recent advances in surrogate-based optimization. *Progress in Aerospace Sciences*, 45(1–3):50–79, 2009.

[19] Jerome H. Friedman. Multivariate adaptive regression splines. *The Annals of Statistics*, 19 (1):1–67, 1991.

[20] Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. Revisiting deep learning models for tabular data. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

[21] Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. Why do tree-based models still outperform deep learning on typical tabular data? In *Advances in Neural Information Processing Systems*, volume 35, pages 507–520, 2022.

[22] Léo Grinsztajn, Klemens Flöge, Oscar Key, Felix Birkel, Philipp Jund, Brendan Roof, Benjamin Jäger, Dominik Safaric, Simone Alessi, Adrian Hayler, Mihir Manium, Rosen Yu, Felix Jablonski, Shi Bin Hoo, Anurag Garg, Jake Robertson, Magnus Bühler, Vladyslav Moroshan, Lennart Purucker, Clara Cornu, Lilly Charlotte Wehrhahn, Alessandro Bonetto, Bernhard Schölkopf, Sauraj Gambhir, Noah Hollmann, and Frank Hutter. TabPFN-2.5: Advancing the state of the art in tabular foundation models, 2025. URL https://arxiv.org/abs/2511.08667.

[23] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Krevelen, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585:357–362, 2020.

[24] Trevor J. Hastie and Robert J. Tibshirani. *Generalized Additive Models*. Chapman and Hall, 1990.

[25] Steffen Herbold. Autorank: A Python package for automated ranking of classifiers. *Journal of Open Source Software*, 5(48):2173, 2020. doi: 10.21105/joss.02173.

[26] Arthur E. Hoerl and Robert W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

[27] Noah Hollmann, Samuel Müller, Lennart Purucker, Arjun Krishnakumar, Max Körfer, Shi Bin Hoo, Robin Tibor Schirrmeister, and Frank Hutter. Accurate predictions on small data with a tabular foundation model. *Nature*, 2025. doi: 10.1038/s41586-024-08328-6.

[28] John D. Hunter. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[29] Donald R. Jones, Matthias Schonlau, and William J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13(4):455–492, 1998.

[30] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. Numba: A LLVM-based Python JIT compiler. *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, pages 1–6, 2015.

[31] Greg Landrum. RDKit: Open-source cheminformatics software, 2016. URL https://www.rdkit.org. https://www.rdkit.org.

[32] Yi Lin and Yongho Jeon. Random forests and adaptive nearest neighbors. *Journal of the American Statistical Association*, 101(474):578–590, 2006.

[33] Ziming Liu, Yixuan Wang, Sachin Vaidya, Fabian Ruehle, James Halverson, Marin Soljačić, Thomas Y. Hou, and Max Tegmark. KAN: Kolmogorov–Arnold networks. In *International Conference on Learning Representations*, 2025.

[34] Duncan McElfresh, Sujay Khandagale, Jonathan Valverde, Vishak Prasad C, Benjamin Feuer, Chinmay Hegde, Ganesh Ramakrishnan, Micah Goldblum, and Colin White. When do neural nets outperform boosted trees on tabular data? In *Advances in Neural Information Processing Systems*, volume 36, 2023.

[35] Harsha Nori, Samuel Jenkins, Paul Koch, and Rich Caruana. InterpretML: A unified framework for machine learning interpretability. *arXiv preprint arXiv:1909.09223*, 2019.

[36] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[37] Tomaso Poggio and Federico Girosi. Networks for approximation and learning. *Proceedings of the IEEE*, 78(9):1481–1497, 1990.

[38] Sergei Popov, Stanislav Morozov, and Artem Babenko. Neural oblivious decision ensembles for deep learning on tabular data. In *International Conference on Learning Representations*, 2020.

[39] Jingang Qu, David Holzmüller, Gaël Varoquaux, and Marine Le Morvan. TabICL: A tabular foundation model for in-context learning on large data. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025.

[40] J. Ross Quinlan. Learning with continuous classes. In *Proceedings of the 5th Australian Joint Conference on Artificial Intelligence*, pages 343–348, 1992.

[41] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[42] Joseph D. Romano, Trang T. Le, William La Cava, John T. Greber, Daniel E. Goldberg, and Jason H. Moore. PMLB v1.0: An open-source dataset collection for benchmarking machine learning methods. *Bioinformatics*, 38(3):878–880, 2022.

[43] Ravid Shwartz-Ziv and Amitai Armon. Tabular data: Deep learning is not all you need. *Information Fusion*, 81:84–90, 2022.

[44] The Joblib Developers. Joblib, 2025. URL https://github.com/joblib/joblib.

[45] Michalis K. Titsias. Variational learning of inducing variables in sparse Gaussian processes. In *Proceedings of the 12th International Conference on Artificial Intelligence and Statistics*, pages 567–574, 2009.

[46] Lloyd N. Trefethen. *Approximation Theory and Approximation Practice, Extended Edition*. SIAM, 2019.

[47] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luís Torgo. OpenML: Networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60, 2014.

[48] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C. J. Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, 17:261–272, 2020.

[49] Yong Wang and Ian H. Witten. Induction of model trees for predicting continuous classes. *Working Paper Series, Department of Computer Science, University of Waikato*, (96/23), 1997.

[50] Michael L. Waskom. seaborn: statistical data visualization. *Journal of Open Source Software*, 6(60):3021, 2021.

[51] Zhenqin Wu, Bharath Ramsundar, Evan N. Feinberg, Joseph Gomes, Caleb Geniesse, Anurag S. Pappu, Karl Leswing, and Vijay Pande. MoleculeNet: A benchmark for molecular machine learning. *Chemical Science*, 9(2):513–530, 2018.

[52] Yuguo Zhou, Tong Mu, Zhong-Hua Pang, and Changbing Zheng. A survey on hyper basis function neural networks. *Systems Science & Control Engineering*, 7(1):399–415, 2019. doi: 10.1080/21642583.2019.1699474.

[53] Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320, 2005.

A ERBF Training Pipeline

Algorithm 1 Three-stage ERBF training pipeline.

Require: Training data $\{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$, number of centres K, ridge penalty α, neighbourhood size k_{nn}

Ensure: Centres $\{\mathbf{c}_k\}$, widths $\{\boldsymbol{\sigma}_k\}$, weights $\{w_k\}$, bias b

Stage 1: Centre placement ▷ Hyperparameter: `center_init`

if Lipschitz-guided (default) **then**
  for $i = 1, \ldots, n$ **do**
    Compute local Lipschitz constant $L_i = \max_{j \in \text{kNN}(i)} \frac{|y_i - y_j|}{\|\mathbf{x}_i - \mathbf{x}_j\| + \epsilon}$
  end for
  Clip L_i at the 99th percentile
  Sample K centres $\{\mathbf{c}_k\}$ from training points without replacement, with $P(\mathbf{x}_i) \propto L_i$
else if K-means **then**
  Run K-means on training features $\rightarrow$ centroids $\{\mathbf{c}_k\}$
end if

Stage 2: Width initialisation ▷ Hyperparameter: `width_init`

for $k = 1, \ldots, K$ **do**
  Find k_{nn} nearest neighbours of $\mathbf{c}_k$
  if Local ridge (default) **then**
    Fit local ridge regression $\rightarrow$ coefficients β_j
    Set $\sigma_{kj}^2 \leftarrow \text{Var}_{\text{local}}(x_j) \,/\, |\beta_j|$ for $j = 1, \ldots, d$
  else if Local variance **then**
    Set $\sigma_{kj} \leftarrow \text{std}_{\text{local}}(x_j) \cdot \sqrt{d}$ for $j = 1, \ldots, d$
  end if
end for

Stage 3: Width optimisation

$\boldsymbol{\theta} \leftarrow \log \boldsymbol{\sigma}$ ▷ Enforce positivity via log-space

$\boldsymbol{\theta}^* \leftarrow \text{L-BFGS-B}\big(\min_{\boldsymbol{\theta}} \frac{1}{n} \sum_{i=1}^{n} (y_i - f(\mathbf{x}_i))^2,\ \boldsymbol{\theta}_0 = \boldsymbol{\theta},\ \text{maxiter} = 30\big)$

$\boldsymbol{\sigma} \leftarrow \exp(\boldsymbol{\theta}^*)$

Output weights

Compute activation matrix $\boldsymbol{\Phi} \in \mathbb{R}^{n \times K}$ with $\Phi_{ik} = \phi_k(\mathbf{x}_i)$ ▷ Eq. 1

$\mathbf{w} \leftarrow (\boldsymbol{\Phi}^\top \boldsymbol{\Phi} + \alpha \mathbf{I})^{-1} \boldsymbol{\Phi}^\top \mathbf{y}$ ▷ $\mathbf{I} \in \mathbb{R}^{K \times K}$ identity matrix

B Hyperparameter Search Spaces

Table 5 lists the full per-model search spaces used by Optuna during the inner loop of nested cross-validation. All continuous regularisation parameters are sampled on a log-uniform scale; structural parameters (depths, counts) use uniform integer sampling; algorithmic variants use categorical sampling. Fixed (non-tuned) settings are shown as greyed rows.

Table 5: Hyperparameter search spaces and fixed settings for each model. "Log" indicates log-uniform sampling; "Cat" indicates categorical. Trial budgets are given in parentheses after each model name. Greyed rows show fixed (non-tuned) settings.

Model	Parameter	Range / choices	Scale
Ridge (20 trials)	`alpha`	$[10^{-3}, 10^{3}]$	Log
Decision Tree (25 trials)	`max_depth`	$[1, 20]$	Int
	`min_samples_leaf`	$[0.005, 0.1]$	Uniform
	`min_samples_split`	$[0.01, 0.1]$	Uniform
Random Forest (25 trials)	`n_estimators`	$[50, 500]$	Int
	`max_depth`	$[3, 20]$	Int
	`max_features`	{0.3, 0.5, 0.7, sqrt}	Cat
	`min_samples_leaf`	0.005	Fixed
XGBoost (50 trials)	`max_depth`	$[1, 9]$	Int
	`learning_rate`	$[0.01, 0.3]$	Log
	`subsample`	$[0.6, 1.0]$	Uniform
	`colsample_bytree`	$[0.6, 1.0]$	Uniform
	`reg_lambda`	$[10^{-3}, 10^{3}]$	Log
	`min_child_weight`	$[1, 100]$	Log
	`n_estimators`	2 000 (early stop, patience 10)	Fixed
	validation split	15% of training set	Fixed
ERBF (30 trials)	`n_rbf`	{auto} $\cup$ $[10, 80]$	Cat / Int
	`alpha`	$[10^{-3}, 10^{3}]$	Log
	`center_init`	{lipschitz, kmeans}	Cat
	`width_init`	{local_ridge, local_variance}	Cat
	`width_optim_iters`	30	Fixed
	`width_mode`	full (gradient-based)	Fixed
ChebyPoly (30 trials)	`complexity`	$[1, 14]$	Int
	`alpha`	$[10^{-3}, 10^{3}]$	Log
	`include_interactions`	{True, False}	Cat
	`max_interaction_complexity`	{1, 2}	Cat[†]
	`solver` / `clip_input`	Ridge; True	Fixed
	`interaction_types`	[product]	Fixed
ChebyTree (30 trials)	`complexity`	$[1, 6]$	Int
	`alpha`	$[10^{-3}, 10^{3}]$	Log
	`max_depth`	$[1, 12]$	Int
	`min_samples_leaf`	$[0.01, 0.1]$	Uniform
	`solver`	Ridge	Fixed
EBM (15 trials)	`max_bins`	{128, 256}	Cat
	`learning_rate`	$[0.01, 0.1]$	Log
	`min_samples_leaf`	{2, 4, 10}	Cat
	`interactions`	0 (pure additive)	Fixed
	`max_leaves`	3	Fixed
	`outer_bags`	4	Fixed
	`max_rounds` / `early_stopping_rounds`	5 000 / 50	Fixed
`tabpfn` (0 trials)	Zero-shot; no hyperparameters tuned.		

[†] Conditional on `include_interactions` = True. All stochastic models use `random_state` = 42.

C Implementation and Reproducibility

All three contributed models are implemented in Python, building on NumPy [23], SciPy [48], scikit-learn [36], and joblib [44] for parallelisation. The Chebyshev models are provided by the

poly_basis_ml package and the RBF network by the erbf package; both expose scikit-learn-compatible estimators (fit/predict interface) and are available on PyPI (pip install erbf poly-basis-ml) and GitHub.[4]

C.1 ERBF implementation details

The four tuned hyperparameters and their API names are:

- n_rbf: the number of centres K (integer, or 'auto' for the adaptive heuristic in eq. (5));
- alpha: the ridge penalty α for the output weights;
- center_init: the centre placement strategy ('lipschitz' for Lipschitz-guided placement, 'kmeans' for K-means clustering);
- width_init: the width initialisation method ('local_ridge' for supervised local ridge, 'local_variance' for unsupervised local variance).

The width optimisation iteration count (width_optim_iters = 30) and gradient-based width mode are fixed throughout.

C.2 ChebyPoly implementation details

The Vandermonde matrix is constructed using numpy.polynomial.chebyshev. Inputs are mapped to $[-1, 1]$ via min-max scaling; out-of-range values are clipped (clip_input = True) for numerical stability. The four tuned hyperparameters are:

- complexity: the polynomial degree c;
- alpha: the ridge penalty α;
- include_interactions: whether to generate pairwise product features (Boolean);
- max_interaction_complexity: the degree to which product terms are expanded (1 = raw products only, 2 = products plus T_2 evaluation).

C.3 ChebyTree implementation details

The four tuned hyperparameters are:

- complexity: the per-leaf polynomial degree;
- max_depth: the maximum tree depth;
- min_samples_leaf: the minimum leaf size (expressed as a fraction of the training set);
- alpha: the ridge penalty α for leaf-level ridge regression.

C.4 Code availability and installation

The model packages and benchmark code are publicly available under the MIT licence.

C.4.1 Model packages

Both packages are on PyPI and can be installed with:

```
pip install erbf poly-basis-ml
```

[4] https://github.com/gerberl/erbf and https://github.com/gerberl/poly-basis-ml

Source repositories:

- https://github.com/gerberl/erbf
- https://github.com/gerberl/poly-basis-ml

C.4.2 Benchmark code

The full benchmark–including data loaders, tuning pipeline, analysis scripts, and pre-computed results–is available at:

https://github.com/gerberl/poly-erbf-benchmark

To reproduce the experiments:

```
git clone https://github.com/gerberl/poly-erbf-benchmark.git
cd poly-erbf-benchmark
micromamba create -n polyerbfbench -f environment.yml
micromamba activate polyerbfbench
pip install erbf poly-basis-ml
python scripts/run_benchmark.py --max-features 50 --max-samples 50000
```

The repository README provides full instructions, including separate commands for Benchmark A (all models, preprocessed data) and Benchmark B (scalable models, full-scale data).

C.5 Software versions

All experiments were run under Python 3.12 using the `micromamba` package manager. Table 6 lists the key packages and their versions. `tabpfn` uses the v2.5 model weights distributed with `tabpfn` package version 6.3.1 and run in local GPU mode on a 24 GB TITAN RTX.

Table 6: Software versions used in all experiments.

Package	Version	Role
Model implementations		
`tabpfn`	6.3.1 (v2.5 weights)	TabPFN regressor
`xgboost`	3.1.3	XGBoost
`scikit-learn`	1.6.1	Ridge, Decision Tree, Random Forest
`erbf`	0.1.0	ERBF regressor
`poly_basis_ml`	0.2.0	ChebyPoly, ChebyTree regressors
Infrastructure		
Python	3.12.12	Runtime
`optuna`	4.6.0	Hyperparameter tuning
`numpy`	2.3.5	Numerical computation
`scipy`	1.17.0	Optimisation (L-BFGS-B for ERBF)
`pandas`	2.3.3	Data handling
`joblib`	1.5.3	Parallel execution and result serialisation
`rdkit` [31]	2025.09	Molecular descriptor computation
`matplotlib` [28]	3.10	Figure generation
`seaborn` [50]	0.13	Figure generation

D Method-Validation Studies

The following three analyses examine the robustness of the smooth-model design choices: the convergence of ERBF width optimisation across random seeds, the numerical conditioning of the Chebyshev basis, and ERBF's sensitivity to its centre- and width-initialisation strategies.

D.1 ERBF convergence analysis

To assess the sensitivity of the width optimisation to initialisation, we trained ERBF with five different random seeds on five representative datasets spanning different sizes and dimensionalities. Table 7 reports the mean and standard deviation of test R^2 and generalisation gap across seeds. Across all five datasets, the standard deviation of test R^2 across seeds is at most 0.012, suggesting that width optimisation converges to solutions of similar quality across the initialisations tested.

Table 7: ERBF convergence stability across five random seeds. Values are mean $\pm$ standard deviation over seeds.

Dataset	R^2 (mean $\pm$ std)	Gap (mean $\pm$ std)
`california_housing`	0.769 ± 0.008	0.037 ± 0.003
`concrete_strength`	0.850 ± 0.012	0.047 ± 0.010
`esol`	0.910 ± 0.004	0.017 ± 0.003
`power_plant`	0.938 ± 0.001	0.010 ± 0.001
`superconduct`	0.853 ± 0.005	0.023 ± 0.006

D.2 Chebyshev basis conditioning

The default interaction terms are raw products $x_i x_j$ of the scaled features rather than Chebyshev cross-terms. The orthogonality-preserving alternative replaces each product with the tensor terms $T_a(x_i)\,T_b(x_j)$, which remain pairwise but whose count within each pair grows as the square of the interaction degree. Because $T_1(x) = x$, at the default budget the single product per pair coincides with its lowest cross term, $x_i x_j = T_1(x_i)\,T_1(x_j)$, so the interaction columns form a subset of the orthogonal tensor-product basis and strict orthogonality is preserved; richer curved interactions such as $T_2(x_i)\,T_1(x_j)$ would require the fuller tensor construction, which we leave as a natural extension. Orthogonality departs only at the higher-complexity setting, which appends the non-tensor term $T_2(x_i x_j)$. The main-effect columns retain their favourable conditioning, and the ridge penalty handles the residual ill-conditioning from the interaction columns.

To quantify the conditioning advantage of the Chebyshev basis over monomials, we computed the condition number of the design matrix for 21 datasets spanning all four strata, under four configurations: Chebyshev main effects, Chebyshev with interactions, monomial main effects, and monomial with interactions. All experiments used degree 5 with the top 10 features selected by mutual information, and datasets with $n > 5{,}000$ were subsampled for SVD tractability. Table 8 reports the ratio of monomial to Chebyshev condition number per dataset.

Table 8: Ratio of monomial to Chebyshev condition number (linear scale) for 21 datasets at degree 5 with top-10 MI-selected features. Values >1 indicate Chebyshev advantage; values <1 indicate monomial advantage. Datasets ordered by stratum.

Dataset	Stratum	d	Main effects	+Interactions
concrete_strength	S1	8	15.5	127.9
power_plant	S1	4	3.7	24.3
airfoil_noise	S1	5	6.7	3.7
energy_efficiency_heating	S1	8	0.9	12.2
friedman1	S1	5	12.6	1.7
cpu_act	S1	10	0.7	0.5
elevators	S1	10	9.8	1.3
pmlb_225_puma8NH	S1	8	3.1	3.9
abalone	S3	7	4.6	4.5
esol	S3	7	29.1	1.7
lipophilicity	S3	10	5.2	0.6
qsar_fish_toxicity	S3	6	1.8	1.3
superconduct	S3	10	198.4	1.1
sulfur	S3	6	45.2	4.0
wine_quality	S2	10	2.4	0.3
analcatdata_supreme	S2	7	0.5	0.1
pol	S2	10	38.1	24.2
california_housing	S4	8	6.1	0.1
diamonds	S4	6	0.1	1.1
house_sales	S4	10	24.0	1.6
MiamiHousing2016	S4	10	42.8	3.7

For main effects, the Chebyshev basis is better-conditioned on 17 of 21 datasets (81%), with a geometric mean ratio of 6.2×. Four datasets show monomial advantage (diamonds, cpu_act, energy_efficiency_heating, analcatdata_supreme); for three the advantage is modest (ratios 0.5–0.9), while diamonds favours the monomial basis more strongly (0.1). When interaction terms are added, the advantage narrows but remains: Chebyshev is better-conditioned on 16 of 21 datasets (76%), with a geometric mean ratio of 2.2×. The interaction columns (raw products $x_i x_j$) are identical in the Chebyshev and monomial design matrices; being common to both, they bring the two condition numbers closer together and dilute the ratio, whose advantage derives from the differing main-effect columns. The narrowing therefore reflects these shared columns rather than a loss of orthogonality at the default budget (section 3.2); the ridge penalty compensates for the residual ill-conditioning in practice.

D.3 ERBF initialisation sensitivity

To verify that ERBF's downstream performance is robust to the choice of initialisation strategy, we evaluated all four combinations of centre placement (lipschitz, kmeans) and width initialisation (local_ridge, local_variance) across five representative datasets, using three random seeds per configuration. Table 9 reports the mean test R^2 for each combination. The maximum R^2 range across configurations is 0.023 (california_housing), and every dataset shows a range below 0.025. The choice of centre placement and width initialisation strategy has only a modest effect on downstream performance, indicating that the three-stage pipeline is robust to these hyperparameter choices.

Table 9: ERBF mean test R^2 across all four initialisation combinations (2 centre placement strategies $\times$ 2 width initialisation strategies), averaged over 3 random seeds. Lip = Lipschitz-guided centre placement; KM = K-means centre placement; Ridge = local ridge width initialisation; Var = local variance width initialisation. Range = max minus min across the four configurations.

Dataset	Lip/Ridge	Lip/Var	KM/Ridge	KM/Var	Range
california_housing	0.768	0.763	0.745	0.751	0.023
concrete_strength	0.849	0.850	0.871	0.849	0.022
esol	0.908	0.896	0.900	0.898	0.012
power_plant	0.938	0.938	0.938	0.940	0.002
superconduct	0.855	0.847	0.833	0.837	0.022

E Dataset Catalogue

Table 10: All 55 benchmark datasets grouped by stratum. n = samples, d = features, n' and d' = effective counts after preprocessing (shown only where they differ from n/d). For the eight datasets that undergo per-fold feature selection, d' is the median effective feature count across the five outer folds, as the selected set varies by fold; for all other datasets d' reflects only deterministic quality filtering (removal of high-missingness and quasi-constant features). Datasets with non-continuous targets are marked† (13 total; see table 2). Data URLs are hyperlinked in the electronic version.

Dataset	n	d	n'	d'	Source	Data URL
S1 – Engineering/Simulation (13 datasets)						
airfoil_noise	1 503	5			UCI	uci.edu/dataset/291
Ailerons†	13 750	33		20	OpenML	openml.org/d/44137
concrete_strength	1 030	8			UCI	uci.edu/dataset/165
cpu_act†	8 192	21			OpenML	openml.org/d/44132
elevators†	16 599	16			OpenML	openml.org/d/44134
energy_efficiency_heating	768	8			UCI	uci.edu/dataset/242
feynman_gaussian	8 000	2			HF	srsd-feynman_hard
feynman_wave_interference	8 000	4			HF	srsd-feynman_hard
friedman1	2 000	5			sklearn	make_friedman1
friedman1_d100	2 000	100		19	sklearn	make_friedman1
pmlb_215_2dplanes	40 768	10			PMLB	215_2dplanes
pmlb_225_puma8NH	8 192	8			PMLB	225_puma8NH
power_plant	9 568	4			UCI	uci.edu/dataset/294
S2 – Behavioural/Social (10 datasets)						
analcatdata_supreme†	4 052	7		5	OpenML	openml.org/d/504
Bike_Sharing_Demand	17 379	6			OpenML	openml.org/d/44142
food_delivery_time†	45 451	9			OpenML	openml.org/d/46928
pmlb_1028_SWD†	1 000	10			PMLB	1028_SWD
pmlb_1029_LEV†	1 000	4			PMLB	1029_LEV
pmlb_1030_ERA†	1 000	4			PMLB	1030_ERA
pmlb_4544_GeoMusic	1 059	117		23	PMLB	4544_Geo...Music
pol†	15 000	26		2	OpenML	openml.org/d/44133
student_performance†	649	30		20	UCI	uci.edu/dataset/320
wine_quality†	6 497	11			OpenML	openml.org/d/44136
S3 – Physics/Chemistry/Life Sciences (16 datasets)						
abalone†	4 177	7			HF	tabular-benchmark

Continued on next page

Dataset	n	d	n'	d'	Source	Data URL
diabetes	442	10			sklearn	load_diabetes
esol	1 128	7			MolNet	moleculenet.org
freesolv	642	16		13	MolNet	moleculenet.org
lipophilicity	4 200	16		13	MolNet	moleculenet.org
particulate-matter	394 299	9	50K		OpenML	openml.org/d/42207
physiochem_protein	45 730	9			OpenML	openml.org/d/46949
pmlb_503_wind	6 574	14			PMLB	503_wind
pmlb_522_pm10	500	7			PMLB	522_pm10
pmlb_529_pollen	3 848	4			PMLB	529_pollen
pmlb_547_no2	500	7			PMLB	547_no2
qm7	6 834	16		14	MolNet	moleculenet.org
qsar_fish_toxicity	907	6			OpenML	openml.org/d/46954
qsar_tid_11	5 742	1 024		28	OpenML	openml.org/d/46953
sulfur	10 081	6			OpenML	openml.org/d/44145
superconduct	21 263	79		50	OpenML	openml.org/d/44148
S4 – Economic/Pricing (16 datasets)						
Allstate_Claims	188 318	130	50K	50	OpenML	openml.org/d/42571
Brazilian_houses	10 692	8			OpenML	openml.org/d/44141
california_housing	20 640	8			sklearn	fetch_california_housing
diamonds	53 940	6	50K		OpenML	openml.org/d/44140
fiat_500_price	1 538	7			OpenML	openml.org/d/46907
healthcare_insurance	1 338	6			OpenML	openml.org/d/46931
house_16H	22 784	16			OpenML	openml.org/d/44139
house_sales	21 613	15		14	OpenML	openml.org/d/44144
medical_charges	163 065	3	50K		OpenML	openml.org/d/44146
MiamiHousing2016	13 932	13			OpenML	openml.org/d/44147
nyc-taxi[†]	581 835	9	50K	8	OpenML	openml.org/d/44143
pmlb_218_house_8L	22 784	8			PMLB	218_house_8L
power_grid_stability	10 000	12			UCI	uci.edu/dataset/471
synthetic_multithreshold	750	6			synth.	this study [a]
synthetic_piecewise	2 000	5			synth.	this study [b]
synthetic_step	2 000	8			synth.	this study [c]

[†] Non-continuous target (ordinal, integer-valued, or discrete-like); see table 2 for details. Treated as continuous regression throughout. All synthetic targets include additive Gaussian noise ε.
[a] $y = 3\mathbf{1}[x_0 > 0] + 2\mathbf{1}[x_1 > 0.5] + 1.5\mathbf{1}[x_2 < -0.3] + \mathbf{1}[|x_3| < 1] + 0.5\,\mathbf{1}[x_0 > 0]\mathbf{1}[x_1 > 0] + \varepsilon$
[b] $y = 2[x_0]_+ + 1.5[-x_1]_+ + [x_2 - 0.5]_+ - [x_0 + x_1]_+ + \varepsilon$
[c] $y = 2\mathbf{1}[x_0 > 0] + 3\mathbf{1}[x_1 > 0.5] - 1.5\mathbf{1}[x_2 < -0.5] + \mathbf{1}[x_0 > 0]\mathbf{1}[x_1 > 0] + \varepsilon$

Source key. UCI = UCI Machine Learning Repository; OpenML = OpenML repository; PMLB = Penn Machine Learning Benchmarks; MolNet = MoleculeNet; HF = Hugging Face Datasets; sklearn = scikit-learn `datasets` module; synth. = generated by this study. All Data URL entries are hyperlinked to the specific dataset version used.

F Supplementary Results

F.1 Predictive accuracy rankings

Table 11: Predictive accuracy–all models (55 datasets, with `tabpfn`). Models ordered by mean rank on $\bar{R}^2$. Top-2 = rank-1 / rank-2 counts. Best in **bold**, second underlined.

Model	Top-2	Mean Rank	Rank IQR	$\bar{R}^2$ median	$\bar{R}^2$ mean $\pm$ std
tabpfn	**41** / 6	**1.71**	1–1	**0.836**	**0.761 ± 0.231**
erbf	4 / 8	4.05	3–6	0.791	0.710 ± 0.245
chebytree	2 / 5	4.27	3–5	0.785	0.700 ± 0.251
xgb	1 / 14	4.53	2–6	0.771	0.704 ± 0.245
chebypoly	1 / 7	4.85	4–6	0.764	0.678 ± 0.247
ebm	3 / 5	5.05	4–6	0.748	0.672 ± 0.242
rf	3 / 6	5.16	4–7	0.739	0.691 ± 0.239
dt	0 / 1	7.42	7–9	0.706	0.643 ± 0.256
ridge	0 / 3	7.95	8–9	0.579	0.554 ± 0.242

Table 12: Predictive accuracy–CPU-viable models (55 datasets, without `tabpfn`). Models ranked by mean rank on $\bar{R}^2$. Top-2 = rank-1 / rank-2 counts. Best in **bold**, second underlined.

Model	Top-2	Mean Rank	Rank IQR	$\bar{R}^2$ median	$\bar{R}^2$ mean $\pm$ std
erbf	12 / 13	**3.18**	2–5	**0.791**	**0.710 ± 0.245**
chebytree	6 / 12	3.40	2–4	0.785	0.700 ± 0.251
xgb	**15** / 8	3.56	1–5	0.771	0.704 ± 0.245
chebypoly	6 / 6	3.96	3–5	0.764	0.678 ± 0.247
ebm	6 / 7	4.16	3–5	0.748	0.672 ± 0.242
rf	8 / 5	4.25	3–6	0.739	0.691 ± 0.239
dt	0 / 2	6.49	6–8	0.706	0.643 ± 0.256
ridge	2 / 2	6.98	7–8	0.579	0.554 ± 0.242

F.2 Generalisation gap rankings

Table 13: Generalisation gap rankings (55 datasets, all nine models). Lower gap = better generalisation. Models ordered by mean gap rank. Top-2 = rank-1 / rank-2 counts. Best in **bold**, second underlined (among competitive models; `ridge`'s low gap reflects underfitting).

Model	Top-2	Mean Rank	Rank IQR	Gap median	Gap mean $\pm$ std
ridge	40 / 7	1.82	1–2	0.003	0.011 ± 0.016
chebypoly	2 / **16**	**3.47**	2–4	**0.014**	**0.029 ± 0.040**
ebm	2 / 6	4.40	3–6	0.020	0.035 ± 0.044
erbf	2 / 6	4.47	3–6	0.018	0.039 ± 0.054
chebytree	2 / 7	4.82	3–6	0.029	0.040 ± 0.053
rf	2 / 4	5.80	4–7	0.024	0.062 ± 0.097
dt	1 / 6	5.84	4–7	0.032	0.048 ± 0.050
tabpfn	3 / 3	6.02	4–8	0.044	0.069 ± 0.086
xgb	1 / 0	8.31	8–9	0.085	0.107 ± 0.107

F.3 Sensitivity to the matched-accuracy threshold

The matched-accuracy gap analysis (section 4.2) uses a threshold of $|\Delta\bar{R}^2| \leq 0.02$ to define "comparable accuracy." Table 14 shows that the smooth-model gap advantage is stable across thresholds spanning an order of magnitude, from 0.005 to 0.10. The proportion of smooth-model wins on gap remains between 83% and 87% regardless of the threshold chosen, while the number of qualifying pairs increases from 67 at the tightest threshold to 307 at the widest. This stability indicates that the 0.02 threshold is not cherry-picked; the finding is robust to the particular definition of "matched accuracy."

Table 14: Sensitivity of the matched-accuracy gap analysis to the accuracy-equivalence threshold $|\Delta\bar{R}^2|$. "Pairs" is the number of smooth–tree model-dataset pairs meeting the threshold; "Smooth wins gap" is the percentage where the smooth model has the smaller generalisation gap.

Threshold	Pairs	Smooth wins gap (%)
0.005	67	86.6
0.010	108	86.1
0.020	167	86.8
0.050	252	83.7
0.100	307	82.7

F.4 Computational cost details

Table 15: Computational cost (mean across 55 datasets). Tune time includes all Optuna trials for one outer fold. Predict time is per 1 000 instances. Train/1K is train time normalised per 1 000 instances, averaged across datasets; actual scaling is model-dependent. Models ordered by tune time. Fastest CPU-viable competitive model in **bold**, second underlined (tune time).

Model	Tune (s)	Train (s)	Train (ms/1K)	Predict (ms/1K)
tabpfn	2	43.47	3 000	3 609
ridge	20	0.01	0	0
dt	30	0.26	0	1
chebypoly	**40**	0.34	0	12
chebytree	61	0.47	0	20
xgb	182	0.99	0	9
ebm	391	9.43	2	2
erbf	777	16.98	1	9
rf	1 343	41.64	4	157

Each tuning trial involves inner cross-validation (3-fold for $n \geq 1000$, 5-fold otherwise), so per-trial time exceeds single-training time. Table 16 illustrates per-trial and training times on three datasets spanning the size range.

Table 16: Time per tuning trial and training time (seconds) on three datasets spanning the size range (CPU models only; `tabpfn` requires no per-trial tuning). Trial counts: `ridge` 20, `dt` 25, `rf` 25, `xgb` 50, `ebm` 15, `erbf` 30, `chebypoly` 30, `chebytree` 30.

Dataset		n	d	**ridge**	**dt**	**rf**	**xgb**	**ebm**	**erbf**	**chebypoly**	**chebytree**
`concrete_strength`	/trial	1 030	8	0.3	0.2	10.2	1.2	16.7	2.2	0.2	0.8
	train			0.0	0.0	6.5	0.5	5.9	1.0	0.0	0.1
`california_housing`	/trial	20 640	8	0.3	0.8	92.4	4.9	33.5	25.6	0.7	2.4
	train			0.0	0.4	48.3	1.2	12.3	18.1	0.2	1.1
`superconduct`	/trial	21 263	79	1.7	4.1	312.6	27.0	175.0	83.2	4.5	7.1
	train			0.1	2.2	217.8	8.8	61.4	51.9	2.6	3.9

F.5 Distribution of winning configurations

Table 17 summarises the hyperparameter values selected by Optuna across the 55 datasets, reported as medians with ranges across outer folds. These distributions characterise the configurations that the tuning procedure converges to and may serve as informed defaults for practitioners.

Table 17: Distribution of winning configurations across 55 datasets. *Tuned hyperparameters* are selected by Optuna; *fitted properties* are emergent characteristics of the resulting model. Numeric values are median (IQR) of per-dataset fold-averaged best parameters. Categorical values show the majority choice with its frequency. Search spaces are given in table 5.

Model	Type	Parameter	Median (IQR)
ridge	HP	alpha	5.8 (1.2, 18.3)
dt	HP	max_depth	12 (10, 13)
	HP	min_samples_leaf	0.007 (0.007, 0.014)
	HP	min_samples_split	0.019 (0.014, 0.035)
	Fit	actual depth	9 (8, 11)
	Fit	leaves	72 (34, 87)
rf	HP	n_estimators	396 (285, 450)
	HP	max_depth	16 (13, 18)
	HP	max_features	0.7 (65%); 0.5 (13%); sqrt (9%)
xgb	HP	max_depth	8 (6, 9)
	HP	learning_rate	0.295 (0.292, 0.297)
	HP	subsample	0.85 (0.80, 0.90)
	HP	colsample_bytree	0.86 (0.78, 0.90)
	HP	reg_lambda	0.012 (0.005, 0.039)
	HP	min_child_weight	6.0 (3.7, 10.9)
	Fit	best iteration (early stop)	25 (17, 51)
erbf	HP	n_rbf	67 (53, 78)
	HP	n_rbf = auto	20% of datasets
	HP	alpha	0.25 (0.03, 4.4)
	HP	center_init	lipschitz (55%); kmeans (45%)
	HP	width_init	local_ridge (65%); local_variance (35%)
	Fit	effective K	60 (40, 79)
	Fit	width parameters	440 (308, 948)
chebypoly	HP	complexity	9 (4, 13)
	HP	alpha	0.69 (0.07, 2.2)
	HP	include_interactions	True: 69%; False: 31%
	HP	max_interaction_complexity	2 (median); 1 or 2
	Fit	total parameters	138 (53, 202)
chebytree	HP	complexity	2 (2, 4)
	HP	max_depth	4 (1, 9)
	HP	min_samples_leaf	0.053 (0.028, 0.066)
	HP	alpha	0.14 (0.009, 0.74)
	Fit	leaves	8 (2, 20)
	Fit	parameters per leaf	23 (14, 41)
	Fit	total parameters	198 (90, 491)
ebm	HP	max_bins	256 (64%); 128 (36%)
	HP	learning_rate	0.052 (0.030, 0.074)

Continued on next page

Model	Type	Parameter	Median (IQR)
	HP	`min_samples_leaf`	4 (78%); 10 (13%); 2 (9%)

HP = tuned hyperparameter (selected by Optuna); Fit = fitted model property (emergent, not directly tuned). Categorical entries show mode with percentage; numeric entries show median (Q1, Q3) across 55 datasets.

Several patterns emerge. `erbf` favours Lipschitz-guided centre placement (55% of datasets) and supervised width initialisation via local ridge (65%), suggesting that target-aware initialisation tends to improve over purely geometric alternatives. Fixed centre counts are preferred (80% of datasets) with a median of 67 (IQR 53–78); the `auto` rule is selected for 20% of datasets (typically higher-dimensional ones), yielding a median of 60 effective centres. `chebypoly` converges to high polynomial degrees (median 9, IQR 4–13) with interactions enabled in 69% of datasets, suggesting that the ridge penalty is sufficient to control the resulting high-dimensional basis. `chebytree` uses moderate tree depth (median 4, yielding $\sim$8 leaves) with low-degree leaf polynomials (median 2), consistent with the design principle that leaf models should be simpler than a global fit to avoid overfitting small subsets. `xgb` selects high learning rates (median 0.295) but early stopping terminates at a median of 25 boosting rounds (IQR 17–51, out of 2 000 maximum), indicating that relatively few trees suffice when the learning rate is high. `rf` favours large ensembles (median 396 trees) with high feature subsampling (0.7 in 65% of datasets), consistent with known good practice for random forests.

F.6 Per-dataset results

Table 18 presents the adjusted R^2 for each model on each of the 55 benchmark datasets. The best CPU-viable model per dataset is shown in **bold**. Table 19 gives the corresponding per-dataset ranks over the eight CPU-viable models (rank 1 = best on each dataset); these average to the mean ranks reported in section 4.1.

Table 18: Per-dataset adjusted R^2 for all nine models, grouped by stratum. Overall best per dataset in **bold**; best CPU-viable model underlined (shown only when different from overall best). †Ordinal (discrete integer) target.

Dataset	tabpfn	erbf	chebytree	xgb	ebm	chebypoly	rf	dt	ridge
S1: Engineering/Simulation									
Ailerons	**0.793**	0.760	0.779	0.771	0.765	0.781	0.756	0.719	0.756
airfoil_noise	**0.977**	0.921	0.831	0.923	0.557	0.728	0.828	0.725	0.503
concrete_strength	**0.945**	0.885	0.865	0.917	0.907	0.892	0.880	0.809	0.597
cpu_act	**0.988**	0.983	0.982	0.982	0.981	0.983	0.976	0.968	0.736
elevators	**0.929**	0.862	0.886	0.873	0.873	0.890	0.689	0.594	0.812
energy_efficiency_heating	0.998	**0.998**	0.997	0.996	0.989	0.997	0.995	0.996	0.913
feynman_gaussian	**0.999**	0.969	0.991	0.490	0.500	0.381	0.945	0.939	0.068
feynman_wave_interference	**0.999**	0.965	0.883	0.935	0.846	0.909	0.888	0.873	0.841
friedman1	**0.999**	0.998	0.948	0.961	0.920	0.923	0.883	0.751	0.741
friedman1_d100	**0.286**	0.261	0.256	0.243	0.253	0.257	0.262	0.230	0.215
pmlb_215_2dplanes	0.948	0.948	**0.948**	0.947	0.705	0.948	0.947	0.948	0.705
pmlb_225_puma8NH	**0.687**	0.686	0.671	0.656	0.430	0.431	0.676	0.649	0.368
power_plant	**0.967**	0.948	0.944	0.961	0.952	0.941	0.947	0.939	0.928
S2: Behavioural/Social									
Bike_Sharing_Demand	**0.718**	0.671	0.682	0.689	0.631	0.644	0.668	0.654	0.333
analcatdata_supreme	0.977	0.977	0.978	0.978	**0.978**	0.978	0.972	0.978	0.429
food_delivery_time	—	0.252	0.335	**0.346**	0.336	0.331	0.284	0.278	0.181
pmlb_1028_SWD†	0.412	0.405	0.412	0.387	**0.415**	0.409	0.409	0.363	0.388
pmlb_1029_LEV†	0.544	0.548	0.538	0.501	0.545	**0.551**	0.524	0.478	0.551
pmlb_1030_ERA†	0.352	0.344	**0.365**	0.325	0.356	0.340	0.342	0.334	0.351
pmlb_4544_GeographicalOrigi...	**0.641**	0.586	0.574	0.570	0.587	0.605	0.599	0.507	0.617
pol†	0.164	**0.169**	0.168	0.161	0.163	0.166	0.168	0.167	0.133
student_performance†	0.196	0.161	0.170	0.195	0.193	0.178	**0.208**	0.144	0.186
wine_quality†	**0.514**	0.387	0.335	0.439	0.353	0.363	0.391	0.301	0.287
S3: Physics/Chemistry/Life Sciences									
abalone	**0.586**	0.556	0.564	0.521	0.526	0.559	0.548	0.481	0.514
diabetes	**0.473**	0.432	0.460	0.409	0.455	0.464	0.435	0.296	0.464
esol	**0.916**	0.896	0.847	0.881	0.858	0.893	0.894	0.852	0.827

Continued on next page

Dataset	tabpfn	erbf	chebytree	xgb	ebm	chebypoly	rf	dt	ridge
freesolv	**0.935**	0.885	0.884	0.882	0.898	0.886	0.877	0.782	0.823
lipophilicity	**0.592**	0.387	0.329	0.390	0.343	0.331	0.397	0.293	0.247
particulate-matter-ukair-2017	0.744	0.740	0.746	0.741	**0.752**	0.750	0.726	0.718	0.733
physiochemical_protein	**0.768**	0.505	0.505	0.615	0.378	0.404	0.424	0.359	0.281
pmlb_503_wind	**0.815**	0.791	0.785	0.768	0.784	0.799	0.760	0.706	0.758
pmlb_522_pm10	0.398	**0.398**	0.094	0.296	0.377	0.349	0.365	0.132	0.123
pmlb_529_pollen	**0.794**	0.791	0.792	0.740	0.782	0.791	0.710	0.639	0.792
pmlb_547_no2	**0.600**	0.588	0.543	0.519	0.555	0.539	0.599	0.469	0.476
qm7	**0.810**	0.785	0.768	0.779	0.771	0.773	0.771	0.737	0.746
qsar_fish_toxicity	**0.639**	0.611	0.564	0.552	0.584	0.559	0.615	0.498	0.556
qsar_tid_11	**0.520**	0.426	0.432	0.484	0.251	0.350	0.376	0.327	0.251
sulfur	**0.920**	0.779	0.791	0.828	0.748	0.764	0.731	0.716	0.393
superconduct	**0.931**	0.848	0.892	0.915	0.887	0.819	0.856	0.817	0.674
S4: Economic/Pricing									
Allstate_Claims_Severity	**0.526**	0.482	0.486	0.480	0.457	0.462	0.430	0.391	0.447
Brazilian_houses	**0.996**	0.972	0.982	0.987	0.978	0.966	0.979	0.973	0.836
MiamiHousing2016	**0.944**	0.918	0.897	0.917	0.895	0.909	0.864	0.802	0.717
california_housing	**0.875**	0.792	0.751	0.829	0.772	0.730	0.739	0.667	0.601
diamonds	**0.947**	0.945	0.943	0.945	0.945	0.943	0.944	0.942	0.925
fiat_500_price	**0.857**	0.841	0.844	0.834	0.850	0.846	0.851	0.823	0.840
healthcare_insurance	**0.856**	0.839	0.856	0.845	0.737	0.833	0.855	0.840	0.733
house_16H	**0.583**	0.491	0.502	0.514	0.515	0.508	0.485	0.422	0.235
house_sales	**0.903**	0.870	0.852	0.876	0.847	0.846	0.838	0.790	0.746
medical_charges	0.980	**0.980**	0.980	0.979	0.980	0.980	0.977	0.978	0.827
nyc-taxi-green-dec-2016	**0.578**	0.459	0.530	0.513	0.568	0.393	0.440	0.517	0.307
pmlb_218_house_8L	**0.724**	0.661	0.657	0.672	0.619	0.638	0.621	0.575	0.380
power_grid_stability	**0.987**	0.951	0.896	0.926	0.784	0.895	0.815	0.689	0.645
synthetic_multithreshold	0.965	0.888	0.931	0.959	0.963	0.858	**0.966**	0.958	0.569
synthetic_piecewise	**0.963**	0.960	0.952	0.935	0.920	0.955	0.935	0.902	0.791
synthetic_step	0.939	0.877	0.932	0.922	0.923	0.844	**0.939**	0.938	0.579

Table 19: Per-dataset rank of the eight CPU-viable models by adjusted R^2 (rank 1 = best on that dataset; ties share the lower rank). Best per dataset in **bold**. †Ordinal (discrete integer) target. TabPFN is excluded (analysed separately). Mean ranks per model are reported in the main text.

Dataset	erbf	chebytree	xgb	ebm	chebypoly	rf	dt	ridge
S1: Engineering/Simulation								
Ailerons	5	2	3	4	**1**	7	8	6
airfoil_noise	2	3	**1**	7	5	4	6	8
concrete_strength	4	6	**1**	2	3	5	7	8
cpu_act	2	3	4	5	**1**	6	7	8
elevators	5	2	3	4	**1**	7	8	6
energy_efficiency_heating	**1**	2	4	7	3	6	5	8
feynman_gaussian	2	**1**	6	5	7	3	4	8
feynman_wave_interference	**1**	5	2	7	3	4	6	8
friedman1	**1**	3	2	5	4	6	7	8
friedman1_d100	2	4	6	5	3	**1**	7	8
pmlb_215_2dplanes	4	**1**	5	8	3	6	2	7
pmlb_225_puma8NH	**1**	3	4	7	6	2	5	8
power_plant	3	5	**1**	2	6	4	7	8
S2: Behavioural/Social								
Bike_Sharing_Demand	3	2	**1**	7	6	4	5	8
analcatdata_supreme	6	4	5	**1**	2	7	3	8
food_delivery_time	7	3	**1**	2	4	5	6	8
pmlb_1028_SWD†	5	2	7	**1**	4	3	8	6
pmlb_1029_LEV†	3	5	7	4	**1**	6	8	2
pmlb_1030_ERA†	4	**1**	8	2	6	5	7	3
pmlb_4544_GeographicalOrigi...	5	6	7	4	2	3	8	**1**
pol†	**1**	3	7	6	5	2	4	8
student_performance†	7	6	2	3	5	**1**	8	4
wine_quality†	3	6	**1**	5	4	2	7	8
S3: Physics/Chemistry/Life Sciences								
abalone	3	**1**	6	5	2	4	8	7
diabetes	6	3	7	4	**1**	5	8	2
esol	**1**	7	4	5	3	2	6	8

Continued on next page

Dataset	erbf	chebytree	xgb	ebm	chebypoly	rf	dt	ridge
freesolv	3	4	5	**1**	2	6	8	7
lipophilicity	3	6	2	4	5	**1**	7	8
particulate-matter-ukair-2017	5	3	4	**1**	2	7	8	6
physiochemical_protein	2	3	**1**	6	5	4	7	8
pmlb_503_wind	2	3	5	4	**1**	6	8	7
pmlb_522_pm10	**1**	8	5	2	4	3	6	7
pmlb_529_pollen	4	2	6	5	3	7	8	**1**
pmlb_547_no2	2	4	6	3	5	**1**	8	7
qm7	**1**	6	2	5	3	4	8	7
qsar_fish_toxicity	2	4	7	3	5	**1**	8	6
qsar_tid_11	3	2	**1**	7	5	4	6	8
sulfur	3	2	**1**	5	4	6	7	8
superconduct	5	2	**1**	3	6	4	7	8
S4: Economic/Pricing								
Allstate_Claims_Severity	2	**1**	3	5	4	7	8	6
Brazilian_houses	6	2	**1**	4	7	3	5	8
MiamiHousing2016	**1**	4	2	5	3	6	7	8
california_housing	2	4	**1**	3	6	5	7	8
diamonds	2	5	**1**	3	6	4	7	8
fiat_500_price	5	4	7	2	3	**1**	8	6
healthcare_insurance	5	**1**	3	7	6	2	4	8
house_16H	5	4	2	**1**	3	6	7	8
house_sales	2	3	**1**	4	5	6	7	8
medical_charges	**1**	2	5	4	3	7	6	8
nyc-taxi-green-dec-2016	5	2	4	**1**	7	6	3	8
pmlb_218_house_8L	2	3	**1**	6	4	5	7	8
power_grid_stability	**1**	3	2	6	4	5	7	8
synthetic_multithreshold	6	5	3	2	7	**1**	4	8
synthetic_piecewise	**1**	3	4	6	2	5	7	8
synthetic_step	6	3	5	4	7	**1**	2	8

F.7 ERBF performance on discrete targets

ERBF's mean rank drops from 2.88 on continuous-target datasets to 4.15 on the thirteen datasets with non-continuous targets (and 3.89 on the nine datasets with discrete or ordinal targets shown in table 20). However, with only thirteen non-continuous datasets in our benchmark, this rank difference should be interpreted cautiously: the sample is too small for the shift to be statistically reliable. We present the analysis below as exploratory rather than confirmatory.

Table 20 reports adjusted R^2 for all eight CPU-viable models on the nine datasets with discrete or ordinal targets, ordered by the number of unique target values. ERBF's performance is mixed: it outperforms XGBoost on several datasets (e.g., `pol` $+0.01$, `abalone` $+0.04$, `pmlb_1029_LEV` $+0.05$) but underperforms it on others (`wine_quality` -0.05, `student_performance` -0.03). The pattern does not reduce to a simple threshold on the number of distinct target values, and the small sample precludes identifying a reliable predictor of when ERBF will underperform. Plausible contributing factors include the combination of few target levels, categorical input features (which produce target-encoded representations with limited gradient information), and small sample size; these are confounded across the nine datasets, and separating their individual contributions is a natural direction for future work. For instance, `student_performance` ($n = 649$, 17 unique y, 17 categorical features) is challenging for all models ($R^2 = 0.18$–0.24), with ERBF's smooth surface providing no advantage over tree-based alternatives.

Table 20: Adjusted R^2 on the nine datasets with discrete or ordinal targets, ordered by number of unique target values. ERBF and XGBoost are shown alongside the best-performing model among the remaining six (named in the rightmost column). All results use Optuna-tuned hyperparameters from the main benchmark. `Bike_Sharing_Demand` is an integer count with many distinct values (869); it is the highest-cardinality of the thirteen non-continuous datasets (table 2) and is included here as a contrast to the low-resolution ordinal and count targets.

Dataset	Unique y	n	ERBF	XGB	Best other	Model
`pmlb_1028_SWD`	4	1 000	0.413	0.394	**0.422**	ebm
`pmlb_1029_LEV`	5	1 000	0.550	0.504	**0.554**	chebypoly
`wine_quality`	7	6 497	0.388	**0.440**	0.393	rf
`pmlb_1030_ERA`	9	1 000	0.347	0.328	**0.368**	chebytree
`analcatdata_supreme`	10	4 052	0.977	**0.978**	**0.978**	chebypoly
`pol`	11	15 000	**0.169**	0.161	0.168	rf
`student_performance`	17	649	0.194	0.227	**0.240**	rf
`abalone`	28	4 177	0.557	0.522	**0.565**	chebytree
`Bike_Sharing_Demand`	869	17 379	0.672	**0.689**	0.682	chebytree

To isolate the effect of target discretisation from confounding factors (sample size, categorical features, tuning), we constructed a synthetic experiment. A smooth function ($\sin(2\pi x_1) + \cos(3\pi x_2) + \varepsilon$, $\varepsilon \sim \mathcal{N}(0, 0.01)$, with three noise features) was rounded to varying numbers of equally spaced levels, from 3 to continuous. Only the target is discretised; the inputs and the underlying smooth relationship are held fixed, so any change in accuracy isolates the effect of target rounding alone. Table 21 reports 5-fold cross-validated R^2 for ERBF, XGBoost, and a decision tree at each discretisation level using default hyperparameters.

On this synthetic function – where the underlying process is smooth and ERBF is well suited by construction – all three models degrade gracefully as the number of levels decreases, with ERBF matching or slightly exceeding XGBoost across the entire range. This indicates that, at least for a smooth data-generating process, target discretisation alone does not account for the rank degradation observed on real discrete-target datasets. Combined with the mixed picture

in table 20, this suggests that the real-data degradation may reflect an interaction of multiple factors rather than discreteness alone, though the small number of non-continuous datasets in our benchmark (nine) limits the conclusions that can be drawn.

Table 21: Synthetic experiment: effect of target rounding on model accuracy. A smooth function is discretised to varying numbers of equally spaced levels. Default hyperparameters; 5-fold CV.

Levels	**ERBF** R^2	**XGB** R^2	**DT** R^2
3	0.845	0.849	0.779
5	0.929	0.915	0.850
7	0.954	0.940	0.879
10	0.971	0.959	0.907
15	0.980	0.970	0.921
21	0.984	0.975	0.932
50	0.986	0.978	0.931
100	0.987	0.978	0.930
Continuous	0.987	0.978	0.931