TABNSM: NEURAL SPARSE MIXER FOR TABULAR REGRESSION

Anonymous authors
Paper under double-blind review

ABSTRACT

Modern prediction tables increasingly contain hundreds of columns derived from text or large-language-model embeddings, yet only a few matter for any given record. Existing neural models often compare every pair of columns, which becomes costly as tables widen, and models that condition on the training set can exceed practical memory limits. We introduce *TabNSM*, a neural model for regression on these wide tables. Its core *Adaptive Sparse Interaction Module* selects a small, record-specific set of relevant column groups, models their interactions, and combines them with broader table information, so that computation and memory grow linearly rather than quadratically with the number of columns. Across seven highly redundant real-world datasets (266–405 columns, up to one million rows), TabNSM attains the lowest prediction error on five and a significantly better average rank than each of 41 tuned and pretrained baselines (rank 1.43 of 42, $p \leq 0.016$ against every baseline), with a 21% error reduction on the most interaction-rich table. On the largest tables it trains within a modest GPU-memory budget that does not grow with dataset size, where models that condition on the training set run out of memory or fall back to a small fixed context; because TabNSM is trained rather than conditioned, its feature attributions can also be validated by removing and retraining on the top-ranked features, which sharply degrades accuracy, unlike attributions from in-context models. An ordinal binning loss (GridLoss) and error-aware sampling (RISE) are secondary training additions. TabNSM is built for the wide, redundant tables that embedding pipelines increasingly produce; tuned trees and in-context models remain ahead on low-dimensional tables and on TabArena.

1 INTRODUCTION

Modern applications in healthcare, finance, and industrial monitoring depend on large-scale tabular regression to forecast continuous outcomes, such as clinical risk scores or sensor-based targets. These tasks involve high-dimensional, heterogeneous feature spaces comprising numerical signals, categorical attributes, and text-derived embeddings [1; 2]. Predictive signal is rarely spread evenly across such a space: a small subset of features may be decision-relevant for each instance, while the majority are noisy, redundant, or weakly informative [3]. This contrasts with image and text modalities, where informative content forms coherent spatial or sequential regions.

This motivates a foreground–background view of tabular regression: for each sample, a small *foreground* of decision-relevant features sits within a much larger *background* of uninformative ones, and the partition shifts across samples, so no single feature subset characterizes the dataset. Tree ensembles such as CatBoost approximate this through recursive, axis-aligned feature selection [4; 5; 6; 7], a prior well aligned with tabular sparsity [8], but are less suited to end-to-end representation learning and joint optimization with high-dimensional learned embeddings. Deep tabular models offer this flexibility, but dense self-attention over feature tokens [9; 10] costs $\mathcal{O}(D^2)$, while state-space models [11; 12; 13] scan features in linear time but rely on an imposed ordering and provide no sparse, instance-adaptive selection. In-context tabular foundation models [14; 15; 16; 17; 18] are strong on small benchmarks, but condition a frozen network on the training set, tying them to context-size and GPU-memory limits on long, wide tables.

We propose **Tabular Neural Sparse Mixer (TabNSM)**, a scalable architecture for wide, heterogeneous tabular regression built around the *Adaptive Sparse Interaction Module (ASIM)*. ASIM

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

combines instance-adaptive foreground discovery, sparse local interaction encoding, and residual feature mixing. By selecting structured subsets of feature interactions, it provides a differentiable analogue of tree-like selective feature isolation while preserving end-to-end trainability; the residual mixer propagates information beyond the sparse attention support to balance selective interaction with broader feature context. Under a fixed sparse configuration and token dimension, compute and activation memory grow linearly in D (Proposition 3.1).

The sparse-mixing architecture is the central contribution. TabNSM also uses two training refinements for continuous targets: *GridLoss*, which aligns predictions and targets in an ordinal grid space as a smooth complement to pointwise regression losses, and *RISE* (Reweighted Instance Sampling by Error), which periodically reweights training examples by per-sample loss. We treat these as secondary and quantify their contributions through matched ablations (Section 4.2).

Two practical properties follow from the design. *Memory:* TabNSM is trained with mini-batches and interaction cost linear in D, enabling long tables with up to about a million rows in a small GPU-memory budget, including settings where in-context models exceed available memory, are limited to a fixed training-set size, or must retrieve a small context per query (Section 4.3). *Interpretability:* gradient$\times$input feature attributions from the trained model are validated by remove-and-retrain and randomization tests: removing the top-ranked features and retraining sharply degrades accuracy, whereas removing the lowest-ranked ones does not (Section 4.4).

In summary, we contribute (i) ASIM, an instance-adaptive module integrating foreground discovery, sparse local interaction encoding, and residual mixing, with a multi-stage residual regression head; (ii) GridLoss, a differentiable soft-binning objective with bounded, Lipschitz gradients and an MAE-consistent limit, and RISE, difficulty-aware sampling by loss quantile, each quantified by a matched ablation; (iii) a memory-efficient design that scales linearly with the feature count and trains on wide, large tables; and (iv) feature attributions validated through remove-and-retrain and randomization tests. We evaluate TabNSM on seven real-world regression datasets with more than 250 features (266–405) and 4k–1M rows, achieving the lowest RMSE on five of seven datasets and a significantly better average rank than each of 41 baselines, including in-context foundation models. We further provide GPU-memory and matched-hardware cost comparisons. On low-dimensional datasets and the standardized TabArena benchmark [19], whose tables are small and far less redundant, tuned gradient-boosted trees and in-context models remain competitive (Section 5 and Appendices D.3 and E.1). These results characterize TabNSM's primary use case as wide tabular data with substantial feature redundancy, including settings where LLM-derived embeddings contribute hundreds of columns to production tables.

2 Related Work

Deep Learning versus Tree Ensembles for Regression. GBDTs such as XGBoost [6], LightGBM [5], and CatBoost [4] remain strong and widely used baselines for tabular regression [8; 20]. Our motivation for a neural alternative is primarily representational: neural architectures provide an end-to-end differentiable framework that can jointly learn representations and predictions, including settings involving learned embeddings. Accordingly, the complexity analysis in Section 3 focuses on scaling with the feature dimension D relative to dense-attention neural architectures, which are the relevant comparators for our design. Computational cost and additional comparisons with tree-based methods are reported in Appendix J.

Deep tabular models have progressed beyond standard MLPs to include regularization-focused architectures [21], logic-aware networks [22], and differentiable tree-inspired or hybrid tree-neural models [23; 24]. Recent tabular foundation models (TFM), notably TabPFN [25], TabPFNv2 [14], TabICL [15], TabDPT [17], and TabFM [18], achieve strong generalization through in-context learning and now lead standardized leaderboards such as TabArena [19]. However, their transformer-based designs can incur substantial computational and memory costs due to attention over samples and/or features, often scaling quadratically in the relevant dimension. In addition, inference may require conditioning on training examples, which can limit scalability on large datasets. As a result, practical use of such models is often constrained by dataset size, feature dimensionality, and memory budget [26; 27; 28].

Feature Interaction and Scalability. Transformer-based tabular models, such as TabTransformer [29] and FT-Transformer [9], enable end-to-end representation learning by using dense self-attention to capture global feature interactions. However, their dense attention mechanisms can be sensitive to feature noise in high-dimensional settings [30; 9]. While expressive, dense attention incurs $\mathcal{O}(D^2)$ computational and memory costs with respect to feature dimension D, which becomes expensive for wide tables with hundreds of features. Recent sparse attention variants [3; 31] reduce this burden through block-diagonal masks, hierarchical patterns, or selective attention mechanisms. Concurrently, State-Space Models (SSMs), such as Mamba [32], provide linear-time sequence mixing along the feature axis through selective scan operations, improving scalability and parameter efficiency [33]. However, SSM-based tabular models typically rely on imposed feature ordering and do not explicitly model sparse pairwise feature interactions.

Heterogeneous Data, Text Features, and Regression Objectives. Modern tabular datasets increasingly include text-derived embeddings alongside structured fields [34; 35; 36], and pretrained sentence encoders provide useful representations for such columns [14; 37]. In-context models such as TabICL [15] and MITRA [38], and Hopfield-based BiSHop [39], are constrained by feature count or sample size on wide, large tables. On the objective side, robust losses such as Huber and quantile losses resist outliers [40], and distribution-aware or histogram-based objectives discretize continuous targets to improve calibration [41; 42]; hard-example mining remains less developed for regression with long-tailed targets.

Positioning of TabNSM. GBDTs provide strong feature selectivity but lack differentiable representation learning; dense-attention models learn flexible interactions but scale poorly with feature dimension; and SSM-based models improve efficiency but do not impose sparse, instance-adaptive interaction structure [43; 44]. TabNSM's ASIM makes foreground-selective sparse interaction modeling the core architectural bias rather than a computational shortcut, with GridLoss and RISE as auxiliary training choices whose measured contribution is smaller and dataset-dependent (Section 4.2). We position the empirical claims carefully: on standardized benchmarks such as TabArena [19], whose datasets are small and low-redundancy, tuned GBDTs and in-context foundation models remain very strong and we do not claim to surpass them there (Appendix E.1); TabNSM is designed for, and evaluated on, wide tables with highly redundant, often text-embedding-derived, features, the regime that motivates the sparse-interaction prior in the first place.

3 METHODOLOGY

3.1 PROBLEM FORMULATION AND PREPROCESSING

Let $\{(x_i, y_i)\}_{i=1}^{N}$ be a supervised regression dataset where $x_i \in \mathbb{R}^D$ represents a sample of D heterogeneous features and $y_i \in \mathbb{R}$ is the continuous target. Our objective is to learn a mapping $f_\Theta : \mathbb{R}^D \to \mathbb{R}$ that minimizes a robust regression objective while remaining scalable as D grows. The regime studied empirically in this paper spans tens to several hundred features ($250 \leq D$) on the main suite, with additional experiments at lower D reported in the Appendix D.3 for completeness.

Following standard practice [45], numerical features are standardized, low-cardinality categoricals (fewer than 10 unique values) are integer-encoded, and all transformation parameters are fitted on the training split only. High-cardinality and free-text fields are concatenated per row and encoded with a frozen `all-MiniLM-L6-v2` sentence encoder [46; 47; 48; 2] using masked mean pooling over last-layer states followed by ℓ_2 normalization, yielding a fixed 384-dimensional vector E_i; the input x_i concatenates numerical, categorical, and text features. Larger encoders are compatible with this pipeline [35].

3.2 ARCHITECTURE OVERVIEW

The proposed network consists of two major components: (i) an *Adaptive Sparse Interaction Module (ASIM)* that operates in a tokenized feature space and combines foreground feature discovery, sparse local interaction encoding, and residual feature mixing; and (ii) a *Multi-Stage Regression Head* that applies residually coupled MLP branches followed by a compact fusion head. Figure 1 summarizes the overall architecture.

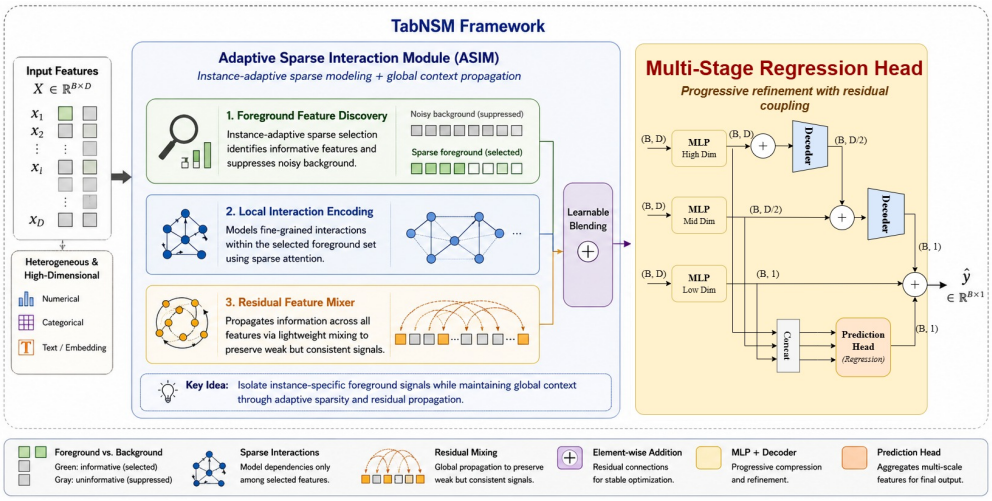


Figure 1: **Overview of the TabNSM architecture.** TabNSM consists of an **Adaptive Sparse Interaction Module (ASIM)** and a **Multi-Stage Regression Head**. ASIM models feature interactions through three complementary pathways: *Foreground Feature Discovery*, which identifies instance-specific informative features; *Local Interaction Encoding*, which captures sparse fine-grained dependencies among selected features; and a *Residual Feature Mixer*, which propagates global information through lightweight mixing. The resulting representation is passed to a multi-stage regression head with progressively narrowed MLP branches, residual coupling, and final feature fusion to produce the prediction $\hat{y}$.

3.3 Adaptive Sparse Interaction Module (ASIM)

To model cross-feature interactions at scale, we project each scalar feature into a d_{tok}-dimensional embedding space. Given an input vector $x \in \mathbb{R}^D$, each feature x_d is mapped via a learnable projection to obtain the tokenized representation $X_{\text{tok}} = \text{Tok}(x) \in \mathbb{R}^{D \times d_{\text{tok}}}$, where $X_{\text{tok}}[d, :] \in \mathbb{R}^{d_{\text{tok}}}$ denotes the token vector for the d-th feature. (Batch dimensions are omitted for clarity).

Sparse Feature-wise Attention. We develop a sparse feature-wise attention mechanism that treats the D input features as a sequence, where each feature token serves as a query for the current sample. Foreground feature discovery is implemented through three components: (i) a *local sliding window* of size s that captures local interactions under the chosen feature ordering; (ii) *compressed block representations*, which aggregate blocks of size b with stride r into summary keys using a learnable MLP, whose resulting attention scores identify the most relevant feature groups for each query; and (iii) retrieval of the top-m highest-scoring blocks at full resolution as the *foreground*. Because the attention scores are computed from sample-specific query tokens, the selected blocks vary across inputs, allowing each sample to dynamically isolate its own foreground feature groups:

$$\tilde{X} = \mathcal{A}_{\text{sparse}}(X_{\text{tok}};\, s, b, r, m, H, d_h) \in \mathbb{R}^{D \times d_{\text{tok}}}. \tag{1}$$

Here H is the number of attention heads and d_h is the per-head dimension, with $d_{\text{tok}} = H d_h$. This mechanism enables instance-adaptive focus on informative feature-to-feature dependencies and can improve robustness to background noise when the sparse support excludes irrelevant dimensions (Proposition 3.2; see Appendix K for the full formulation).

Feature-Token Mixing (FTM). To capture global dependencies that might fall outside the sparse attention mask, we incorporate an FTM operator $\mathcal{M}(\cdot)$, which performs per-instance mixing across the embedding and feature dimensions using lightweight MLPs with parallel embedding-wise and feature-wise branches (see Appendix L for the full formulation):

$$Z = \mathcal{M}\Big(X_{\text{tok}} + \tilde{X}\Big) \in \mathbb{R}^{D \times d_{\text{tok}}}. \tag{2}$$

Residual Blending and Projection. To stabilize optimization, we combine the parallel streams using learnable scalars $w_1, w_2, w_3 \in \mathbb{R}$. All intermediate tensors share the same shape $\mathbb{R}^{D \times d_{\text{tok}}}$ as X_{tok}: $U = w_1 X_{\text{tok}} + \tilde{X}$, $V = w_2 X_{\text{tok}} + Z$, and $Y = w_3 U + V$. Finally, we aggregate across the token axis to return to the feature space via $y_{\text{feat}} = \text{mean}_{\text{tok}}(Y) \in \mathbb{R}^D$. We refer to this module as the *Adaptive Sparse Interaction Module (ASIM)*; it yields a feature-level representation enriched by

adaptive selectivity and global mixing while maintaining linear memory complexity in D under fixed sparse settings.

Proposition 3.1 (ASIM Complexity (Informal))**.** *Under the notation above, let the sparse attention operator attend to at most $L = L(s, b, r, m)$ keys per query feature. For fixed token dimension $d_{\text{tok}} = Hd_h$, number of heads H, and head dimension d_h, one ASIM layer has sparse-attention complexity $\mathcal{O}(DHd_hL) = \mathcal{O}(Dd_{\text{tok}}L)$ and activation memory $\mathcal{O}(D(d_{\text{tok}} + L))$. Under fixed sparse hyperparameters and fixed token dimension, both compute and memory scale linearly with the feature dimension D.*

The formal statement and proof are in Appendix B.1.

The following proposition gives a simplified conditional analysis: if the sparse support covers the informative features while excluding most background dimensions, then sparse aggregation improves the signal-to-noise ratio relative to dense aggregation.

Proposition 3.2 (Conditional SNR Improvement under Sparse Feature Selection)**.** *Let $\mathbf{x} = [\mathbf{x}_{\mathcal{S}}; \mathbf{x}_{\mathcal{N}}] \in \mathbb{R}^D$, where $\mathcal{S}$ indexes $s^\star$ informative features with average signal μ and $\mathcal{N}$ indexes i.i.d. zero-mean noise features with variance σ^2. Consider dense aggregation over all D features and sparse aggregation over a support $\mathcal{T}$ satisfying $\mathcal{S} \subseteq \mathcal{T}$ and $|\mathcal{T}| = mb$. Then, for $mb > s^\star$,*

$$\text{SNR}_{\text{dense}} = \frac{s^\star|\mu|}{\sigma\sqrt{D - s^\star}}, \qquad \text{SNR}_{\text{sparse}} = \frac{s^\star|\mu|}{\sigma\sqrt{mb - s^\star}}.$$

Thus, when the sparse support covers the informative features while excluding most background dimensions, sparse aggregation improves SNR by

$$\frac{\text{SNR}_{\text{sparse}}}{\text{SNR}_{\text{dense}}} = \sqrt{\frac{D - s^\star}{mb - s^\star}}.$$

In the ideal case $mb = s^\star$, the sparse support contains no background noise, so the noise contribution vanishes under this simplified model.

The full statement and proof are provided in Appendix B.2.

3.4 Multi-Stage Feature Refinement

The TabNSM head processes the enriched representation $y_{\text{feat}} \in \mathbb{R}^D$ from the ASIM through a multi-scale residual pipeline. This design enables the model to iteratively refine predictions via coarse-to-fine pathways.

Progressive Branches. Rather than using a purely sequential bottleneck, TabNSM employs residually coupled branches that repeatedly access the full representation y_{feat} while progressively reducing dimensionality. The first branch is a shallow GELU network $\text{MLP}_1 : \mathbb{R}^D \to \mathbb{R}^D$ with a residual connection to y_{feat}, reduced to half width by Dec_1: $y_1 = \text{Dec}_1(\text{MLP}_1(y_{\text{feat}}) + y_{\text{feat}}) \in \mathbb{R}^{D/2}$. The second branch, $\text{MLP}_2 : \mathbb{R}^D \to \mathbb{R}^{D/2}$, performs residual learning in this intermediate $D/2$ space and is reduced to a scalar by Dec_2: $y_2 = \text{Dec}_2(\text{MLP}_2(y_{\text{feat}}) + y_1) \in \mathbb{R}^1$. The third branch, $\text{MLP}_3 : \mathbb{R}^D \to \mathbb{R}^1$, provides the final scalar refinement, $y_3 = \text{MLP}_3(y_{\text{feat}}) + y_2 \in \mathbb{R}^1$; MLP_2 and MLP_3 are shallow GELU networks, so every stage remains grounded in y_{feat}.

Feature Fusion Head. The stage outputs are concatenated and passed through a compact fusion MLP $h : \mathbb{R}^{D/2+2} \to \mathbb{R}$ to produce the prediction $\hat{y} = h([y_1, y_2, y_3]) + y_2 + y_3$, which lets the head learn residual corrections that integrate coarse and fine-grained information at negligible cost. The FTM operator and the head add $\mathcal{O}(Dd_{\text{tok}})$ and $\mathcal{O}(D)$ respectively, so under fixed sparse hyperparameters the per-layer cost of TabNSM is dominated by the $\mathcal{O}(Dd_{\text{tok}}L)$ sparse-attention term of Proposition 3.1.

3.5 GridLoss: Differentiable Soft-Binning for Ordinal Alignment

Regression losses such as ℓ_1 and Huber penalize pointwise errors but do not explicitly enforce *ordinal alignment* between predictions and targets. To preserve relative positions along the target scale, we introduce GridLoss, a differentiable soft-binning objective that aligns predictions and targets in a shared discretized index space.

Batch-Adaptive Grid Construction. For a mini-batch of size N, let $\hat{y}, y \in \mathbb{R}^N$ denote the predictions and targets. We define a batch-adaptive interval $[\alpha, \beta]$ as:

$$\alpha = \min(\min(\hat{y}), \min(y)),\ \beta = \max(\max(\hat{y}), \max(y)). \tag{3}$$

We construct a uniform grid of $K+1$ anchor points $e_k = \alpha + k \cdot \frac{\beta-\alpha}{K}$ for $k \in \{0, \ldots, K\}$. A small constant $\epsilon > 0$ (e.g., 10^{-8}) is added to $\beta - \alpha$ to prevent degenerate ranges. In implementation, α and β are computed per mini-batch and detached from the computation graph.

Soft Assignment via Temperature-Scaled Distances. We compute the distances of each sample to all anchor points: $d_k^{\text{pred}} = |\hat{y} - e_k|$ and $d_k^{\text{true}} = |y - e_k|$. These are converted into soft assignments via a temperature-scaled softmax:

$$w_k^{\text{pred}} = \frac{\exp(-d_k^{\text{pred}}/\tau)}{\sum_{j=0}^{K} \exp(-d_j^{\text{pred}}/\tau)}, \quad w_k^{\text{true}} = \frac{\exp(-d_k^{\text{true}}/\tau)}{\sum_{j=0}^{K} \exp(-d_j^{\text{true}}/\tau)},$$

where $\tau > 0$ controls the assignment softness. We then compute soft indices as expectations over the anchors: $b^{\text{pred}} = \sum_{k=0}^{K} k w_k^{\text{pred}}$ and $b^{\text{true}} = \sum_{k=0}^{K} k w_k^{\text{true}}$.

Objective Function. Let $\Delta = (\beta - \alpha)/K$ denote the grid step. The GridLoss objective is the mean ℓ_1 distance between the soft indices, rescaled from index units back to target units:

$$\mathcal{L}_{\text{Grid}} = \frac{\Delta}{N} \sum_{i=1}^{N} |b_i^{\text{pred}} - b_i^{\text{true}}|, \tag{4}$$

and the temperature is specified in units of the grid step, $\tau = \tau_0 \Delta$ with τ_0 fixed (0.5 by default), so that the assignment softness is invariant to the target scale. Both conventions are required by the consistency analysis in Appendix B.4. An unnormalized variant omits the Δ rescaling and uses an absolute temperature, which is numerically degenerate on large-range targets; Section 4.2 reports the ablation under both variants. GridLoss operates in an ordinal index space, encouraging predictions to fall into the correct relative region of the target distribution. As formally analyzed in Theorem B.9 and Proposition B.13 (Appendix B.4), under the high-resolution and low-temperature limit ($K \to \infty, \tau \to 0^+$), the normalized objective approaches an MAE-like ordinal distance. Thus, GridLoss provides a smooth, structure-aware approximation to robust pointwise regression losses while preserving ordinal information in the target space.

GridLoss has three useful properties: its soft index is monotone in the prediction, its gradient is bounded by $K/(2\tau)$, and it is Lipschitz continuous. For finite τ, these properties imply stable gradients, while the high-resolution, low-temperature limit yields an MAE-like behavior; formal statements and proofs are provided in Appendix B.3.

3.6 Training Strategy with RISE Sampling

Standard regression pipelines typically construct mini-batches via uniform sampling, implicitly treating all training instances as equally informative. In high-dimensional tasks with long-tailed error distributions, this can underrepresent "hard" samples that often correspond to minority subpopulations or extreme target tails, leading to suboptimal generalization. To mitigate this, we adopt a difficulty-aware sampling strategy termed RISE (Reweighted Instance Sampling by Error). RISE periodically estimates per-sample prediction errors and dynamically biases the sampling distribution toward harder examples.

The RISE-enhanced training procedure adapts the model's focus as convergence progresses. Every k epochs, we refresh the sampling distribution via the following steps:

1. **Loss Estimation:** Compute per-sample losses ℓ_i for the entire training set using the current parameters θ_e.
2. **Quantile Partitioning:** Partition samples into G difficulty bins based on the loss quantiles.
3. **Weight Assignment:** Assign a sampling weight $w_i = 1 + g_i \cdot \frac{\beta_{\text{RISE}}-1}{G-1}$, where $g_i \in \{0, \ldots, G-1\}$ is the bin index and $\beta_{\text{RISE}} \geq 1$ is the maximum boost factor.
4. **Weighted Sampling:** Draw mini-batches using a weighted random sampler with probabilities $p_i \propto w_i$.

By upweighting bins with larger errors, RISE implements a dynamic curriculum that prioritizes samples currently underfit by the model or lying in difficult regions of the target distribution. The full pseudocode is provided in Algorithm 1 (Appendix C). Because RISE draws weighted mini-batches without an inverse-probability correction, the expected SGD step descends the reweighted empirical risk $\sum_i w_i \ell_i(\theta) / \sum_j w_j$ rather than the uniform one; with a monotone weight map this emphasizes the tail of the error distribution (formal statement and proof in Appendix B.5).

4 Experiments and Results

Datasets and protocol. We evaluate on seven real-world regression datasets with more than 250 features (266–405) and 4k–1M rows, spanning transportation, public health, criminology, real estate, air quality, and neuroimaging (Table 15; Appendix H). All are highly redundant: five derive 95–99% of their columns from a sentence-embedding block, and all seven have a normalized participation-ratio redundancy score below 0.14 (Appendix H). All methods share a fixed 72/8/20 train/validation/test split; rows with missing targets are removed, missing features are zero-imputed, and features are standardized with training-set statistics. Models train for up to 100 epochs with early stopping on validation loss and are evaluated at the best-validation checkpoint. TabNSM uses AdamW, mixed precision, and RISE sampling (Appendix C); its ASIM hyperparameters, learning rate, and batch size are selected with Optuna [49] on the validation split. Baselines are tuned with 20 Optuna trials each under 3-fold cross-validation on the training and validation rows, and pretrained models use their released defaults (Appendix F). Entries in Tables 1 and 3 are single runs; TabNSM's seed variability, measured on the five datasets loadable by the released pipeline, is 2–13% relative (Appendix I), and we read all margins against this band. Hardware is stated wherever time or memory is reported.

4.1 Main Results

Table 1 reports RMSE against 41 baselines grouped into classical methods, tree ensembles, MLP-style models, transformers (including AMFormer [50]), retrieval-based methods, other deep tabular architectures, and tabular foundation models (TabPFNv2, TabPTM, TabDPT [17], TabICLv2 [16], TabFM [18]). Each row is the best baseline in its family; per-model results are in Table 3 (Appendix D.1).

Table 1: RMSE on the seven wide regression benchmarks (> 250 features); lower is better, best in **bold**, second-best underlined. The last column, # Wins, is the number of datasets (of seven) on which a row attains the bolded best RMSE. Each row is the best-performing baseline in its family; full per-model results are in Appendix D.1. All entries are single runs (seed variability: Appendix I). The two narrow datasets, SA (27 features) and CP (12 features), are in Appendix D.3.

Model	AQ	TO	SC	CD	ChD	EVP	RES	# Wins
Best Classical (6 models)	12.816	0.027	3.602	0.236	8717	26.104	3163	0
Best Tree-based (5 models)	5.869	0.027	3.455	**0.004**	17525	2.799	3135	1
Best MLP-style (5 models)	4.578	0.028	3.920	0.028	4963	1.556	3161	0
Best Transformer (7 models)	4.081	0.028	3.667	0.007	4475	1.507	3163	0
Best Retrieval (4 models)	5.062	0.028	3.454	0.052	22661	2.653	3187	0
Best Other (9 models)	4.690	0.027	3.522	0.136	4808	3.219	3163	0
Best Foundation (5 models)	**3.577**	0.027	3.437	0.007	22771	1.897	3164	1
TabNSM (Ours)	3.999	**0.021**	**3.322**	0.007	**4460**	**1.199**	**3103**	**5**

Per-family best is the minimum RMSE among runnable models in that family. TabPFNv2 is entered only for TO and SC, whose training sets fit within its 10,000-row limit; the note on $*$ in Appendix D.1 explains how it was run and why it is not entered on AQ, CD, ChD, EVP, and RES. TabFM exceeds the 80 GB GPU on CD, ChD, EVP, and RES, so the Foundation entry there is the best of the remaining models.

TabNSM has the lowest RMSE on five of the seven datasets (TO, SC, ChD, EVP, RES), is second on AQ to TabFM (3.999 vs. 3.577) and on CD to ExtraTrees (0.007 vs. 0.004; TabICLv2 is indistinguishable at the reported precision), and has an average rank of 1.43 among 42 methods. One-sided pairwise Wilcoxon signed-rank tests on per-dataset ranks, Benjamini–Hochberg corrected and with missing runs imputed as worst rank, give a significantly better rank than each of the 41 baselines (corrected $p \leq 0.016$), including TabICLv2 ($p = 0.015$) and TabFM ($p = 0.016$, which depends on imputing its four out-of-memory runs). Restricted to the 27 methods with results on all seven datasets, TabNSM still ranks first (average rank 1.29; Friedman $\chi^2(26) = 64.0$, $p = 4.8 \times 10^{-5}$); tests over all nine datasets are in Appendix D.2.

Reading the results. The margins over the strongest baseline are not uniform. On EVP, TabNSM attains an RMSE 21% below that of the best baseline (FT-Transformer, 1.507; tuned single run). On ChD and RES the difference is 0.3–1%, which we treat as a tie with the best baseline, and on AQ and

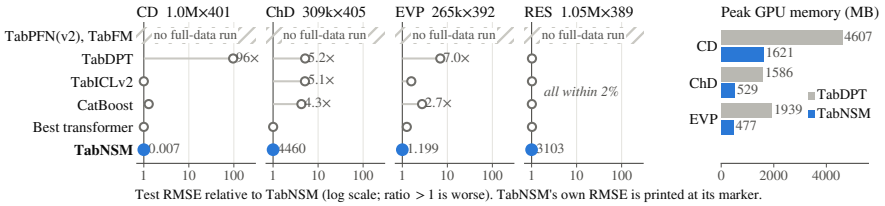


Figure 2: The four largest datasets. *Left:* test RMSE relative to TabNSM for the models that complete (single runs from Table 3; log scale; CD to three decimals). TabPFNv2 (limited to 10k rows), TabPFN with limits disabled (out of memory; not run on RES) and TabFM (out of memory on 80 GB) have no full-training-set result. *Right:* peak GPU memory on an RTX A4000 (not measured for RES); exact values in Table 22, Appendix J.

CD TabNSM ranks second (Appendix G.2). The strongest baseline differs from one dataset to the next, whereas TabNSM remains at or near the top on all five. TO and SC do not discriminate between methods: the baselines cluster around the training-mean predictor (0.0284 and 3.972), and TabNSM's default configuration performs at that level as well ($R^2 = -0.04$ and $+0.01$; Table 12; 5-seed means 0.034 and 5.15, Appendix I). Both datasets are retained in the tables, and the tuned single-run entries (0.021 and 3.322) are reported for completeness. Our conclusions therefore rest on AQ, CD, ChD, EVP, and RES.

4.2 Matched Component Ablation

We ran a common component-removal protocol on the five datasets loadable by the released pipeline (CP, TO, SA, SC, AQ), holding the split, initialization seed, optimizer, training budget, and stopping rule fixed. Removing ASIM replaces the sparse attention operator with dense multi-head attention over the same inputs; removing GridLoss leaves a plain Huber loss; removing RISE reverts to uniform sampling. Table 23 (Appendix M.1) gives the percentage change in test RMSE relative to the full model; variants are evaluated at the terminal training state (the ablation scripts save, but do not reload, the best-validation checkpoint) and are single-seed. Removing the sparse module increases RMSE on all five datasets (mean $+6.1\%$, up to $+18.1\%$ on TO), the largest and most consistent effect of any component, although the increases on SA, SC, and AQ (0.6–1.5%) are within the seed band. RISE matters where the target is heavy-tailed (Table 29): removing it costs $+17.3\%$ on CP, consistent with its emphasis on high-error tail samples, while on the other four datasets it changes RMSE by -0.4% to -4.4%, within seed variability. For GridLoss, an earlier unnormalized implementation hurt on every dataset; with the grid-step normalization of Eq. (4), removal increases RMSE on three of five (mean $+2.1\%$), again within seed variability. We therefore present ASIM as the component that carries the empirical result and GridLoss as a theoretically grounded regularizer (bounded, Lipschitz gradients; MAE-consistent limit) with a modest accuracy effect. Each component is removed one at a time, so interactions between components are not measured. Appendix M additionally reports an operator-swap comparison with dense attention and Mamba (from an earlier version of the pipeline), the FTM pathway (whose removal degrades RMSE on all four datasets tested), an HL-Gauss-style objective, and column-order sensitivity.

4.3 Memory and Scaling on Large, Wide Tables

Figure 2 covers the four largest datasets (2.6×10^5–1.05×10^6 rows, 389–405 features). TabPFNv2 is specified for at most 10,000 training rows, so through the TALENT wrapper [51] each ensemble member sees a 10,000-row subsample rather than the full training set; with the limits disabled, TabPFN requests 34–43 GiB on ChD and EVP, beyond the 16 GB GPU used, and exhausts host RAM on CD. TabFM exceeds an 80 GB H100 on all four. TabDPT completes by retrieving a 2,048-row context per query, at three to four times TabNSM's peak GPU memory and 5–7× TabNSM's RMSE on ChD and EVP (about 100× on CD), but with a smaller CPU-memory increase (Appendix J). TabNSM trains on all four within 0.5–1.6 GB of GPU memory (measured on three) and matches or beats every model that completes on ChD, EVP, and RES. Mini-batch training is why TabNSM, unlike TabPFN and TabFM, does not need the training set in memory at once; within that, ASIM's activation cost is linear in D (Proposition 3.1), and Figure 3 shows parameter count scaling the same way.

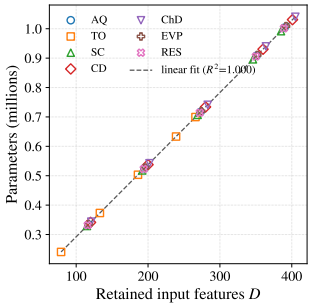


Figure 3: Parameter count versus retained feature fraction, per dataset: growth is approximately linear in D.

Table 2: Remove-and-retrain on AQ and EVP: test RMSE after deleting the given fraction of features and retraining from scratch. AUC is the mean RMSE increase over all four fractions (10/30/50/70%; 50% omitted here). EVP is a 20k-row subsample, mean of three seeds; AQ non-random rows are single runs. Appendix G.2 adds s.d., the 50% column, and TO and SC.

Dataset	Removal ranking	10%	30%	70%	AUC
AQ (full feature set: 4.016)					
	TabNSM attribution (top-k)	3.841	11.063	11.286	**4.99**
	Random (3 draws)	4.213	4.162	6.306	0.98
	TabNSM attribution (bottom-k)	4.098	4.187	4.552	0.06
EVP (full feature set: 3.871 ± 0.250)					
	TabNSM attribution (top-k)	28.95	33.29	44.76	**29.5**
	Random (3 draws)	5.46	6.10	20.34	4.1
	TabNSM attribution (bottom-k)	4.11	3.81	4.27	0.19

4.4 Validated Feature Attribution

The architecture assumes each prediction depends on a small subset of columns. We test that assumption with *remove-and-retrain* (ROAR; 52): features are ranked by mean $|\text{gradient} \times \text{input}|$ on the training split, the top-$k\%$ are deleted, and the model is rebuilt at the reduced width and retrained from scratch, so no model sees a masked input. Controls repeat the pipeline on the bottom-$k\%$ and on a random $k\%$. Of the four datasets on which we ran it, AQ and EVP are the two where the model clearly beats predicting the training mean ($R^2 = 0.975$ and 0.997); Table 2 reports them. Deleting the top-ranked 30% of features on AQ nearly triples test RMSE ($4.02 \to 11.06$); on EVP, deleting the top 10% raises it more than sevenfold ($3.87 \to 28.95$) and the top 70% to 44.8. Deleting the bottom-ranked features leaves both models at their full-feature error and random deletion sits in between; the resulting ordering, summarized by the AUC column, is top-$k \gg$ random $>$ bottom-k—the signature of a faithful ranking, and a higher AUC means the ranking found more load-bearing features. Since 98% of the columns on both datasets come from a 384-dimensional sentence-embedding block (Table 16), this locates signal rather than naming features, and finds it concentrated in a small part of a representation that might be expected to spread it evenly—the structure ASIM's per-sample selection exploits. Appendix G gives the full protocol, the four-dataset table, a comparison against TabNet's native masks, randomization checks [53], and per-instance attribution maps.

5 Conclusion and Limitations

This paper began from a *foreground–background* view of wide tables: a small, record-specific subset of columns carries the signal, and no fixed subset serves every record. ASIM turns that view into an architecture, selecting each record's foreground feature groups and modeling their interactions at a cost linear in the column count—a differentiable counterpart to tree-ensemble feature isolation that stays trainable end to end with learned embeddings. The evidence supports it: TabNSM attains the lowest error on five of seven benchmarks and a significantly better rank than each of 41 baselines, an effect the matched ablation attributes to the sparse module rather than GridLoss or RISE, and its feature attributions pass remove-and-retrain validation. Mini-batch training also lets the same design scale to about a million rows in a small, fixed GPU-memory budget, where in-context foundation models run out of memory or must subsample (Section 4.3).

Scope and limitations. TabNSM's advantage is tied to an intact, redundant embedding block: under compression with independent noise padding, the margin over baselines closes or reverses (Appendix M.7), which bounds the claim to this block structure rather than to noise robustness or width in general. Tuned gradient-boosted trees and in-context models are the better choice on low-dimensional and standardized benchmarks (Appendices D.3, E.1), and CatBoost is one to two orders of magnitude faster to train on matched hardware (Appendix J); the linear-in-D result is a scaling and memory property, not a wall-clock claim. Main-table entries are single runs, so most margins should be read against the measured 2–13% seed band rather than as precise point estimates (Appendix I). Multi-seed evaluation under a documented tuning budget, a block-selection rule independent of column position, and an extension to classification define the natural next steps.

Ethics Statement

This work introduces TabNSM, a framework for tabular regression. While intended for beneficial applications such as healthcare resource optimization, industrial monitoring, and financial risk assessment, high-capacity models trained on high-dimensional tabular data can carry potential risks. Specifically, predictive models applied in financial or social domains could be exploited to facilitate predatory algorithmic strategies or financial fraud if deployed without proper oversight and fairness audits. The authors advocate for responsible deployment and regulatory compliance in sensitive applications.

Reproducibility Statement

To ensure the reproducibility of our results, detailed descriptions of the architecture, loss functions, and optimization steps are provided in Section 3 and Appendix C. Detailed hyperparameter configurations and preprocessing procedures are documented in Appendix F, and dataset descriptions for all benchmark datasets are provided in Appendix H. Furthermore, an anonymized version of our complete codebase and experimental scripts has been provided in the supplementary material to allow full replication of our main empirical evaluations and ablation studies. Code available on: `https://anonymous.4open.science/r/TabNSM-2B15/README.md`

References

[1] Jintai Chen, Kuanlun Liao, Yao Wan, Danny Z. Chen, and Jian Wu. DANets: Deep abstract networks for tabular data classification and regression. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(4):3930–3938, Jun. 2022.

[2] Boshko Koloski, Andrei Margeloiu, Xiangjian Jiang, Blaž Škrlj, Nikola Simidjievski, and Mateja Jamnik. LLM embeddings for deep learning on tabular data. *arXiv preprint arXiv:2502.11596*, 2025.

[3] Ali Eslamian and Qiang Cheng. TabNSA: Native sparse attention for efficient tabular data learning. *Neurocomputing*, 675:132928, 2026.

[4] Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. CatBoost: unbiased boosting with categorical features. In *Advances in Neural Information Processing Systems*, 2018.

[5] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. LightGBM: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*, 2017.

[6] 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*, 2016.

[7] Tony Duan, Anand Avati, Daisy Yi Ding, Khanh K Thai, Sanjay Basu, Andrew Ng, and Alejandro Schuler. NGBoost: Natural gradient boosting for probabilistic prediction. In *International conference on machine learning*, pages 2690–2700. PMLR, 2020.

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

[9] 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, pages 18932–18943, 2021.

[10] Weiping Song, Chence Shi, Zhiping Xiao, Zhijian Duan, Yewen Xu, Ming Zhang, and Jian Tang. AutoInt: Automatic feature interaction learning via self-attentive neural networks. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 1161–1170, 2019.

[11] Md Atik Ahamed and Qiang Cheng. MambaTab: A plug-and-play model for learning tabular data. In *2024 IEEE 7th International Conference on Multimedia Information Processing and Retrieval (MIPR)*, pages 369–375. IEEE, 2024.

[12] Yongchang Chen, Yinning Liu, Guomin Chen, Qian Zhang, Yilin Wu, and Jinxin Ruan. FT-Mamba: A novel deep learning model for efficient tabular regression. In *2024 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 3096–3100, 2024.

[13] Anton Frederik Thielmann, Manish Kumar, Christoph Weisser, Arik Reuter, Benjamin Säfken, and Soheila Samiee. Mambular: A sequential model for tabular deep learning. *arXiv preprint arXiv:2408.06291*, 2024.

[14] 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*, 637(8045):319–326, 2025.

[15] Jingang Qu, David Holzmüller, Gaël Varoquaux, and Marine Le Morvan. TabICL: A tabular foundation model for in-context learning on large data. In *Forty-second International Conference on Machine Learning*, 2025.

[16] Jingang Qu, David Holzmüller, Gaël Varoquaux, and Marine Le Morvan. TabICLv2: A better, faster, scalable, and open tabular foundation model. *arXiv preprint arXiv:2602.11139*, 2026. Accepted to ICML 2026.

[17] Junwei Ma, Valentin Thomas, Rasa Hosseinzadeh, Alex Labach, Hamidreza Kamkari, Jesse C. Cresswell, Keyvan Golestan, Guangwei Yu, Anthony L. Caterini, and Maksims Volkovs. Tabdpt: Scaling tabular foundation models on real data. *Advances in Neural Information Processing Systems*, 38:172692–172722, 2025.

[18] Google Research. TabFM: Tabular foundation model. `https://github.com/google-research/tabfm`, 2026. Version 1.0.0; weights at `https://huggingface.co/google/tabfm-1.0.0-pytorch`.

[19] Nick Erickson, Lennart Purucker, Andrej Tschalzev, David Holzmüller, Prateek Mutalik Desai, David Salinas, and Frank Hutter. TabArena: A living benchmark for machine learning on tabular data. In *Advances in Neural Information Processing Systems (Datasets and Benchmarks Track)*, 2025. arXiv:2506.16791.

[20] Vadim Borisov, Tobias Leemann, Kathrin Seßler, Johannes Haug, Martin Pawelczyk, and Gjergji Kasneci. Deep neural networks and tabular data: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[21] Arlind Kadra, Marius Lindauer, Frank Hutter, and Josif Grabocka. Well-tuned simple nets excel on tabular datasets. In *Advances in Neural Information Processing Systems*, 2023.

[22] Liran Katzir, Gal Elidan, and Ran El-Yaniv. Net-DNF: Effective deep modeling of tabular data. In *International conference on learning representations*, 2020.

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

[24] Chenwei Xu and Yifan Wang. NCART: Neural classification and regression tree for tabular data. *Pattern Recognition*, 148:110–125, 2024.

[25] Noah Hollmann, Samuel Müller, Katharina Eggensperger, and Frank Hutter. TabPFN: A transformer that solves small tabular classification problems in a second. In *NeurIPS 2022 First Table Representation Workshop*, 2022.

[26] Andreas C Mueller, Carlo A Curino, and Raghu Ramakrishnan. MotherNet: Fast training and inference via hyper-network transformers. In *The Thirteenth International Conference on Learning Representations*, 2025.

[27] Han-Jia Ye, Si-Yang Liu, and Wei-Lun Chao. A closer look at tabpfn v2: Strength, limitation, and extension. *CoRR*, abs/2502.17361, February 2025.

[28] Léo Grinsztajn, Klemens Flöge, Oscar Key, Felix Birkel, Philipp Jund, Brendan Roof, Benjamin Jäger, Dominik Safaric, Simone Alessi, Adrian Hayler, et al. TabPFN-2.5: Advancing the state of the art in tabular foundation models. *arXiv preprint arXiv:2511.08667*, 2025.

[29] Xin Huang, Ashish Khetan, Milan Cvitkovic, and Zohar Karnin. TabTransformer: Tabular data modeling using contextual embeddings. *arXiv preprint arXiv:2012.06678*, 2020.

[30] Wenji Li. Tabnet for high-dimensional tabular data: advancing interpretability and performance with feature fusion. *IET Conference Proceedings*, 2025:168–173, 2025.

[31] Jiho Park and Min-Soo Kim. Optimizing FT-Transformer: Sparse attention for improved performance on tabular data. *Pattern Recognition Letters*, 178, 2024.

[32] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *First conference on language modeling*, 2024.

[33] Yifan Wang, J Sun, et al. Representation learning for tabular data: A comprehensive survey. *Journal of Machine Learning Research*, 2025.

[34] Pasan Dissanayake and Sanghamitra Dutta. Distilling transformers into neural nets for few-shot tabular classification, 2025.

[35] Stefan Hegselmann, Alejandro Buendia, Hunter Lang, Monica Agrawal, Xiaoyi Jiang, and David Sontag. TabLLM: Few-shot classification of tabular data with large language models. In Francisco Ruiz, Jennifer Dy, and Jan-Willem van de Meent, editors, *Proceedings of The 26th International Conference on Artificial Intelligence and Statistics*, volume 206 of *Proceedings of Machine Learning Research*, pages 5549–5581. PMLR, 25–27 Apr 2023.

[36] L Zhang and J Wu. Embedding strategies for categorical and text features in tabular deep learning. *Computational Statistics*, 2023.

[37] Manan Roy Choudhury, Anirudh Iyengar Kaniyar Narayana Iyengar, Shikhhar Siingh, Sugeeth Puranam, and Vivek Gupta. TABARD: A novel benchmark for tabular anomaly analysis, reasoning and detection. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 21783–21817, 2025.

[38] Xiyuan Zhang, Danielle C. Maddix, Junming Yin, Nick Erickson, Abdul Fatir Ansari, Boran Han, Shuai Zhang, Leman Akoglu, Christos Faloutsos, Michael W. Mahoney, Cuixiong Hu, Huzefa Rangwala, George Karypis, and Bernie Wang. Mitra: Mixed synthetic priors for enhancing tabular foundation models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2026.

[39] Chenwei Xu, Yu-Chao Huang, Jerry Yao-Chieh Hu, Weijian Li, Ammar Gilani, Hsi-Sheng Goan, and Han Liu. BiSHop: Bi-directional cellular learning for tabular data with generalized sparse modern hopfield model. In *Forty-first International Conference on Machine Learning (ICML)*, 2024.

[40] Sedigheh Khaki et al. A methods guideline for deep learning for tabular data in agriculture. *Computers and Electronics in Agriculture*, 205, 2023.

[41] Ehsan Imani and Martha White. Improving regression performance with distributional losses. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2157–2166. PMLR, 10–15 Jul 2018.

[42] X Li et al. Rethinking soft labels for regression. In *International Conference on Machine Learning*, 2022.

[43] Weijieying Ren, Tianxiang Zhao, Yuqing Huang, and Vasant Honavar. Deep learning within tabular data: Foundations, challenges, advances and future directions. *arXiv preprint arXiv:2501.03540*, 2025.

[44] Badri Narayana Patro and Vijay Srinivas Agneeswaran. Mamba-360: Survey of state space models as transformer alternative for long sequence modelling: Methods, applications, and challenges. *Engineering Applications of Artificial Intelligence*, 159:111279, 2025.

[45] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[46] Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using siamese bert-networks. In *EMNLP*, 2019.

[47] Wenhui Wang, Hangbo Bao, Shaohan Huang, et al. MiniLM: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. In *NeurIPS*, 2020.

[48] Sebastian Bordt, Harsha Nori, and Rich Caruana. TAB2TEXT: A framework for deep learning with tabular data. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, 2024.

[49] 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 & Data Mining*, pages 2623–2631, 2019.

[50] Yi Cheng, Renjun Hu, Haochao Ying, Xing Shi, Jian Wu, and Wei Lin. Arithmetic feature interaction is necessary for deep tabular learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 11516–11524, 2024.

[51] Si-Yang Liu, Hao-Run Cai, Qi-Le Zhou, Huai-Hong Yin, Tao Zhou, Jun-Peng Jiang, and Han-Jia Ye. Talent: A tabular analytics and learning toolbox. *Journal of Machine Learning Research*, 26(226):1–16, 2025.

[52] Sara Hooker, Dumitru Erhan, Pieter-Jan Kindermans, and Been Kim. A benchmark for interpretability methods in deep neural networks. *Advances in neural information processing systems*, 32, 2019.

[53] Julius Adebayo, Justin Gilmer, Michael Muelly, Ian Goodfellow, Moritz Hardt, and Been Kim. Sanity checks for saliency maps. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/294a8ed24b1ad22ec2e7efea049b8737-Paper.pdf.

[54] Yang You, Igor Gitman, and Boris Ginsburg. Large batch training of convolutional networks. *arXiv preprint arXiv:1708.03888*, 2017.

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

[56] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine Learning*, 63(1):3–42, 2006.

[57] Yury Gorishniy, Ivan Rubachev, and Artem Babenko. On embeddings for numerical features in tabular deep learning. In *Advances in Neural Information Processing Systems*, volume 35, pages 24991–25004, 2022.

[58] Yury Gorishniy, Akim Kotelnikov, and Artem Babenko. TabM: Advancing tabular deep learning with parameter-efficient ensembling. In *International Conference on Learning Representations*, 2025. arXiv:2410.24210.

[59] Jiahuan Yan, Jintai Chen, Jian Wu, et al. T2G-Former: Organizing tabular features into relation graphs promotes heterogeneous feature interaction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2023.

[60] Gowthami Somepalli, Micah Goldblum, Avi Schwarzschild, C.B̃ayan Bruss, and Tom Goldstein. SAINT: Improved neural networks for tabular data via row attention and contrastive pre-training. *arXiv preprint arXiv:2106.01342*, 2021.

[61] Ruoxi Wang, Rakesh Shivanna, Derek Z. Cheng, Sagar Jain, Dong Lin, Lichan Hong, and Ed H. Chi. DCN V2: Improved deep & cross network and practical lessons for web-scale learning to rank systems. In *Proceedings of the Web Conference 2021*, pages 1785–1797, 2021.

[62] Jintai Chen, Jiahuan Yan, Qiyuan Chen, Danny Ziyi Chen, Jian Wu, and Jimeng Sun. ExcelFormer: A neural network surpassing GBDTs on tabular data. *arXiv preprint arXiv:2301.02819*, 2023.

[63] Yury Gorishniy, Ivan Rubachev, Nikolay Kartashev, Daniil Shlenskii, Akim Kotelnikov, and Artem Babenko. TabR: Tabular deep learning meets nearest neighbors in 2023. In *International Conference on Learning Representations*, 2024.

[64] Han-Jia Ye et al. Modern neighborhood components analysis. *arXiv preprint arXiv:2407.03257*, 2024.

[65] 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.

[66] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[67] Alan Jeffares, Tennison Liu, Jonathan Crabbé, Fergus Imrie, and Mihaela van der Schaar. TANGOS: Regularizing tabular neural networks through gradient orthogonalization and specialization. In *International Conference on Learning Representations*, 2023.

[68] Chao Ye et al. PTaRL: Prototype-based tabular representation learning via space calibration. In *International Conference on Learning Representations*, 2024.

[69] Sarkhan Badirli, Xuanqing Liu, Zhengming Xing, Avradeep Bhowmik, Khoa Doan, and Sathiya S. Keerthi. Gradient boosting neural networks: GrowNet. *arXiv preprint arXiv:2002.07971*, 2020.

[70] Youssef Nader, Leon Sixt, and Tim Landgraf. DNNR: Differential nearest neighbors regression. In *Proceedings of the 39th International Conference on Machine Learning*, pages 16296–16317, 2022.

[71] Jing Wu, Suiyao Chen, Qi Zhao, Renat Sergazinov, Chen Li, Shengjie Liu, Chongchao Zhao, Tianpei Xie, Hanqing Guo, Cheng Ji, Daniel Cociorva, and Hakan Brunzell. SwitchTab: Switched autoencoders are effective tabular learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 15924–15933, 2024.

[72] Sascha Marton, Stefan Lüdtke, Christian Bartelt, and Heiner Stuckenschmidt. GRANDE: Gradient-based decision tree ensembles. In *International Conference on Learning Representations*, 2024.

[73] Bingzhao Liu et al. TabPTM: Pre-trained model for tabular data via meta-representation. *arXiv preprint arXiv:2303.09884*, 2023.

[74] Yin Lou, Rich Caruana, Johannes Gehrke, and Giles Hooker. Accurate intelligible models with pairwise interactions. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 623–631, 2013.

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

[76] Yoav Benjamini and Yosef Hochberg. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society: Series B (Methodological)*, 57(1):289–300, 1995.

[77] David Holzmüller, Leo Grinsztajn, and Ingo Steinwart. Realmlp: Advancing mlps and default parameters for tabular data. In *ELLIS workshop on Representation Learning and Generative Models for Structured Data*, 2025.

[78] Xiangjian Jiang, Andrei Margeloiu, Nikola Simidjievski, and Mateja Jamnik. Protogate: Prototype-based neural networks with local feature selection for tabular biomedical data. In *ICML 3rd Workshop on Interpretable Machine Learning in Healthcare (IMLH)*, 2023.

[79] Kuan-Yu Chen, Ping-Han Chiang, Hsin-Rung Chou, Ting-Wei Chen, and Tien-Hao Chang. Trompt: Towards a better deep neural network for tabular data. In *ICML*, pages 4392–4434, 2023. URL `https://proceedings.mlr.press/v202/chen23c.html`.

[80] David Bonet, Daniel Mas Montserrat, Xavier Giró-i Nieto, and Alexander G Ioannidis. Hyperfast: Instant classification for tabular data. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 11114–11123, 2024.

[81] Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Yuxing Wei, Lean Wang, Zhiping Xiao, Yuqing Wang, Chong Ruan, Ming Zhang, Wenfeng Liang, and Wangding Zeng. Native sparse attention: Hardware-aligned and natively trainable sparse attention. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 23078–23097, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0.

A THEORETICAL INTUITION OF GRIDLOSS

Gradients and Monotonicity. GridLoss maps a scalar value x to a soft grid index

$$b(x) = \sum_{k=0}^{K} k\, w_k(x), \qquad w_k(x) = \frac{\exp(-|x - e_k|/\tau)}{\sum_{j=0}^{K} \exp(-|x - e_j|/\tau)}.$$

This mapping can be viewed as a smooth approximation to hard binning: for finite temperature τ, the weights concentrate around nearby grid anchors, so $b(x)$ behaves like a differentiable coordinate on the target grid. The derivative $\partial b(x)/\partial x$ is controlled by the temperature and grid resolution, yielding bounded gradients for finite τ. As x increases, mass shifts from lower-index anchors to higher-index anchors, making $b(x)$ nondecreasing on an ordered grid. Thus, GridLoss provides an ordinal training signal: it encourages predictions not only to reduce pointwise error, but also to occupy the correct relative position along the target scale.

Why Not KL/Wasserstein Distribution Matching. Distributional objectives such as KL divergence or Wasserstein distance typically require constructing normalized distributions, such as histograms or kernel-smoothed densities, over predicted and true targets. In per-sample regression, this introduces additional design choices, including binning, smoothing, regularization, and, for optimal transport, potentially iterative solvers. Moreover, batch-level distribution matching can encourage agreement between marginal target distributions without directly enforcing alignment between each prediction–target pair. GridLoss instead performs a sample-wise comparison by mapping each $(\hat{y}_i, y_i)$ pair onto a shared grid and penalizing their soft-index distance. This captures ordinal geometry with low overhead, controlled mainly by the grid resolution K and temperature τ.

Why Pointwise Regression Losses Alone Can Be Insufficient. Standard pointwise losses optimize residual magnitudes but do not explicitly encode ordinal placement on the target scale. MSE and RMSE emphasize large residuals and can over-focus on outliers; MAE and robust variants, such as Huber and log-cosh, improve robustness but still compare predictions and targets only through pointwise deviations. MAPE can be unstable near zero targets and may overweight small values, while quantile loss targets conditional quantiles rather than the conditional mean. GridLoss complements these objectives by providing an ordinal, calibration-oriented signal in grid space.

Ablation Protocol. We ablate two aspects of GridLoss. First, we compare different regression objectives, including MSE, MAE, Huber, log-cosh, GridLoss, and mixed objectives

$$\gamma \mathcal{L}_{\text{Huber}} + (1-\gamma)\mathcal{L}_{\text{Grid}}, \qquad \gamma \in \{1.0, 0.9, 0.7, 0.5, 0.3, 0.0\}.$$

Second, we study the sensitivity of GridLoss to its hyperparameters, using

$$K \in \{50, 100, 200, 500, 1000\}, \qquad \tau \in \{0.1, 0.3, 1.0, 3.0, 10.0\}.$$

In addition to RMSE, MAE, and R^2, we report ordinal and calibration-oriented metrics, including Spearman's ρ and bin accuracy within ± 1 bin, to quantify improvements in target-scale alignment.

B PROOFS OF THEORETICAL RESULTS

B.1 ASIM COMPLEXITY

The full statement of Proposition 3.1 is as follows.

Proposition B.1 (Time and Memory Complexity of ASIM)**.** *Let D be the number of scalar features and let d_{tok} be the token dimension per feature. Let $X_{\text{tok}} \in \mathbb{R}^{D \times d_{\text{tok}}}$ denote the tokenized feature representation. ASIM computes*

$$\tilde{X} = \mathcal{A}_{\text{sparse}}(X_{\text{tok}};\, s, b, r, m, H, d_h) \in \mathbb{R}^{D \times d_{\text{tok}}}, \tag{5}$$

$$Z = \mathcal{M}\Big(X_{\text{tok}} + \tilde{X}\Big) \in \mathbb{R}^{D \times d_{\text{tok}}}, \tag{6}$$

$$U = w_1 X_{\text{tok}} + \tilde{X} \in \mathbb{R}^{D \times d_{\text{tok}}}, \tag{7}$$

$$V = w_2 X_{\text{tok}} + Z \in \mathbb{R}^{D \times d_{\text{tok}}}, \tag{8}$$

$$Y = w_3 U + V \in \mathbb{R}^{D \times d_{\text{tok}}}, \tag{9}$$

followed by $y_{\text{feat}} = \text{mean}_{\text{tok}}(Y) \in \mathbb{R}^D$. *Assume the sparse attention operator attends, for each query feature, to at most*

$$L := L(s, b, r, m)$$

feature positions, where L *is determined by the fixed sparse configuration. If* $d_{\text{tok}} = Hd_h$, *then the sparse interaction part of one ASIM layer has complexity*

$$\mathcal{O}(DHd_hL) = \mathcal{O}(Dd_{\text{tok}}L)$$

and activation memory

$$\mathcal{O}(D(d_{\text{tok}} + L)) .$$

Including the token projections, FTM operator, residual blending, and token-wise aggregation, the overall ASIM computation remains linear in D *under fixed* d_{tok}, H, d_h, *and sparse hyperparameters* (s, b, r, m).

Proof. The sparse attention path restricts each query feature to at most L attended feature positions. Across D query features and H attention heads with head dimension d_h, the sparse dot-product and value-aggregation cost is

$$\mathcal{O}(DHd_hL) = \mathcal{O}(Dd_{\text{tok}}L),$$

using $d_{\text{tok}} = Hd_h$. If sparse attention weights or indices are materialized, the sparse attention pattern contributes $\mathcal{O}(DL)$ storage, while the token representations contribute $\mathcal{O}(Dd_{\text{tok}})$ storage. Thus, the sparse attention activation memory is $\mathcal{O}(D(d_{\text{tok}} + L))$.

The token projections contribute $\mathcal{O}(Dd_{\text{tok}}^2)$ computation under standard linear projections, which is linear in D for fixed d_{tok}. The FTM operator $\mathcal{M}(\cdot)$ applies lightweight mixing and permutations over the $D \times d_{\text{tok}}$ representation and contributes $\mathcal{O}(Dd_{\text{tok}})$ computation under fixed hidden widths. The residual blending operations and the token-wise mean projection also require $\mathcal{O}(Dd_{\text{tok}})$ computation and storage. Therefore, under fixed sparse hyperparameters and fixed token dimension, both compute and activation memory scale linearly with the feature dimension D. □

B.2 Conditional SNR Improvement under Sparse Feature Selection

We provide the formal statement and proof of Proposition 3.2. The result is a simplified conditional analysis: if the sparse support covers the informative features while excluding most background dimensions, then sparse aggregation improves the signal-to-noise ratio relative to dense aggregation.

Proposition B.2 (Conditional SNR Improvement under Sparse Feature Selection)**.** *Let* $\mathbf{x} = [\mathbf{x}_{\mathcal{S}}; \mathbf{x}_{\mathcal{N}}] \in \mathbb{R}^D$ *be partitioned into informative features indexed by* $\mathcal{S}$, *with* $|\mathcal{S}| = s^\star$, *and background noise features indexed by* $\mathcal{N} = \{1, \dots, D\} \setminus \mathcal{S}$. *Assume that the noise features are independent with mean zero and variance* σ^2. *Let*

$$\mu_{\mathcal{S}} = \frac{1}{s^\star} \sum_{d \in \mathcal{S}} x_d$$

denote the average signal over informative features. Define

$$A_{\text{dense}}(\mathbf{x}) = \frac{1}{D} \sum_{d=1}^{D} x_d, \qquad A_{\text{sparse}}(\mathbf{x}; \mathcal{T}) = \frac{1}{|\mathcal{T}|} \sum_{d \in \mathcal{T}} x_d,$$

where $\mathcal{T} \subseteq \{1, \dots, D\}$ *is a sparse support satisfying* $\mathcal{S} \subseteq \mathcal{T}$ *and* $|\mathcal{T}| = mb$.

The noise variance contributions are

$$\text{Var}_{\mathcal{N}}[A_{\text{dense}}] = \frac{D - s^\star}{D^2}\sigma^2, \qquad \text{Var}_{\mathcal{N}}[A_{\text{sparse}}] = \frac{mb - s^\star}{(mb)^2}\sigma^2.$$

Moreover, the corresponding signal-to-noise ratios are

$$\text{SNR}_{\text{dense}} = \frac{s^\star |\mu_{\mathcal{S}}|}{\sigma\sqrt{D - s^\star}},$$

and, for $mb > s^\star$,

$$\text{SNR}_{\text{sparse}} = \frac{s^\star |\mu_{\mathcal{S}}|}{\sigma\sqrt{mb - s^\star}}.$$

Thus, when $\mathcal{T}$ covers the informative features while excluding most background dimensions, sparse aggregation improves SNR by a factor of

$$\frac{\mathrm{SNR}_{\mathrm{sparse}}}{\mathrm{SNR}_{\mathrm{dense}}} = \sqrt{\frac{D - s^\star}{mb - s^\star}}, \qquad mb > s^\star.$$

In the ideal case $mb = s^\star$, the sparse support contains no background noise, so the noise contribution vanishes under this simplified model.

Proof. For dense aggregation,

$$A_{\mathrm{dense}}(\mathbf{x}) = \frac{1}{D}\sum_{d\in\mathcal{S}} x_d + \frac{1}{D}\sum_{d\in\mathcal{N}} x_d.$$

Treating the informative features as fixed, the variance comes only from the noise term:

$$\mathrm{Var}_{\mathcal{N}}[A_{\mathrm{dense}}] = \mathrm{Var}\left[\frac{1}{D}\sum_{d\in\mathcal{N}} x_d\right] = \frac{1}{D^2}\sum_{d\in\mathcal{N}} \mathrm{Var}(x_d) = \frac{D - s^\star}{D^2}\sigma^2.$$

The signal component of dense aggregation is

$$\mathbb{E}_{\mathcal{N}}[A_{\mathrm{dense}}] = \frac{1}{D}\sum_{d\in\mathcal{S}} x_d = \frac{s^\star}{D}\mu_{\mathcal{S}}.$$

Hence

$$\mathrm{SNR}_{\mathrm{dense}} = \frac{|s^\star\mu_{\mathcal{S}}/D|}{\sigma\sqrt{D - s^\star}/D} = \frac{s^\star|\mu_{\mathcal{S}}|}{\sigma\sqrt{D - s^\star}}.$$

For sparse aggregation, since $\mathcal{S} \subseteq \mathcal{T}$ and $|\mathcal{T}| = mb$, the number of background noise features included in $\mathcal{T}$ is $mb - s^\star$. Therefore,

$$\mathrm{Var}_{\mathcal{N}}[A_{\mathrm{sparse}}] = \mathrm{Var}\left[\frac{1}{mb}\sum_{d\in\mathcal{T}\cap\mathcal{N}} x_d\right] = \frac{mb - s^\star}{(mb)^2}\sigma^2.$$

The signal component is

$$\mathbb{E}_{\mathcal{N}}[A_{\mathrm{sparse}}] = \frac{1}{mb}\sum_{d\in\mathcal{S}} x_d = \frac{s^\star}{mb}\mu_{\mathcal{S}}.$$

For $mb > s^\star$, this yields

$$\mathrm{SNR}_{\mathrm{sparse}} = \frac{|s^\star\mu_{\mathcal{S}}/mb|}{\sigma\sqrt{mb - s^\star}/mb} = \frac{s^\star|\mu_{\mathcal{S}}|}{\sigma\sqrt{mb - s^\star}}.$$

Taking the ratio gives

$$\frac{\mathrm{SNR}_{\mathrm{sparse}}}{\mathrm{SNR}_{\mathrm{dense}}} = \sqrt{\frac{D - s^\star}{mb - s^\star}}.$$

If $mb = s^\star$, then $\mathcal{T}$ contains no background noise features, so the noise variance of A_{sparse} is zero. This completes the proof. □

Corollary B.3 (Scaling with Feature Dimension)**.** *Fix the number of informative features $s^\star$ and let the sparse budget be $mb = s^\star + \delta$ for a fixed $\delta \geq 0$. As $D \to \infty$, the dense aggregation SNR satisfies*

$$\mathrm{SNR}_{\mathrm{dense}} = \frac{s^\star|\mu_{\mathcal{S}}|}{\sigma\sqrt{D - s^\star}} = \mathcal{O}(D^{-1/2}),$$

whereas, for $\delta > 0$,

$$\mathrm{SNR}_{\mathrm{sparse}} = \frac{s^\star|\mu_{\mathcal{S}}|}{\sigma\sqrt{\delta}} = \Theta(1).$$

If $\delta = 0$, the sparse support contains no background noise and the noise contribution vanishes under the simplified model. Thus, dense aggregation suffers from signal dilution as the ambient feature dimension grows, whereas sparse aggregation preserves a dimension-independent SNR when the sparse support remains tight.

Proof. The dense SNR expression follows directly from Proposition B.2:

$$\mathrm{SNR}_{\mathrm{dense}} = \frac{s^\star |\mu_{\mathcal{S}}|}{\sigma\sqrt{D - s^\star}} = \mathcal{O}(D^{-1/2})$$

for fixed $s^\star$, $\mu_{\mathcal{S}}$, and σ. For sparse aggregation with $mb = s^\star + \delta$ and $\delta > 0$,

$$\mathrm{SNR}_{\mathrm{sparse}} = \frac{s^\star |\mu_{\mathcal{S}}|}{\sigma\sqrt{mb - s^\star}} = \frac{s^\star |\mu_{\mathcal{S}}|}{\sigma\sqrt{\delta}},$$

which is independent of D. When $\delta = 0$, the sparse aggregation contains no background noise, so its noise variance is zero under the assumptions of the proposition. □

Remark B.4 (Extension to Softmax Attention Weights). The uniform-weight analysis provides a tractable baseline. For softmax attention with learned queries and keys, the SNR behavior depends on the model's ability to assign small weights to background features and larger weights to informative features. If the attention mechanism concentrates most mass on $\mathcal{S}$, then the effective support

$$\mathcal{T}_{\mathrm{eff}} = \{d : \alpha_d > \varepsilon\}$$

can approach the idealized sparse case. This motivates structurally sparse mechanisms, which encourage restricted feature interaction patterns rather than relying solely on learned dense softmax weights to become sparse in high-dimensional noisy settings.

Remark B.5 (Connection to Empirical Results). The conditional analysis in Proposition B.2 and Corollary B.3 provides intuition for the empirical findings in Appendix M.2 and Figure 12:

1. **Dense attention.** In high-dimensional settings, dense attention mixes informative and background features through all-pair interactions. The analysis suggests that, when only a small subset of features is informative, dense aggregation can dilute the useful signal as D grows.

2. **Mamba.** SSM-based mixing provides linear-time propagation along the feature axis, but it does not explicitly enforce sparse feature selection. As a result, it may propagate both informative and background dimensions, yielding intermediate performance.

3. **ASIM.** Structured sparse selection encourages the model to restrict feature interactions to a smaller support. When this support covers informative features while excluding many irrelevant dimensions, the analysis predicts improved SNR. This provides theoretical intuition for why the ASIM design can improve robustness in high-dimensional tabular regression.

B.3 Properties of GridLoss

Lemma B.6 (Monotonicity of the GridLoss soft index)**.** *Fix ordered grid anchors $e_0 < e_1 < \cdots < e_K$ and temperature $\tau > 0$. For any scalar $x \in \mathbb{R}$, define*

$$w_k(x) \;=\; \frac{\exp(-|x - e_k|/\tau)}{\sum_{j=0}^{K} \exp(-|x - e_j|/\tau)}, \qquad b(x) \;=\; \sum_{k=0}^{K} k\, w_k(x).$$

Then $b(x)$ is nondecreasing in x; equivalently, $x_1 \le x_2$ implies $b(x_1) \le b(x_2)$.

Proof. Let $z_k(x) = -|x - e_k|/\tau$ and $w_k(x) = \mathrm{softmax}(z(x))_k$. For $x \notin \{e_0, \ldots, e_K\}$, we have

$$z_k'(x) = -\frac{1}{\tau}\,\mathrm{sign}(x - e_k).$$

The softmax derivative gives

$$w_k'(x) = w_k(x)\left(z_k'(x) - \sum_{j=0}^{K} w_j(x) z_j'(x)\right).$$

Therefore,

$$b'(x) = \sum_{k=0}^{K} k\, w_k'(x) = -\frac{1}{\tau}\left(\mathbb{E}_w[k s_k] - \mathbb{E}_w[k]\mathbb{E}_w[s_k]\right) = -\frac{1}{\tau}\operatorname{Cov}_w(k, s_k),$$

where $s_k = \operatorname{sign}(x - e_k)$ and $\mathbb{E}_w[\cdot]$ denotes expectation under the weights $\{w_k(x)\}_{k=0}^K$. Since e_k increases with k, the sequence $\{s_k\}_{k=0}^K$ is nonincreasing in k. Hence $\operatorname{Cov}_w(k, s_k) \le 0$, which implies $b'(x) \ge 0$ wherever the derivative exists. At anchor locations $x = e_k$, monotonicity follows by continuity and one-sided derivatives. □

Lemma B.7 (Bounded slope and stable GridLoss gradients)**.** *Under the same definitions as Lemma B.6, for all $x \notin \{e_0, \dots, e_K\}$,*

$$0 \le b'(x) \le \frac{K}{2\tau}.$$

Consequently, for the per-sample GridLoss term

$$\ell_{\mathrm{Grid}}(\hat{y}, y) = \big|b(\hat{y}) - b(y)\big|,$$

any subgradient $g \in \partial_{\hat{y}} \ell_{\mathrm{Grid}}(\hat{y}, y)$ satisfies

$$|g| \le \frac{K}{2\tau}.$$

Proof. From the proof of Lemma B.6,

$$b'(x) = -\frac{1}{\tau}\operatorname{Cov}_w(k, s_k), \qquad s_k \in \{-1, +1\}.$$

By Cauchy–Schwarz,

$$|\operatorname{Cov}_w(k, s_k)| \le \sqrt{\operatorname{Var}_w(k)\operatorname{Var}_w(s_k)} \le \sqrt{\operatorname{Var}_w(k)},$$

since $\operatorname{Var}_w(s_k) \le 1$. Because $k \in \{0, 1, \dots, K\}$, Popoviciu's inequality gives

$$\operatorname{Var}_w(k) \le \frac{K^2}{4}.$$

Therefore,

$$0 \le b'(x) \le \frac{K}{2\tau}.$$

For $\ell_{\mathrm{Grid}}(\hat{y}, y) = |b(\hat{y}) - b(y)|$, the chain rule for subgradients yields

$$\partial_{\hat{y}} \ell_{\mathrm{Grid}}(\hat{y}, y) \subseteq \operatorname{sign}\big(b(\hat{y}) - b(y)\big) \cdot \partial_{\hat{y}} b(\hat{y}),$$

so every subgradient satisfies $|g| \le K/(2\tau)$. □

Lemma B.8 (Lipschitz continuity of the soft index)**.** *Under the conditions of Lemma B.7, for all $x_1, x_2 \in \mathbb{R}$,*

$$|b(x_1) - b(x_2)| \le \frac{K}{2\tau}|x_1 - x_2|.$$

Proof. On intervals not containing grid anchors, the result follows from the mean value theorem and the slope bound in Lemma B.7. Since b is continuous at the anchors, the bound extends to arbitrary $x_1, x_2 \in \mathbb{R}$ by partitioning the interval between x_1 and x_2 at any anchors it contains and summing the bounds over the subintervals. □

B.4 Asymptotic Consistency of GridLoss

We show that normalized GridLoss recovers the mean absolute error in a high-resolution, low-temperature limit. This provides a link between GridLoss and classical robust regression losses.

Fixed-interval analysis versus batch-adaptive implementation. For theoretical clarity, the analysis assumes a fixed interval $[\alpha, \beta]$. In implementation, the grid interval is batch-adaptive: α and β are computed from the current mini-batch, for example using the batch extrema of y and $\hat{y}$, and are detached from the computation graph. The analysis applies conditionally within each batch-defined interval because the key arguments depend on the local grid geometry and the relative distances to nearby anchors.

Theorem B.9 (GridLoss Consistency)**.** *Let $\mathcal{L}_{\text{Grid}}^{(K,\tau)}(\hat{y}, y)$ denote GridLoss with K bins and temperature τ on a fixed interval $[\alpha, \beta]$. Define the normalized GridLoss*

$$\widetilde{\mathcal{L}}_{\text{Grid}}^{(K,\tau)}(\hat{y}, y) = \frac{\beta - \alpha}{K} \mathcal{L}_{\text{Grid}}^{(K,\tau)}(\hat{y}, y).$$

Then, for any $\hat{y}, y \in [\alpha, \beta]$,

$$\lim_{\tau \to 0^+} \lim_{K \to \infty} \widetilde{\mathcal{L}}_{\text{Grid}}^{(K,\tau)}(\hat{y}, y) = |\hat{y} - y|.$$

That is, normalized GridLoss converges to the ℓ_1 loss as the grid becomes infinitely fine and the soft assignments become sharp.

Proof. Let

$$\Delta = \frac{\beta - \alpha}{K}, \qquad e_k = \alpha + k\Delta, \qquad k = 0, \ldots, K.$$

For a scalar $x \in [\alpha, \beta]$, define the soft index

$$b^{(K,\tau)}(x) = \sum_{k=0}^{K} k\, w_k(x), \qquad w_k(x) = \frac{\exp(-|x - e_k|/\tau)}{\sum_{j=0}^{K} \exp(-|x - e_j|/\tau)}.$$

The normalized soft coordinate is $\Delta b^{(K,\tau)}(x)$. As $K \to \infty$ with τ fixed, the discrete grid becomes dense and the weighted sum converges to the expectation of a truncated Laplace-type density on $[\alpha, \beta]$ centered at x:

$$\Delta b^{(K,\tau)}(x) \longrightarrow B_\tau(x) := \frac{\int_\alpha^\beta u \exp(-|u - x|/\tau)\, du}{\int_\alpha^\beta \exp(-|u - x|/\tau)\, du}.$$

The quantity $B_\tau(x)$ is a smoothed version of x. Since the exponential kernel concentrates around x as $\tau \to 0^+$, we have

$$B_\tau(x) \to x \qquad \text{for every } x \in [\alpha, \beta].$$

Therefore,

$$\lim_{\tau \to 0^+} \lim_{K \to \infty} \Delta b^{(K,\tau)}(x) = x.$$

Applying this to $\hat{y}$ and y gives

$$\lim_{\tau \to 0^+} \lim_{K \to \infty} \widetilde{\mathcal{L}}_{\text{Grid}}^{(K,\tau)}(\hat{y}, y) = \lim_{\tau \to 0^+} |B_\tau(\hat{y}) - B_\tau(y)| = |\hat{y} - y|,$$

which completes the proof. □

Corollary B.10 (Approximation Behavior)**.** *On a fixed interval $[\alpha, \beta]$, normalized GridLoss approaches an MAE-like ordinal distance as the grid resolution increases and the temperature decreases. Finer grids reduce discretization error, while finite temperature smooths the hard binning operation and yields bounded gradients. Thus, K controls target-scale resolution, whereas τ controls the smoothness–sharpness trade-off.*

Proof. The convergence to MAE follows from Theorem B.9. The role of finite temperature follows from Lemma B.7, which bounds the gradient magnitude by $K/(2\tau)$ for fixed K and τ. Thus, larger K gives a finer discretization of the target interval, while τ controls how sharply each scalar value is assigned to nearby grid anchors. □

Remark B.11 (Relationship to Soft Label Regression). Theorem B.9 positions GridLoss within a broader family of soft-binning regression losses. Related approaches include:

- **Label Distribution Learning (LDL):** LDL represents targets as distributions over discrete labels; GridLoss can be viewed as using Gibbs-type soft assignments centered at $\hat{y}$ and y.
- **Ordinal Regression:** Ordinal regression methods often enforce rank consistency through hard or probabilistic class decompositions. GridLoss instead uses soft indices to provide a differentiable relaxation of ordinal positioning for continuous targets.
- **Histogram or Distributional Losses:** Histogram losses and Earth Mover's Distance compare target distributions. GridLoss compares each prediction–target pair through its soft grid index, yielding a sample-wise ordinal distance.

The consistency result shows that GridLoss approaches MAE in the high-resolution, low-temperature limit while maintaining smooth gradients for finite τ.

Remark B.12 (Practical Implications). Theorem B.9 has several practical implications:

1. **Hyperparameter guidance:** Increasing K improves target-scale resolution, but very large K with very small τ can increase gradient magnitudes, as Lemma B.7 shows. Thus, K and τ should be selected jointly to balance resolution and stability.
2. **MAE-like behavior:** Since normalized GridLoss approaches MAE in the high-resolution, low-temperature limit, it inherits an MAE-like robustness profile while retaining smooth gradients at finite temperature.
3. **Ordinal structure:** Unlike standard pointwise losses, GridLoss explicitly operates in a discretized ordinal space, encouraging predictions to respect relative placement along the target range.

Proposition B.13 (GridLoss as Interpolation)**.** *For fixed K and varying τ, GridLoss interpolates between:*

1. *As $\tau \to \infty$, $w_k(x) \to 1/(K+1)$ for all k, so $b(x) \to K/2$ and $\mathcal{L}_{\text{Grid}}^{(K,\tau)}(\hat{y}, y) \to 0$ for all $\hat{y}, y$.*
2. *As $\tau \to 0^+$, the weights concentrate on the nearest anchor, so $\mathcal{L}_{\text{Grid}}^{(K,\tau)}(\hat{y}, y)$ converges to a hard ordinal bin distance.*

As K increases, the grid provides a finer discretization of the target interval. Under the high-resolution and low-temperature limit in Theorem B.9, the normalized GridLoss converges to MAE.

Proof. As $\tau \to \infty$, each exponential term satisfies $\exp(-|x - e_k|/\tau) \to 1$, so $w_k(x) \to 1/(K+1)$ for all k. Hence

$$b(x) \to \frac{1}{K+1} \sum_{k=0}^{K} k = \frac{K}{2}$$

for all x, and the index distance between any $\hat{y}$ and y converges to zero.

As $\tau \to 0^+$, the weights concentrate on the nearest grid anchor. Thus $b(x)$ converges to the nearest-anchor index, and GridLoss becomes the absolute distance between hard ordinal bin indices. Finally, the convergence of normalized GridLoss to MAE under joint high-resolution and low-temperature behavior follows from Theorem B.9. □

B.5 Objective Interpretation

Proposition B.14 (Objective Interpretation of RISE as Reweighted Empirical Risk)**.** *Let $\{(x_i, y_i)\}_{i=1}^N$ be the training set and let $\ell_i(\theta)$ denote the per-sample loss at parameters θ, e.g., $\ell_i(\theta) = \ell(f_\theta(x_i), y_i)$. At RISE refresh step e, define a positive weight*

$$w_i^{(e)} = \varphi(q_i^{(e)}) > 0,$$

where $q_i^{(e)} \in \{0, \dots, G-1\}$ is the empirical loss-quantile bin index of sample i, computed from $\{\ell_j(\theta_e)\}_{j=1}^N$, and $\varphi(\cdot)$ is a monotone increasing bin-to-weight map. Consider SGD where each

mini-batch is drawn by weighted sampling using probabilities

$$p_i^{(e)} = \frac{w_i^{(e)}}{\sum_{j=1}^{N} w_j^{(e)}},$$

and the update uses the uncorrected stochastic gradient $\nabla_\theta \ell_i(\theta)$ of the sampled example, without inverse-probability correction. Conditional on the weights $\{w_i^{(e)}\}$, the expected stochastic gradient equals the gradient of the reweighted empirical risk

$$\mathcal{L}_{\mathrm{RISE}}^{(e)}(\theta) = \sum_{i=1}^{N} p_i^{(e)} \ell_i(\theta) = \frac{\sum_{i=1}^{N} w_i^{(e)} \ell_i(\theta)}{\sum_{j=1}^{N} w_j^{(e)}}.$$

That is,

$$\mathbb{E}_{i \sim p^{(e)}}[\nabla_\theta \ell_i(\theta)] = \nabla_\theta \mathcal{L}_{\mathrm{RISE}}^{(e)}(\theta),$$

where the weights are treated as fixed between refresh steps. Therefore, RISE can be interpreted as following stochastic gradients of a sequence of reweighted empirical risks that emphasize higher-loss quantile bins when $\varphi(\cdot)$ is increasing.

Proof. Fix a refresh step e and condition on the weights $\{w_i^{(e)}\}$ and probabilities $\{p_i^{(e)}\}$. A single-sample stochastic gradient drawn by weighted sampling satisfies

$$\mathbb{E}_{i \sim p^{(e)}}[\nabla_\theta \ell_i(\theta)] = \sum_{i=1}^{N} p_i^{(e)} \nabla_\theta \ell_i(\theta).$$

Between refresh steps, the sampling weights are fixed with respect to the current optimization variable θ. Hence

$$\sum_{i=1}^{N} p_i^{(e)} \nabla_\theta \ell_i(\theta) = \nabla_\theta \left(\sum_{i=1}^{N} p_i^{(e)} \ell_i(\theta) \right) = \nabla_\theta \mathcal{L}_{\mathrm{RISE}}^{(e)}(\theta).$$

Substituting

$$p_i^{(e)} = \frac{w_i^{(e)}}{\sum_{j=1}^{N} w_j^{(e)}}$$

gives the normalized reweighted empirical risk. If $\varphi(\cdot)$ is increasing in the loss-quantile bin, higher-loss samples receive larger sampling probabilities, so the resulting stochastic gradients emphasize the tail of the empirical loss distribution. □

C Implementation and Training Details

This appendix provides additional implementation details to support reproducibility of the results presented in the main paper. Unless otherwise stated, all experiments follow the same data splits, preprocessing pipeline, model-selection criterion, and training protocol described in the main text.

C.1 Training Loop Overview

All models are trained using mini-batch stochastic optimization. Each training epoch consists of forward passes, loss computation, backpropagation, and parameter updates. Validation, when applicable, is performed at the end of each epoch and is used for learning-rate scheduling, early stopping, and model checkpoint selection.

Algorithm 1 presents the training loop with RISE sampling. For clarity, the algorithm focuses on the RISE refresh mechanism; validation, learning-rate scheduling, and early stopping are described in the following subsections.

Algorithm 1 Training Loop with RISE Sampling

Require: Training set $\mathcal{D}_{\text{train}} = \{(x_i, y_i)\}_{i=1}^{N}$, model f_θ, per-sample loss $\ell(\cdot, \cdot)$, total epochs E, number of bins G, boost factor β_{RISE}, refresh interval k, batch size B
Ensure: Trained model parameters θ

Initialize: θ, optimizer, and sampling weights $w_i \leftarrow 1$ for all $i \in \{1, \dots, N\}$
if $k > 0$ **then** ▷ Initial difficulty estimation; evaluation mode, no gradient tracking
  Set model to evaluation mode and disable gradient tracking
  $\ell_i \leftarrow \ell(f_\theta(x_i), y_i)$ for each $(x_i, y_i) \in \mathcal{D}_{\text{train}}$
  Re-enable gradient tracking and set model back to training mode
  $\{g_i\}_{i=1}^{N} \leftarrow \text{QuantileBins}(\{\ell_i\}_{i=1}^{N}, G)$
  $w_i \leftarrow 1 + g_i \cdot \frac{\beta_{\text{RISE}} - 1}{G - 1}$ for all i
end if
Initialize `WeightedSampler` using $\{w_i\}_{i=1}^{N}$ ▷ Sampling only; no importance correction
for $e = 1$ to E **do**
  for all mini-batches (x_b, y_b) of size B sampled from $\mathcal{D}_{\text{train}}$ via `WeightedSampler` **do**
    $\hat{y}_b \leftarrow f_\theta(x_b)$
    $\mathcal{L}_b \leftarrow \frac{1}{B} \sum_{(x_i, y_i) \in (x_b, y_b)} \ell(\hat{y}_i, y_i)$
    Backpropagate $\mathcal{L}_b$ and update θ
  end for
  if $k > 0$ **and** $e \bmod k = 0$ **then** ▷ Periodic RISE refresh; evaluation mode, no gradient tracking
    Set model to evaluation mode and disable gradient tracking
    $\ell_i \leftarrow \ell(f_\theta(x_i), y_i)$ for each $(x_i, y_i) \in \mathcal{D}_{\text{train}}$
    Re-enable gradient tracking and set model back to training mode
    $\{g_i\}_{i=1}^{N} \leftarrow \text{QuantileBins}(\{\ell_i\}_{i=1}^{N}, G)$
    $w_i \leftarrow 1 + g_i \cdot \frac{\beta_{\text{RISE}} - 1}{G - 1}$ for all i
    Update `WeightedSampler` weights using $\{w_i\}_{i=1}^{N}$
  end if
end for
return θ

C.2 Optimizer Configurations

C.2.1 Hyperparameter Optimization with Optuna

We use **Optuna** [49] to tune the sparse attention module while keeping the training protocol fixed. The search space includes sparse attention architectural parameters, including the number of heads, head dimension, sliding-window size, block-selection size, compression block size, and number of selected blocks, as well as a small set of training hyperparameters such as learning rate and batch size. Each trial uses the same data split, loss function, and early-stopping criterion to ensure fair comparison. The optimization objective is validation performance, measured by validation loss or a validation regression metric such as RMSE.

C.2.2 Optimizers

We evaluate several optimizers under a unified training framework with consistent early stopping, model selection, and, when applicable, RISE-based sampling. All optimizers operate on the same model parameters without architectural modification:

- **AdamW**: AdamW is the default optimizer for all datasets. We use decoupled weight decay to separate regularization from the adaptive gradient update.
- **LARS** [54]: LARS is used in selected large-scale settings to stabilize optimization when layer-wise parameter norms vary substantially. For a parameter tensor p with gradient g, we compute the trust ratio

$$q = \eta \frac{\|p\|_2}{\|g\|_2 + \epsilon},$$

and apply momentum updates

$$\mu \leftarrow m\mu + q(g + \lambda p), \qquad p \leftarrow p - \alpha_{\mathrm{LARS}}\mu,$$

where α_{LARS} is the learning rate, m is the momentum coefficient, and λ is the weight decay factor. Bias and normalization parameters are excluded from both weight decay and LARS adaptation. We use an epoch-aware interface to synchronize LARS with curriculum and learning-rate scheduling.

- **Stabilized AdamW variants**: These optimizers retain AdamW's update structure while damping the effective step size for improved robustness under noisy gradients and long training horizons. Updates take the form

$$p \leftarrow p - \eta_t \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon},$$

where $\hat{m}_t$ and $\hat{v}_t$ are bias-corrected moment estimates, and η_t is dynamically rescaled using statistics of the denominator.

  - **StableAdamW**: rescales η_t using debiased second-moment statistics to avoid overly large updates.
  - **StableAdamWUnfused**: additionally applies tensor-wise update clipping by scaling η_t using the ratio between gradient magnitude and estimated variance.
- **Second-order optimization with L-BFGS**: L-BFGS is evaluated in selected experiments to study curvature-aware updates. We use a limited-memory configuration with fixed history size and strong Wolfe line search. Optimization is performed with mini-batches; gradients are clipped to an ℓ_2 norm of 1.0, and automatic mixed precision is disabled due to repeated closure evaluations.

Unless otherwise stated, optimizer hyperparameters are selected using the validation split, or via Optuna when applicable, and then held fixed across model variants to ensure fair comparison.

C.3 Learning-Rate Schedules

For first-order optimizers, we use a schedule that combines **linear warmup** with **cosine annealing**. Let η denote the base learning rate, T the total number of training epochs, w the warmup duration in epochs, and $\eta_{\min}$ the minimum learning rate. If $w = 0$, we apply cosine annealing over all T epochs without warmup. Otherwise, the learning rate at epoch t is

$$\eta_t = \begin{cases} \eta_{\min} + (\eta - \eta_{\min}) \cdot \frac{t}{w}, & 0 \le t < w, \\ \eta_{\min} + \frac{\eta - \eta_{\min}}{2} \left(1 + \cos\left(\pi \frac{t-w}{T-w}\right)\right), & w \le t \le T. \end{cases}$$

We set $\eta_{\min} = \eta/10$ by default. Warmup mitigates early training instability, after which cosine decay smoothly anneals the learning rate over the remaining epochs. Learning-rate updates are applied once per epoch.

Reduce-on-Plateau. In validation-driven settings, we alternatively use `ReduceLROnPlateau`[1], which reduces the learning rate when the validation loss does not improve for a fixed patience window. The scheduler is updated once per epoch using validation metrics and integrates naturally with early stopping.

C.4 Automatic Mixed Precision

We employ **automatic mixed-precision (AMP)** training for first-order optimizers to accelerate computation and reduce GPU memory usage. AMP uses mixed-precision forward passes with dynamic loss scaling to preserve numerical stability.

We disable AMP for **L-BFGS** due to its repeated closure evaluations within each update step. Reduced-precision arithmetic in this setting can cause numerical instability and NaN gradients.

[1] `https://docs.pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.ReduceLROnPlateau.html` (Accessed September 2025)

C.5 Early Stopping and Model Selection

Training is conducted for up to $E_{\max}$ epochs with early stopping based on validation loss. If the validation loss fails to improve for a predefined patience window, training is terminated early. We select the checkpoint with the lowest validation loss for final evaluation, providing a consistent model-selection criterion across datasets with different noise levels and difficulty.

C.6 RISE Sampling Details

RISE (Reweighted Instance Sampling by Error) is implemented as a weighted sampling strategy that dynamically adjusts mini-batch composition based on per-sample difficulty (Section 3.6). RISE modifies only the sampling distribution and does not alter loss computation, gradient evaluation, or optimizer update rules; thus, all optimizers and schedulers described above are compatible with RISE.

Per-sample losses for RISE binning are computed in evaluation mode without gradient tracking for efficiency and stability. The sampler is refreshed periodically to adapt to the changing difficulty distribution during training. In our experiments, we use $G = 4$ difficulty bins, a boost factor $\beta_{\text{RISE}} = 2.0$, and refresh the sampler every $k = 1$ epoch.

C.7 Batch Size, Epochs, and Gradient Clipping

Batch size is selected per dataset based on memory constraints and validation performance. All models are trained for up to a fixed maximum number of epochs. To stabilize training, particularly for second-order optimization, we apply **gradient clipping** with a maximum ℓ_2 norm of 1.0. This is applied uniformly across all optimizers unless otherwise specified.

Final Training Protocol. After fixing the sparse attention architecture via Optuna, we retrain the full model using the training strategies described in Section 3.6 and in this appendix, including RISE-based sampling, learning-rate scheduling, automatic mixed precision when applicable, and early stopping. No further architectural search is performed beyond this stage, ensuring that performance gains are not attributable to repeated architecture search or additional post-hoc hyperparameter tuning. Preprocessing steps, including feature scaling, categorical encoding, and text embedding, are fitted exclusively on the training split to prevent data leakage.

Hardware Setup. Experiments were conducted on a single NVIDIA A100 GPU with 40 GB VRAM, paired with an AMD EPYC 7742 CPU and 512 GB RAM. The software environment uses PyTorch 2.1 and CUDA 12.2.

D Full Benchmark Results and Statistical Analyses

D.1 Full Benchmark Results

Table 3 extends the summary in Table 1 with per-model RMSE across all nine benchmarks. This table compares against classical baselines [45], tree-based ensembles [55; 4; 5; 6; 56], MLP-style architectures [9; 57; 58; 59], transformer-based models [29; 60; 10; 61; 62; 50], retrieval-based methods [63; 64; 23; 65], other deep tabular architectures [66; 67; 68; 1; 69; 70; 71; 72], and tabular foundation models (pretrained in-context or non-parametric inference) [25; 14; 73; 17; 15; 18]. TabDPT [17], TabICLv2 [16] and TabFM [18] are evaluated here under the same protocol as every other row (identical splits, seed 42), each with its released default configuration and no tuning: TabDPT with a 2,048-row context and 8-member ensemble, TabICLv2 with 8 estimators, and TabFM (v1.0.0, PyTorch weights) with the empty configuration used by TabArena. TabFM exceeds the 80 GB GPU on CD, ChD, EVP and RES (‡). AMFormer [50] is a supervised architecture trained per dataset and is grouped with the transformer-based models. The baseline denoted EBM is the Explainable Boosting Machine [74], a glass-box generalized additive boosting model; since it is neither pretrained nor used for in-context inference, it is grouped with the other deep tabular architectures rather than the foundation models. The TabPFNv2 entry on AQ (13,580 training rows, above the model's 10,000-sample limit) is likewise omitted because the exact invocation could not be recovered. Symbols indicate runs that could not be completed due to time constraints (†), GPU

Table 3: Evaluation of different models on regression tasks. Results report RMSE (lower is better).

	Model	SA	CP	AQ	TO	SC	CD	ChD	EVP	RES
Classical	LinearRegression	0.402	9.277	15.062	0.028	3.997	0.236	8717.544	35.156	3163
	KNN-5	0.238	3.719	12.816	0.029	3.602	1.347	15860.523	26.104	3212
	Lasso	0.402	9.277	15.064	0.028	3.964	0.251	8717.439	35.155	3163
	Ridge	0.402	9.277	15.062	0.027	3.982	0.236	8717.479	35.156	3163
	SVR-Linear	0.406	11.663	17.854	0.042	3.974	♢, †	♢, †	♢, †	♢, †
	SVR-RBF	0.175	7.491	19.416	0.038	3.964	∗	∗	∗	∗
Tree-based	Random Forest	0.201	2.842	9.095	0.027	4.092	0.733	17525	†	†
	CatBoost	0.171	2.739	7.692	0.027	3.971	0.009	19000	3.291	3144
	LightGBM	0.174	2.680	7.902	0.027	4.639	0.043	18688	2.799	3135
	XGBoost	0.197	2.690	11.583	0.028	4.116	0.040	18188	2.894	3186
	ExtraTrees	0.182	2.617	5.869	0.027	3.455	**0.004**	22882	2.809	46267
MLP-style	MLP	0.181	2.939	8.967	0.028	3.920	0.046	4963	1.847	3161
	ResNet	0.167	2.920	4.578	0.028	3.948	0.028	13022	1.556	3163
	MLP-PLR	0.192	2.925	5.296	0.028	3.966	0.143	18476	3.222	40947
	TabM	0.269	3.139	8.862	0.028	3.966	0.095	16032	2.101	40941
	T2G-Former	0.192	2.937	5.358	0.028	3.962	‡	‡	1.867	21058
Transformer	TabTransformer	0.298	8.605	4.189	0.028	3.866	0.046	7188	2.338	3936
	SAINT	0.156	2.901	‡	‡	‡	‡	‡	‡	‡
	FT-Transformer	0.192	2.924	7.330	0.029	3.967	0.014	8940	1.507	3163
	AutoInt	0.213	3.050	6.969	0.028	3.919	0.007	4999	2.587	3163
	DCN V2	0.165	8.133	4.081	0.038	3.751	§	4475	1.992	6667
	ExcelFormer	0.165	2.901	8.088	0.028	3.667	‡	‡	1.608	‡
	AMFormer	0.205	2.980	16.363	0.028	3.964	‡	17858	‡	21149
Retrieval	TabR	0.150	2.867	5.062	0.029	3.967	‡	‡	‡	‡
	ModernNCA	0.167	2.833	12.186	0.028	3.730	‡	‡	‡	‡
	NODE	0.270	7.150	15.575	0.028	3.966	0.052	22661	5.157	3187
	TabNet	0.279	3.346	8.615	0.028	3.454	0.284	24271	2.653	4152
Other	SNN	2.453	14.809	17.833	0.028	3.932	5.276	18759	71.000	3163
	TANGOS	0.181	2.991	4.690	0.028	3.953	0.136	7298	3.219	4094
	PTaRL	0.251	3.183	5.201	0.028	3.570	0.196	7865	4.157	4082
	DANets	0.289	3.256	5.148	0.028	3.970	0.606	4808	9.367	41321
	GrowNet	2.141	16.855	20.089	0.028	3.982	§	25264	13.448	124527
	DNNR	0.149	3.065	§	§	§	†	†	†	†
	SwitchTab	0.457	3.884	6.384	0.029	3.969	0.285	17978	6.227	4083
	GRANDE	0.347	11.716	23.815	0.028	3.964	4.353	25440	69.129	25620
	EBM	0.178	2.740	6.748	0.027	3.522	♢	♢	♢	♢
Found.	TabPFNv2	0.168	3.006	∗	0.027	3.965	∗	∗	∗	∗
	TabPTM	0.579	4.015	18.627	0.029	3.827	5.253	25630	65.471	4171
	TabDPT	0.184	2.675	11.137	0.027	4.047	0.675	23006	8.353	3164
	TabICLv2	0.118	2.505	6.421	0.027	3.858	0.007	22771	1.897	3164
	TabFM	**0.113**	**2.404**	**3.577**	0.027	3.437	‡	‡	‡	‡
	Ours	0.137	2.842	3.999	**0.021**	**3.322**	0.007	**4460**	**1.199**	**3103**

† Not reported due to time constraints (24-hour limit); runs exceeding this duration were terminated.
‡ Not reported due to GPU memory limitations (GPU capacity: 79.21 GiB); models exceeding this requirement could not be executed.
§ Training diverged; the model produced losses orders of magnitude outside the comparable range and is treated as non-convergent.
∗ No TabPFNv2 entry. TabPFNv2 supports at most 10,000 training rows and 500 features. We evaluated it via the TALENT wrapper [51] with 8 ensemble members, which fits each member on a random subsample of at most 10,000 rows; this modified protocol produced the SA result (over 30,000 training rows). We omit subsampled results for CD, ChD, EVP, and RES (190,000–755,000 training rows), where 10,000 rows represent only 1–5% of the training data, as these results would not reflect performance on the full tables. Without subsampling, TabPFNv2 runs out of memory on CD, ChD, and EVP (Appendix J). The AQ result (13,580 training rows) is omitted for the reason stated in the main text.
♢ Infeasible at dataset scale (algorithm complexity prohibitive).
All entries are single runs; see Appendix I for TabNSM's multi-seed variability. TabDPT, TabICLv2 and TabFM were run on a single 80 GB H100.

memory limitations (‡), training divergence (§), hard-coded dataset size restrictions (∗), or prohibitive algorithm complexity (♢). TabNSM achieves the lowest RMSE on five of nine benchmarks (TO, SC, ChD, EVP, RES); TabFM is best on SA, CP and AQ, with TabICLv2 second on SA and CP.

D.2 Statistical Significance of Results

We assess whether TabNSM's improvements over baselines are statistically significant using two complementary non-parametric tests [75].

Friedman Test We first test the global null hypothesis that all models perform equally across benchmarks. For each benchmark we assign integer ranks (1 = lowest RMSE = best), then apply the Friedman test to the resulting $k \times N$ rank matrix. We use the $k = 39$ models that completed all five benchmarks with reliable results (SA, CP, AQ, TO, SC), giving $N = 5$; the three excluded models are SAINT and DNNR, which diverged or ran out of memory on three of the five, and TabPFNv2, whose AQ entry is not reported. The Friedman statistic is $\chi^2(38) = 92.3$, $p = 2.04 \times 10^{-6}$, rejecting the null hypothesis and confirming that performance differences across models are not due to chance. Figure 4 shows the resulting Critical Difference diagram.

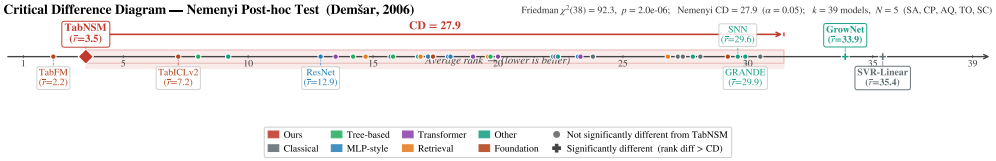


Figure 4: Critical Difference diagram (Nemenyi post-hoc test, $\alpha = 0.05$). Each dot is a model colored by architectural family; its horizontal position is its average rank across five benchmarks (lower = better). The CD bracket shows the minimum rank difference required for statistical significance (CD = 27.9). Models marked with + (GrowNet, SVR-Linear) fall outside TabNSM's critical difference band and are significantly worse. The large CD is expected when $k \gg N$; the Friedman test nonetheless confirms a significant overall difference ($p = 2.04 \times 10^{-6}$).

The Nemenyi post-hoc test is known to be conservative when $k \gg N$: with 39 models and only 5 benchmarks, the critical difference is CD = 27.9 rank units, leaving most pairwise comparisons inconclusive regardless of the true effect. This is a known limitation of the Friedman–Nemenyi procedure in wide-but-short rank matrices. The Wilcoxon signed-rank test sidesteps this by testing each pair directly across all nine benchmarks, providing substantially more statistical power. The two tests are therefore not contradictory: Nemenyi confirms only the largest rank gaps with high confidence given $N = 5$, while Wilcoxon leverages all nine benchmarks to establish significance for all but one baseline.

Pairwise Wilcoxon Signed-Rank Test Following the rejection of the global null, we conduct 41 one-sided pairwise Wilcoxon signed-rank tests comparing TabNSM against each baseline (H_1: TabNSM rank $<$ baseline rank) across all nine benchmarks, imputing missing values as worst rank. p-values are corrected for multiple comparisons using the Benjamini–Hochberg procedure [76] at $\alpha = 0.05$. TabNSM achieves a significantly lower rank than 40 of the 41 baselines after correction (corrected $p \leq 0.035$). The exception is TabFM (corrected $p = 0.064$), which is not significantly different from TabNSM even though its four out-of-memory runs (CD, ChD, EVP, RES) are imputed as worst rank; its average rank ($\bar{r} = 17.56$) is inflated by those imputed entries. The closest competitors by average rank are TabICLv2 ($\bar{r} = 9.33$, $p = 0.035$), ResNet ($\bar{r} = 11.06$, $p = 0.003$), ExtraTrees ($\bar{r} = 12.72$, $p = 0.021$), MLP ($\bar{r} = 13.22$, $p = 0.003$), and CatBoost ($\bar{r} = 13.83$, $p = 0.011$), compared to TabNSM's average rank of $\bar{r} = 2.61$ across nine benchmarks. Figure 6 summarizes the average ranks and significance results for all baselines.

We note that worst-rank imputation is conservative for TabNSM (which has no missing runs) and liberal for baselines with many incomplete runs; results should therefore be interpreted as an aggregate trend rather than a precise pairwise comparison for models with many incomplete entries. The tests above were computed on the same 41-baseline set as Table 3 (Appendix D.1). These tests cover all nine datasets, including the two low-dimensional ones (SA, CP); Section 4 reports the same tests restricted to the seven datasets with more than 250 features, where TabNSM is significantly better than all 41 baselines. Both should be read together with the standardized-benchmark results in Appendix E.1.

D.3 Low-Dimensional Datasets: Full Results and Re-aggregation

Results on SA and CP. Table 1 in the main text reports the suite's seven wide benchmarks (> 250 features); Table 4 below reports the remaining two, SA (27 features) and CP (12 features), in the same per-family format. These two datasets are substantially narrower and, in CP's case, the least redundant in the suite (Appendix H), so we summarize them here rather than in the main table. On SA, TabNSM surpasses DNNR (0.149→0.137), the strongest baseline trained per dataset, but trails the in-context foundation models TabFM (0.113) and TabICLv2 (0.118). On CP, TabFM (2.404) and TabICLv2 (2.505) lead ahead of the tree-based ensembles, with TabNSM (2.842) behind both. SA and CP are the two datasets in the suite where TabNSM does not place in the top two; full per-model results (all 42 methods) are in Table 3.

Table 4: RMSE on the suite's two narrow, low-dimensional benchmarks, in the same per-family format as Table 1.

Model	SA	CP
Best Classical (6 models)	0.175	3.719
Best Tree-based (5 models)	0.171	2.617
Best MLP-style (5 models)	0.167	2.920
Best Transformer (7 models)	0.156	2.901
Best Retrieval (4 models)	0.150	2.833
Best Other (9 models)	0.149	2.740
Best Foundation (5 models)	**0.113**	**2.404**
TabNSM (Ours)	0.137	2.842

Complete-case re-aggregation. Because SA (27 features) and CP (12 features) are not wide, we also re-aggregated Table 3 excluding them. To avoid rewarding or penalizing missing entries, the comparison is restricted to the 27 methods with results on all nine datasets. Table 5 and Figure 5 report complete-case average ranks. The re-aggregation leaves TabNSM first among the complete-case methods; this is a re-analysis of the same single-run results, not new evidence.

Table 5: Complete-case average ranks from Table 3 (lower is better).

Evaluation scope	**Datasets**	**TabNSM avg. rank**	**Position**
All datasets	9	2.00	1/27
Excluding SA and CP	7	1.29	1/27

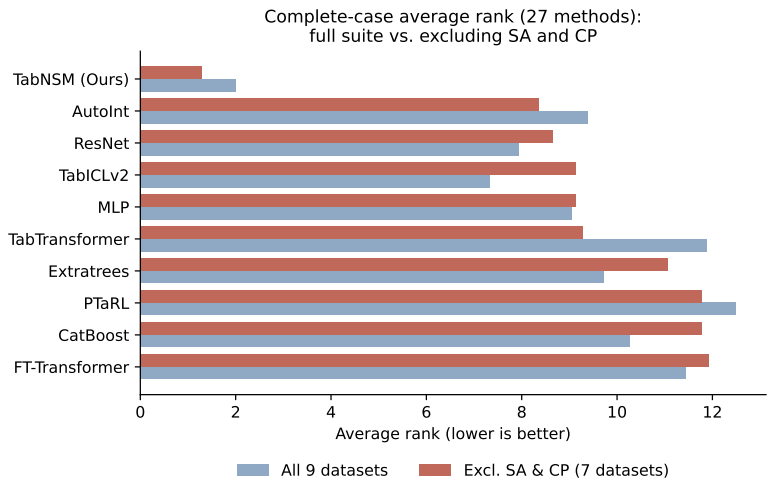


Figure 5: Average rank of the top-10 complete-case methods, all nine datasets versus excluding SA and CP.

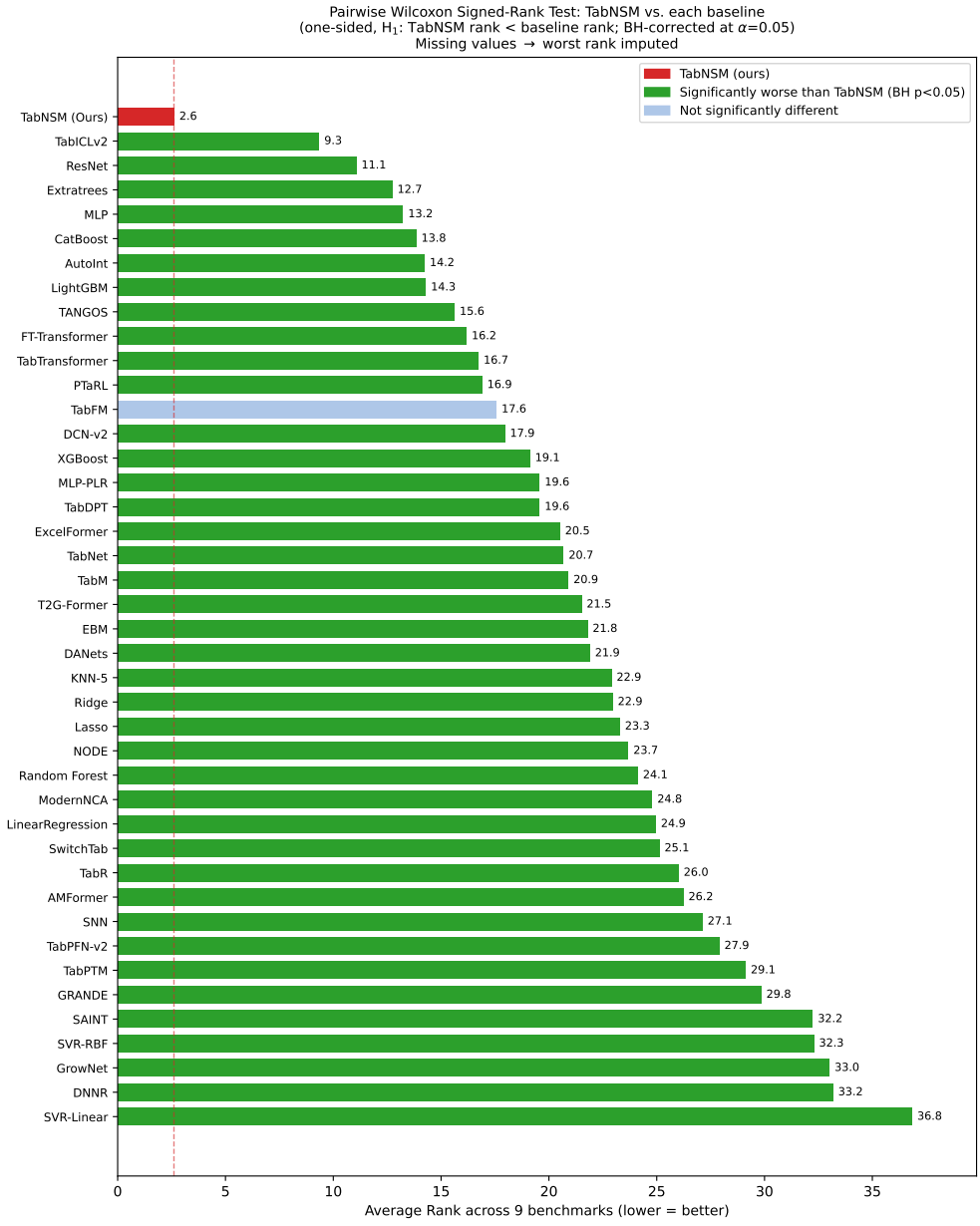


Figure 6: Average rank across nine benchmarks for each model (lower is better), colored by architectural family. All baselines except TabFM are statistically significantly worse than TabNSM (one-sided Wilcoxon signed-rank test, Benjamini–Hochberg corrected [76], $\alpha = 0.05$; corrected $p \leq 0.035$; TabFM: $p = 0.064$). Missing values are imputed as worst rank.

E Standardized-Benchmark Evaluation (TabArena-Lite)

E.1 TabArena-Lite Results

We also evaluated TabNSM on all 13 regression datasets of TabArena-Lite [19] under TabArena's own protocol (official split 0, eight-fold bagging, one hour per configuration and dataset) and compared it with TabArena's cached results for 77 methods (default, tuned, and tuned-plus-ensemble variants). TabNSM is evaluated both as a default configuration (released defaults, corrected GridLoss of Eq. 4, target standardized with training-set statistics) and under TabArena's tuning protocol, a random search of 100 configurations over the search space described below, from which TabArena derives the tuned variant (best configuration by inner validation) and the tuned-plus-ensemble variant (weighted

ensemble of configurations). The protocol and per-dataset RMSE are given below (Table 7). Table 6 reports average rank over the resulting 80 methods.

Table 6: TabArena-Lite regression results: average per-dataset rank over 13 datasets among 80 methods (lower is better). TabNSM (tuned) and (tuned + ens.) use a 100-configuration random search under TabArena's protocol. †TabArena's standard missing-result handling applies to six of the thirteen TabPFNv2 tasks; TabNSM has results on all thirteen.

Method (configuration)	Avg. rank	Method (configuration)	Avg. rank
TabDPT (tuned + ens.) [17]	11.7	TabPFNv2 (tuned + ens.)† [14]	34.2
TabICLv2 (default) [16]	15.8	TabM (default)	41.2
RealMLP (tuned + ens.) [77]	17.2	RealMLP (default)	43.5
TabDPT (default)	19.5	TabPFNv2 (default)†	45.2
LightGBM (tuned + ens.)	24.2	XGBoost (default)	49.8
CatBoost (tuned + ens.)	26.4	**TabNSM (tuned + ens.)**	**55.2**
TabM (tuned + ens.) [58]	27.2	**TabNSM (tuned)**	**58.3**
XGBoost (tuned + ens.)	32.3	Random Forest (default)	59.8
CatBoost (default)	34.2	**TabNSM (default)**	**64.4**

TabArena-Lite is an out-of-regime control for TabNSM: 12 of its 13 datasets have fewer than 100 features, most have under 20k rows, and their median redundancy score is 0.48 versus 0.07 for the nine-dataset suite ($p = 0.003$; see below). Under these conditions TabNSM is not competitive with tuned GBDTs or in-context foundation models: the default configuration ranks 64.4 of 80, and tuning under TabArena's own protocol raises this to 58.3 (tuned) and 55.2 (tuned + ensemble), level with default Random Forest (59.8) and still below default XGBoost (49.8) and CatBoost (34.2). The tuning gains concentrate on the five datasets with fewer than 2,000 rows (e.g., `airfoil` 66→48, `QSAR_fish` 74→44 in rank for the tuned + ensemble variant), whereas on the six datasets that hit TabArena's one-hour fit limit the tuned ranks move by at most 13 places, which points to the wall-clock cap rather than the search space as the binding constraint at those sizes. A second, self-fit evaluation of the same 13 datasets on our own 72/8/20 split, with TabNSM and six tree/linear baselines each given a default configuration plus a 20-configuration random search, gives the same picture (TabNSM 9.5–9.9 of 14 variants; wins on the smallest and third-smallest datasets by row count). Two diagnostics reported below bound what this means. First, TabNSM's validation RMSE is optimistic relative to test ($+3.8\%$ median versus within $\pm 1\%$ for the tree baselines), so validation-based selection under-tunes this architecture; this is consistent with the modest gain from 100 tuned configurations under TabArena's eight-fold inner validation, and the 20-configuration search on an 8%-of-data validation split does not recover the gap either. Second, redundancy does not rank datasets within TabArena either (Spearman $\rho = -0.08$); the cleaner pattern is sample size. On the 1,024-feature QSAR-TID-11, the one TabArena dataset in TabNSM's design regime, the default configuration ranks 34 of 80 (RMSE 0.838 versus 0.791 for the best method; tuning does not improve it), its best placement in the suite, and the only regime-matched data point on a standardized benchmark. TabNSM is thus validated in the wide, redundant regime of the main-text benchmarks; extending it to small, low-redundancy tables is a tuning and model-selection question that the diagnostics above make concrete rather than an open architectural one.

E.2 Protocol and Per-Dataset Results

TabArena-Lite protocol. The evaluation above runs TabNSM on the 13 regression datasets of TabArena-Lite [19] using the official split 0 with eight-fold inner bagging and compares against TabArena's cached results for 77 methods (default, tuned, and tuned-plus-ensemble configurations). Every fit is subject to TabArena's one-hour limit per configuration and dataset (shared across the eight bagged folds); six of the 13 datasets (`diamonds`, `Food_Delivery_Time`, `houses`, `miami_housing`, `physiochemical_protein`, `superconductivity`) hit this limit, so TabNSM's fits there are truncated by early stopping on wall-clock rather than on validation loss. TabNSM is wrapped as an AutoGluon model: categoricals are integer-encoded, missing values zero-imputed, features standardized with training-set statistics, and the target standardized with training-set statistics and de-standardized at prediction time. The loss is the corrected GridLoss of Eq. 4 (grid-step rescaling and relative temperature) blended with Huber loss as in the main experiments. An earlier run with the uncorrected loss and the same target standardization gave an average rank of 68.3; the correction improves the average rank to 64.4, with the largest gains on the two datasets whose targets or feature counts are largest (`QSAR-TID-11`: RMSE 1.112→0.838; `diamonds`: 816→624). All

TabNSM fits ran on a single NVIDIA RTX A4000 (16 GB) with five fits sharing the GPU concurrently, which we measured to slow each fit by roughly 32% relative to an exclusive GPU; under the fixed one-hour limit this reduces the number of epochs completed on the six truncated datasets.

Tuning protocol. TabArena's tuned variants are obtained by fitting the default configuration plus random configurations under the same per-fit limit, selecting the best configuration by the eight-fold inner validation score ("tuned"), or building a weighted ensemble over the fitted configurations ("tuned + ensemble"). We ran 100 random configurations (TabArena's reference methods use up to 200). The search space is the paper's Optuna space for the sparse-attention geometry (head dimension in $\{8, 16, \ldots, 64\}$, 1–8 heads, sliding window 1–8, compression block size in $\{4, 8, 12, 16\}$ with stride 2, selection block size an even number up to the compression block size, 1–4 selected blocks), learning rate log-uniform in $[10^{-5}, 3 \times 10^{-3}]$, batch size in $\{32, 64, 128, 256\}$, together with token width in $\{16, 32, 64\}$, dropout uniform in $[0, 0.5]$, weight decay log-uniform in $[10^{-6}, 10^{-2}]$, the GridLoss weight in the Huber/GridLoss blend in $\{0, 0.1, 0.3, 0.5\}$ (0 recovers plain Huber), grid resolution in $\{128, 256, 512, 1024\}$, relative temperature in $\{0.25, 0.5, 1\}$, and RISE sampling either off or on with 2–8 loss-quantile groups and boost in $\{1.5, 2, 4\}$; all configurations train for up to 100 epochs with early stopping (patience 10) on the inner validation loss. Table 7 reports per-dataset RMSE for the three TabNSM variants and representative baselines, with TabNSM's rank among the 80 methods in brackets.

Table 7: TabArena-Lite regression, per-dataset test RMSE (lower is better; best in bold). Datasets are ordered by size (n rows, d features). "T+E" denotes TabArena's tuned-plus-ensemble configuration and "def." the default configuration. Bracketed numbers give TabNSM's rank among all 80 methods on that dataset.

Dataset (n, d)	CatBoost (def.)	CatBoost (T+E)	LightGBM (T+E)	RealMLP (T+E)	TabM (T+E)	TabPFNv2 (def.)	TabICLv2 (def.)	TabDPT (T+E)	TabNSM (T+E)	TabNSM (tuned)	TabNSM (def.)
QSAR-fish (907, 6)	0.918	0.913	0.915	0.901	0.908	0.903	**0.894**	0.897	0.920 [44]	0.924 [52]	0.950 [74]
concrete (1,030, 8)	4.542	4.346	4.232	4.141	4.358	3.970	**3.673**	4.040	5.057 [53]	4.962 [50]	5.833 [72]
insurance (1,338, 6)	4,110	4,115	4,101	4,071	4,097	4,322	**4,042**	4,104	4,207 [54]	4,240 [60]	4,274 [65]
airfoil (1,503, 5)	1.475	1.181	1.340	1.012	0.978	0.969	1.015	**0.949**	1.526 [48]	1.645 [57]	1.868 [66]
Fiat-500 (1,538, 7)	699.31	699.19	694.78	690.27	718.35	689.27	686.06	**676.98**	715.03 [51]	714.57 [50]	711.16 [46]
QSAR-TID-11 (5,742, 1024)	0.848	0.843	0.820	0.831	0.825	0.862	0.874	**0.800**	0.840 [37]	0.842 [40]	0.838 [34]
wine (6,497, 12)	0.612	0.594	0.594	0.584	0.597	0.683	0.588	**0.579**	0.634 [61]	0.662 [71]	0.654 [68]
miami (13,776, 15)	81,043	79,487	81,592	81,793	81,751	85,523	79,394	**78,293**	88,232 [53]	89,677 [58]	89,340 [56]
houses (20,640, 8)	0.213	0.213	0.213	0.200	0.206	0.234	**0.196**	0.203	0.238 [64]	0.239 [65]	0.249 [70]
supercond. (21,263, 81)	9.264	9.259	9.139	9.182	9.114	9.583	**8.942**	8.956	10.09 [66]	10.10 [67]	11.05 [72]
food-deliv. (45,451, 9)	7.419	**7.405**	7.415	7.437	7.691	7.891	7.669	7.444	8.004 [68]	8.014 [70]	8.056 [71]
protein (45,730, 9)	3.557	3.418	3.416	3.141	3.260	3.592	3.027	**2.938**	3.631 [61]	3.634 [62]	4.199 [74]
diamonds (53,940, 9)	529.75	529.30	522.90	511.28	525.01	548.89	**503.00**	532.04	565.40 [58]	563.19 [56]	624.03 [69]

Self-fit 72/8/20 comparison. As an independent check that does not depend on TabArena's cached folds, we ran the same 13 datasets under the 72/8/20 split of Section 4 and fit every method ourselves: CatBoost, XGBoost, LightGBM, Random Forest, ExtraTrees, and Ridge, each with its default configuration and a 20-configuration random search selected on validation RMSE (the same budget the main experiments give to baselines). Preprocessing is identical across methods (integer-encoded categoricals, zero-imputed missing values); TabNSM standardizes features and target internally, tree models use raw values. Table 8 reports the aggregate result and Table 9 the per-dataset rank of TabNSM.

Table 8: Self-fit 13-dataset benchmark, 72/8/20 split, all methods fit under a matched default-plus-20-configuration budget. nRMSE is RMSE normalized per dataset; wins count first-place finishes. TabNSM fits ran on a V100-32GB GPU, tree/linear baselines on CPU.

Method	Mean rank (of 14)	Mean nRMSE	Wins	Median fit (s)
CatBoost (tuned)	3.38	0.408	3	13.2
XGBoost (tuned)	3.46	0.412	3	0.38
CatBoost (default)	4.62	0.418	2	20.6
LightGBM (tuned)	4.77	0.416	2	0.41
ExtraTrees (tuned)	5.15	0.424	1	1.50
Random Forest (tuned)	6.15	0.426	0	4.59
LightGBM (default)	6.46	0.423	0	0.20
ExtraTrees (default)	7.62	0.446	1	2.77
Random Forest (default)	7.85	0.434	0	8.33
XGBoost (default)	8.62	0.431	0	0.19
TabNSM (default)	**9.54**	**0.467**	**1**	**388.7**
TabNSM (tuned)	**9.92**	**0.466**	**1**	**249.3**
Ridge (tuned)	13.08	0.617	0	0.0014
Ridge (default)	13.31	0.631	0	0.017

TabNSM's two wins are on the smallest dataset (`QSAR_fish_toxicity`, 907 rows) and the third-smallest (`healthcare_insurance_expenses`, 1,338 rows), not the two smallest: on the second-smallest (`concrete`, 1,030 rows) it ranks 11th/12th. On every dataset above 13,776 rows it ranks 10th, 11th, or 12th. Tuning ranks slightly worse than the default (9.92 versus 9.54) despite a marginally better mean nRMSE, consistent with a 20-configuration search overfitting an 8%-of-data validation split; on QSAR-TID-11 no searched configuration beat the default on validation. Fit cost is three orders of magnitude above LightGBM's.

Table 9: Per-dataset rank of TabNSM (of 14 method variants) in the self-fit benchmark, sorted by row count, with the feature-redundancy score of Appendix H.

Dataset	n	Redundancy	TabNSM (default)	TabNSM (tuned)	Winner
QSAR_fish_toxicity	907	0.805	2	1	TabNSM (tuned)
concrete_compressive_strength	1,030	0.690	11	12	CatBoost (tuned)
healthcare_insurance_expenses	1,338	0.991	1	9	TabNSM (default)
airfoil_self_noise	1,503	0.944	12	11	CatBoost (tuned)
Another-Dataset-on-used-Fiat-500	1,538	0.668	11	8	LightGBM (tuned)
QSAR-TID-11	5,742	0.243	9	10	XGBoost (tuned)
wine_quality	6,497	0.541	12	11	XGBoost (tuned)
miami_housing	13,776	0.474	11	12	CatBoost (default)
houses	20,640	0.385	11	12	CatBoost (tuned)
superconductivity	21,263	0.066	11	12	LightGBM (tuned)
Food_Delivery_Time	45,451	0.481	11	10	CatBoost (default)
physiochemical_protein	45,730	0.210	11	12	ExtraTrees (default)
diamonds	53,940	0.341	11	12	XGBoost (tuned)

Validation-to-test gap. Table 10 reports, for default configurations, the relative degradation from validation RMSE to test RMSE across the 13 datasets. Every baseline's median is within one point of zero; TabNSM's is $+3.8\%$. A plausible mechanism is that restoring the best-validation checkpoint of a high-capacity model trained for many epochs overfits a small validation split (8% of data, i.e., 73–4,300 rows on these datasets) more than a gradient-boosted model's early stopping does; we have not isolated it further. Two consequences follow. First, the gap is a model-selection issue, not a measurement one: every test RMSE we report is computed once on a held-out split after selection is complete, so the reported numbers are unbiased estimates for the selected model, and if anything conservative for the architecture, since selection is picking sub-optimal checkpoints and configurations. Second, it explains why the 20-configuration search does not help on this suite and points to the fix, selection on a larger or inner held-out split, which we leave to future work.

Table 10: Validation-to-test RMSE degradation, default configurations, 13 datasets. Positive values mean validation RMSE understates test RMSE.

	TabNSM	LightGBM	Ridge	Random Forest	CatBoost	ExtraTrees	XGBoost
Mean	$+4.7\%$	$+1.5\%$	$+0.6\%$	-0.0%	-0.6%	-0.7%	-0.8%
Median	$+3.8\%$	-0.0%	-0.5%	$+0.2\%$	-0.1%	$+0.0\%$	$+1.0\%$

Does feature structure explain the gap? Applying the redundancy score of Appendix H to TabArena's 13 datasets gives a median of 0.481 versus 0.073 for the nine-dataset suite (Mann–Whitney $p = 0.003$), so the two suites do test structurally different feature regimes. However, the Spearman correlation between redundancy score and TabNSM's rank in Table 9 is -0.08 ($p = 0.79$), and TabNSM's two best ranks are on the two *least* redundant datasets. Redundancy therefore characterizes the difference between the two suites rather than ranking datasets within either of them; within TabArena the cleaner pattern is sample size. The defensible claim is that TabNSM has not yet been evaluated on a standardized benchmark in the specific feature regime it was designed and tuned for, with QSAR-TID-11 as the single regime-matched point.

F Brief Summary of Baseline Model Setup

All baseline models were implemented in Python using standard libraries. Data were loaded from ARFF files into pandas DataFrames, categorical strings decoded to UTF-8 and converted to numeric where possible, missing values filled with zeros, and the target variable removed before training. The data were split into 80% training+validation and 20% test, with the training+validation portion further split into 90% training and 10% validation. Neural and hybrid models (DeepGBM, TabNet, FT-Transformer) used z-score normalization, while tree-based models used raw values.

Performance was evaluated on the held-out test set using RMSE (primary), MSE, MAE, and R^2. Hyperparameter optimization was performed with Optuna (3-fold CV on training+validation, 20 trials, maximizing negative RMSE). The in-context foundation models (TabPFNv2, TabPTM, TabDPT, TabICLv2, TabFM) have no per-dataset training and were not tuned; each was run once with its released default configuration on the same splits. The best configuration was used to train a final model on the training set (using the validation set when supported). Final settings were reported in table 11 exclusively on the test set, and all configurations were logged for reproducibility.

Table 11: Hyperparameter search spaces for baseline models. Log-scale indicates logarithmic sampling.

Model	Hyperparameter Search Space
XGBoost	$n_{\text{estimators}} \in [200, 800]$; learning rate $\in [10^{-3}, 0.3]$ (log); max depth $\in [3, 12]$; subsample $\in [0.6, 1.0]$; colsample_bytree $\in [0.6, 1.0]$; $\gamma \in [0, 5]$; ℓ_1, ℓ_2 regularization $\in [0, 5]$
LightGBM	$n_{\text{estimators}} \in [200, 1000]$; learning rate $\in [10^{-3}, 0.3]$ (log); num leaves $\in [15, 255]$; max depth $\in [-1, 16]$; subsample $\in [0.6, 1.0]$; colsample_bytree $\in [0.6, 1.0]$; ℓ_1, ℓ_2 regularization $\in [0, 5]$
Random Forest	$n_{\text{estimators}} \in [200, 1000]$; max depth $\in [4, 32]$; max features $\in [0.4, 1.0]$; min samples split $\in [2, 10]$; min samples leaf $\in [1, 5]$
CatBoost	depth $\in [4, 10]$; learning rate $\in [10^{-3}, 0.3]$ (log); ℓ_2 leaf regularization $\in [10^{-3}, 10]$ (log); iterations $\in [500, 1500]$
DeepGBM$\star$	num boost rounds $\in [200, 800]$; learning rate $\in [10^{-3}, 0.1]$ (log); max depth $\in [3, 10]$
TabNet	$n_d, n_a \in [8, 64]$; n steps $\in [3, 7]$; $\gamma \in [1.0, 2.5]$; $\lambda_{\text{sparse}} \in [10^{-6}, 10^{-3}]$ (log); learning rate $\in [10^{-4}, 10^{-2}]$ (log); momentum $\in [0.01, 0.4]$
FT-Transformer	model dimension $\in \{64, 128, 256\}$; attention heads $\in \{4, 8\}$; transformer layers $\in [2, 5]$; FFN expansion $\in [2.0, 6.0]$; learning rate $\in [10^{-4}, 5 \times 10^{-3}]$ (log); epochs $\in [60, 200]$; batch size $\in \{128, 256, 512\}$
MLP	hidden layers $\in [1, 8]$ with widths $\in [64, 512]$; dropout $\in [0, 0.5]$; learning rate $\in [10^{-5}, 10^{-2}]$ (log); weight decay $\in [10^{-6}, 10^{-3}]$ (log)
ResNet	n layers $\in [1, 8]$; hidden dim $\in [64, 512]$; hidden factor $\in [1.0, 4.0]$; hidden dropout $\in [0, 0.5]$; residual dropout $\in [0, 0.5]$; learning rate $\in [10^{-5}, 10^{-2}]$ (log); weight decay $\in [10^{-6}, 10^{-3}]$ (log)
AutoInt	n layers $\in [1, 6]$; $d_{\text{token}} \in \{8, 16, 32, 64, 128\}$; attention dropout $\in [0, 0.5]$; residual dropout $\in [0, 0.2]$; learning rate $\in [10^{-5}, 10^{-3}]$ (log); weight decay $\in [10^{-6}, 10^{-3}]$ (log)
SNN	hidden layers $\in [2, 16]$ with widths $\in [1, 512]$; $d_{\text{embedding}} \in [64, 512]$; dropout $\in [0, 0.1]$; learning rate $\in [10^{-5}, 10^{-2}]$ (log); weight decay $\in [10^{-6}, 10^{-3}]$ (log)
DCN V2	$d \in [64, 512]$; $d_{\text{embedding}} \in [64, 512]$; n cross layers $\in [1, 8]$; n hidden layers $\in [1, 8]$; hidden / cross dropout $\in [0, 0.5]$; learning rate $\in [10^{-5}, 10^{-1}]$ (log); weight decay $\in [10^{-6}, 10^{-3}]$ (log)
TabR	$d_{\text{main}} \in [96, 384]$; encoder blocks $\in [0, 1]$; predictor blocks $\in [1, 2]$; context dropout $\in [0, 0.6]$; PLR embeddings: $n_{\text{freq}} \in [16, 96]$, freq scale $\in [10^{-2}, 10^{2}]$ (log), $d_{\text{emb}} \in [16, 64]$; learning rate $\in [3\times10^{-5}, 10^{-3}]$ (log)
ModernNCA	dim $\in [64, 1024]$; $d_{\text{block}} \in [64, 1024]$; n blocks $\in [0, 2]$; dropout $\in [0, 0.5]$; sample rate $\in [0.05, 0.6]$; PLR embeddings: $n_{\text{freq}} \in [16, 96]$, freq scale $\in [5\times10^{-3}, 10]$ (log); learning rate $\in [10^{-5}, 10^{-1}]$ (log)
NODE	num layers $\in [1, 4]$; depth $\in [4, 6]$; tree dim $\in [2, 3]$; layer dim $\in \{512, 1024\}$; learning rate $\in [10^{-5}, 10^{-1}]$ (log); weight decay $\in [10^{-6}, 10^{-3}]$ (log)
TabCaps*	sub class $\in [1, 5]$; init dim $\in [32, 128]$; primary capsule size $\in [4, 32]$; digit capsule size $\in [4, 32]$; leaves $\in [16, 64]$; learning rate $\in [10^{-5}, 10^{-1}]$ (log)
SAINT	dim $\in \{16, 32, 64\}$; depth $\in \{4, 6\}$; heads $\in \{4, 8\}$; attn / ff dropout $\in [0, 0.5]$; attention type $\in$ {colrow, row, col}; learning rate $\in [3\times10^{-5}, 10^{-3}]$ (log)
TANGOS	hidden layers $\in [1, 8]$ with widths $\in [64, 512]$; dropout $\in [0, 0.5]$; $\lambda_1 \in [10^{-3}, 10]$ (log); $\lambda_2 \in [10^{-4}, 1]$ (log); subsample $\in [30, 100]$; learning rate $\in [10^{-4}, 10^{-3}]$ (log)
PTaRL	hidden layers $\in [1, 3]$ with widths $\in [64, 512]$; dropout $\in [0, 0.5]$; $d_{\text{embedding}} \in [64, 128]$; learning rate $\in [10^{-5}, 10^{-1}]$ (log)

Continued on next page

Model	Hyperparameter Search Space
DANets	n layers $\in [6, 32]$; base outdim $\in [64, 128]$; dropout $\in [0, 0.5]$; learning rate $\in [10^{-5}, 10^{-1}]$ (log); weight decay $\in [10^{-6}, 10^{-3}]$ (log)
TabTransformer	dim $\in \{32, 64, 128, 256\}$; depth $\in \{1, 2, 3, 6, 12\}$; heads $\in \{2, 4, 8\}$; attn / ff dropout $\in [0, 0.5]$; learning rate $\in [10^{-5}, 10^{-1}]$ (log); weight decay $\in [10^{-6}, 10^{-3}]$ (log)
GrowNet	hidden dim $\in [32, 512]$; $d_{\text{embedding}} \in [32, 512]$; epochs per stage $\in [1, 2]$; correction epochs $\in [1, 2]$; learning rate $\in [10^{-5}, 10^{-1}]$ (log); weight decay $\in [10^{-6}, 10^{-3}]$ (log)
DNNR	n neighbors = 3; n derivative neighbors $\in [32, 1024]$; order $\in \{1, 2\text{diag}\}$; solver $\in$ {linear regression, scipy lsqr, numpy, ridge, lasso}; index = annoy
SwitchTab	$\alpha \in [10^{-2}, 10^{2}]$ (log); learning rate $\in [10^{-6}, 10^{-3}]$ (log); weight decay $\in [10^{-6}, 10^{-3}]$ (log)
BiSHop ⋆	emb dim $\in \{16, 24, 32, 48, 64, 128, 256, 320\}$; out dim $\in \{2, 4, \ldots, 128\}$; patch dim $\in \{1, 2, 4, 6, 8, 12, 16, 24\}$; $n_{\text{agg}} \in \{2, \ldots, 8\}$; factor $\in \{5, 10, 15\}$; $d_{\text{model}} \in \{64, \ldots, 1024\}$; $d_{\text{ff}} \in \{128, \ldots, 1024\}$; heads $\in \{2, 4, 6, 8, 10, 12\}$; encoder layers $\in \{2, 3, 4, 5\}$; decoder layers $\in \{0, 1\}$; learning rate $\in [10^{-6}, 10^{-4}]$ (log)
ProtoGate* [78]	hidden layers $\in [1, 4]$ with widths $\in [64, 512]$; learning rate $\in \{0.05, 0.075, 0.1\}$; ℓ_1 coefficient $\in \{1, 2, 3, 4, 6\} \times 10^{-4}$; k for prediction $\in \{1, \ldots, 5\}$
RealMLP [77] ⋆	embedding type $\in$ {none, pbld, pl, plr}; front scale $\in$ {true, false}; hidden sizes $\in \{[256, 256, 256], [64\times5], [512]\}$; activation $\in$ {selu, relu, mish}; dropout $\in \{0, 0.15, 0.30\}$; weight decay $\in \{0, 2\times10^{-2}\}$; learning rate $\in [0.02, 0.3]$ (log); PLR $\sigma \in [0.05, 0.5]$ (log); label smoothing $\in \{0, 0.1\}$
MLP-PLR	hidden layers $\in [1, 8]$ with widths $\in [64, 1024]$; dropout $\in [0, 0.5]$; PLR embeddings: $n_{\text{freq}} \in [16, 96]$, freq scale $\in [10^{-2}, 10^{2}]$ (log), $d_{\text{emb}} \in [16, 64]$; learning rate $\in [10^{-5}, 10^{-2}]$ (log); weight decay $\in [10^{-6}, 10^{-3}]$ (log)
ExcelFormer	n layers $\in [2, 5]$; $d_{\text{token}} \in \{8, 16, 32, 64, 128\}$; attention / ffn / residual dropout $\in [0, 0.5]$; mix type $\in$ {none, feat_mix, hidden_mix}; learning rate $\in [10^{-5}, 10^{-3}]$ (log); weight decay $\in [10^{-6}, 10^{-3}]$ (log)
GRANDE	depth $\in [3, 7]$; $n_{\text{estimators}} \in [512, 2048]$; selected variables $\in \{0.5, 0.75, 1.0\}$; data subset fraction $\in \{0.8, 1.0\}$; dropout $\in \{0, 0.25\}$; learning rate $\in [10^{-4}, 0.25]$ (log); weight decay $\in [10^{-6}, 10^{-3}]$ (log)
AMFormer	depth $\in [1, 4]$; dim $\in \{8, 16, 32, 64, 128\}$; attn / ff dropout $\in [0, 0.5]$; num special tokens $\in [1, 4]$; learning rate $\in [10^{-5}, 10^{-3}]$ (log); weight decay $\in [10^{-6}, 10^{-3}]$ (log)
Trompt[79] ⋆	n cycles $\in [3, 8]$; $d \in \{32, 64, 128, 256\}$; $P \in \{32, 64, 128, 256\}$; learning rate $\in [10^{-5}, 10^{-3}]$ (log); weight decay $\in [10^{-6}, 10^{-3}]$ (log)
TabM	backbone n blocks $\in [1, 5]$; $d_{\text{block}} \in [64, 1024]$; dropout $\in [0, 0.5]$; PLR embeddings: $n_{\text{freq}} \in [16, 96]$, freq scale $\in [5\times10^{-3}, 10]$ (log), $d_{\text{emb}} \in [16, 64]$; learning rate $\in [10^{-4}, 5\times10^{-3}]$ (log); weight decay $\in [10^{-4}, 10^{-1}]$ (log)
T2G-Former	n layers $\in [1, 4]$; $d_{\text{token}} \in \{8, 16, 32, 64, 128\}$; attention / ffn / residual dropout $\in [0, 0.5]$; d_{ffn} factor $\in [0.667, 2.667]$; frozen switch $\in$ {true, false}; learning rate $\in [10^{-5}, 10^{-3}]$ (log); weight decay $\in [10^{-6}, 10^{-3}]$ (log)

Continued on next page

Model	Hyperparameter Search Space
TabAutoPNPNet [51] ⋆	Fourier features $\in \{10, 25, 50, 75, 100\}$; max poly terms $\in \{2, 3, 5, 7, 10\}$; poly type $\in$ {chebyshev, legendre, hermite, laguerre}; hidden size $\in \{64, 128, 256, 512\}$; residual $\in$ {true, false}; learning rate $\in \{5\times10^{-4}, 5\times10^{-3}, 5\times10^{-2}\}$
TabPFNv2	non-parametric inference (foundation model); no tunable hyperparameters; sample size capped at $10\,000$ and feature count at 500 by the public implementation; datasets above either limit (AQ, CD, ChD, EVP, RES) were not evaluated and are marked $*$ in Table 3, with no subsampling or limit override applied
HyperFast* [80]	non-parametric inference (foundation model); no tunable hyperparameters
TabPTM	non-parametric inference (foundation model); no tunable hyperparameters
TabDPT [17]	in-context inference (foundation model), released v1.3 weights; not tuned: default configuration with a 2,048-row subsampled context, 8-member ensemble, PCA feature reduction above the model's feature limit, `torch.compile` disabled
TabICLv2 [16]	in-context inference (foundation model), released regression checkpoint (`tabicl-regressor-v2-20260212`); not tuned: default configuration with 8 estimators; full training set used as context on all nine datasets
TabFM [18]	in-context inference (foundation model), v1.0.0 PyTorch weights; not tuned: empty configuration, as benchmarked in TabArena; exceeds the 80 GB GPU with the full training set as context on CD, ChD, EVP and RES (marked ‡ in Table 3)

The code for models marked with * was not provided for the regression task. Models marked with ⋆ were not tuned for most large-dataset benchmarks, often resulting in non-convergence or outlier performance. Therefore, we do not report their results in Table 3.

G Instance-adaptive feature selectivity

G.1 Per-instance attribution maps

To examine whether TabNSM learns instance-adaptive feature selectivity, we visualize per-instance feature attributions on the test set. Figure 7 shows a feature attribution heatmap, where each row corresponds to a test instance and each column corresponds to an input feature. The attribution score is computed using gradient–embedding attribution and normalized to $[0, 1]$ within each instance. Brighter values indicate features with stronger influence on the model's regression output for that sample.

The heatmap reveals two patterns. First, some features appear as consistently bright columns across many instances, suggesting globally influential predictors. Second, many instances exhibit distinct sparse attribution patterns, indicating that TabNSM dynamically emphasizes different feature subsets depending on the input context. This behavior is consistent with the design of the Adaptive Sparse Interaction Module (ASIM), which encourages instance-adaptive sparse feature interaction rather than relying on a fixed global feature weighting.

G.2 Are the attributions faithful? Remove-and-retrain

The attribution maps above show *where* the model is sensitive, but a heatmap alone cannot establish that the highlighted features are the ones the model actually depends on: an attribution can be visually sparse and still be uninformative. We therefore evaluate the attributions with the standard remove-and-retrain benchmark (ROAR) [52].

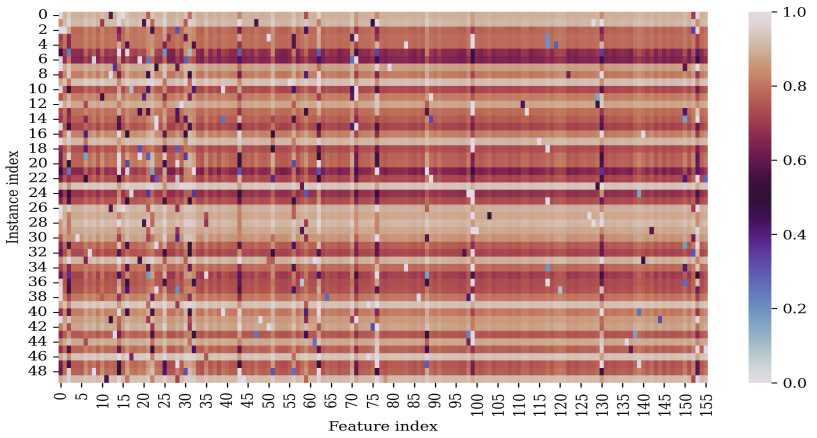


Figure 7: **Per-instance feature attribution heatmap for TabNSM.** Each row corresponds to a test instance and each column corresponds to an input feature. Cell intensity represents the normalized attribution score, computed using gradient–embedding attribution and scaled to $[0, 1]$ within each instance. Brighter cells indicate features with stronger influence on the prediction. The sparse and instance-varying attribution patterns suggest that TabNSM adaptively focuses on different feature subsets across samples.

Protocol. Features are ranked by their mean $|\text{gradient} \times \text{input}|$ computed on the *training* split only, so the test split never informs the ranking. We then delete the top-$k\%$ of features outright, rebuild the model at the reduced feature count, retrain from scratch with the released default configuration and the paper's 72/8/20 split, and evaluate the best-validation checkpoint. This is full ROAR, not inference-time masking: nothing is imputed and no model sees a masked input. Two controls use the identical pipeline: removing the *bottom*-$k\%$ of the same ranking, and removing a random $k\%$. A faithful ranking should degrade fastest under top-k removal and slowest under bottom-k removal. On AQ each non-random condition is a single training seed and the random control is averaged over three draws; on the other datasets every condition, including a $k = 0$ full-feature reference, is trained with three seeds and reported as mean $\pm$ standard deviation.

Which datasets the test can speak to. Remove-and-retrain is only informative when the model has learned feature-dependent structure: if a model is no better than predicting the training mean, no feature removal can hurt it and every ranking looks alike. Table 12 therefore reports, for each dataset, the mean-predictor floor, the trained model's RMSE and R^2, and the RMSE of a model trained on shuffled targets. AQ and EVP are strongly learned ($R^2 = 0.975$ and 0.997). TO and SC are not: under the fixed default configuration the trained model is within noise of the mean predictor ($R^2 = -0.04$ and $+0.01$) and of the shuffled-label model, consistent with the default-configuration means reported in Appendix I. On those two datasets the test is uninformative rather than negative, and we report it as such. EVP is run on a fixed-seed 20,000-row subsample (of 264,628) so that the 51-run retraining grid is tractable; the subsample is drawn once before the split and shared by every condition.

What the ranking does and does not claim. On AQ and EVP, 98% of the columns are dimensions of the 384-dimensional sentence-embedding block (Table 16), so no single column carries a standalone meaning and the result should not be read as an account of which named quantities drive the prediction. What it establishes is *where* predictive signal sits. That is the weaker claim in interpretive terms and the stronger one for this paper's argument: a dense sentence embedding might be expected to distribute signal across all of its dimensions, and instead most of the block can be removed with little effect provided the ranking chooses what to keep. This concentration is the premise ASIM's per-sample selection is built on, and it implies the embedding block can be pruned substantially at low cost. The two wide datasets whose features are not embedding-derived, TO and SC, cannot corroborate it: as noted above, no model there improves on the mean predictor, so their removal curves are flat whatever the ranking.

Results. Table 13 and Figure 8 report test RMSE as a function of the fraction removed. On AQ, deleting the 30% of features the attribution ranks highest nearly triples the error, from 4.016 to 11.063, and the damage persists at larger fractions, while deleting the 30% ranked lowest leaves the model essentially unchanged (4.187) and random deletion sits between the two. On EVP the picture is

Table 12: Informativeness of the remove-and-retrain test. A mean predictor outputs the training-set mean; R^2 is computed on the test split. The test can only discriminate rankings where R^2 is well above zero.

Dataset	Train rows	D	Mean-predictor RMSE	TabNSM RMSE	R^2	Shuffled-label RMSE
AQ	13,580	390	25.33	4.016	+0.975	25.66
EVP (20k-row subsample)	14,400	392	81.38	4.119	+0.997	90.43
TO	6,397	266	0.02844	0.02901	-0.043	0.02971
SC	3,135	385	3.972	3.942	+0.014	3.979

the same: deleting the 10% of features TabNSM ranks highest raises RMSE from 3.87 ± 0.25 to 28.95 ± 3.13 and the 70% to 44.76 ± 1.41, whereas bottom-k removal stays at 3.8–4.3 throughout and random removal reaches only 6.65 ± 3.27 at 50%. Summarizing each curve by the mean RMSE increase over the removal range gives, on AQ, 4.99 (top-k) $\gg$ 0.98 (random) $>$ 0.27 (TabNet) $>$ 0.06 (bottom-k), and on EVP 29.5 (top-k) $>$ 27.7 (TabNet) $\gg$ 4.1 (random) $>$ 0.19 (bottom-k); both are the signature expected of a faithful ranking. On TO and SC every curve is flat at the full-feature reference, as the informativeness analysis predicts.

Two caveats belong with the AQ numbers, which are single runs. At $k = 10\%$ the pruned model is nominally *better* than the full one (3.841 vs. 4.016); AQ's seed-to-seed spread is 11.7% relative (Appendix I), so this difference is within noise and should not be read as an improvement from pruning. Random removal also has a wide spread at 50–70% (± 2.8 and ± 3.0 across three draws). The EVP runs, with three seeds per condition, do not share this limitation.

Table 13: Feature-level ROAR. Each cell is the test RMSE after removing that fraction of features *and retraining from scratch*. AUC is the mean RMSE increase over the removal range (higher = the ranking identified more load-bearing features); the best AUC per dataset is in bold. Entries with $\pm$ are mean $\pm$ standard deviation over three training seeds; AQ non-random rows are single runs.

Dataset — Removal ranking	10%	30%	50%	70%	AUC
AQ, full feature set: RMSE 4.016					
TabNSM attribution (top-k)	3.841	11.063	10.924	11.286	**4.988**
TabNet masks (top-k)	4.307	3.951	4.574	4.451	0.266
Random	4.213 ± 0.245	4.162 ± 0.330	5.988 ± 2.829	6.306 ± 3.003	0.975
TabNSM attribution (bottom-k)	4.098	4.187	3.715	4.552	0.057
EVP (20k-row subsample), full feature set: RMSE 3.871 ± 0.250					
TabNSM attribution (top-k)	28.954 ± 3.127	33.294 ± 1.697	38.489 ± 1.580	44.759 ± 1.412	**29.514**
TabNet masks (top-k)	31.107 ± 1.105	30.247 ± 0.214	34.988 ± 0.539	42.154 ± 1.895	27.732
Random	5.463 ± 2.065	6.101 ± 2.551	6.652 ± 3.271	20.339 ± 18.168	4.126
TabNSM attribution (bottom-k)	4.107 ± 0.163	3.810 ± 0.402	4.212 ± 0.217	4.268 ± 0.417	0.187
TO, full feature set: RMSE 0.029 ± 0.001					
TabNSM attribution (top-k)	0.029 ± 0.000	0.029 ± 0.000	0.029 ± 0.000	0.029 ± 0.000	0.000
TabNet masks (top-k)	0.029 ± 0.000	0.029 ± 0.001	0.028 ± 0.000	0.029 ± 0.000	0.000
Random	0.028 ± 0.000	0.029 ± 0.000	0.028 ± 0.000	0.029 ± 0.000	0.000
TabNSM attribution (bottom-k)	0.029 ± 0.001	0.028 ± 0.000	0.029 ± 0.000	0.029 ± 0.000	0.000
SC, full feature set: RMSE 3.949 ± 0.008					
TabNSM attribution (top-k)	3.963 ± 0.007	3.949 ± 0.005	3.947 ± 0.011	3.956 ± 0.011	0.003
TabNet masks (top-k)	3.957 ± 0.010	3.958 ± 0.021	3.956 ± 0.010	3.954 ± 0.007	0.007
Random	3.965 ± 0.009	3.968 ± 0.014	3.944 ± 0.009	3.948 ± 0.017	0.007
TabNSM attribution (bottom-k)	3.957 ± 0.004	3.956 ± 0.002	3.944 ± 0.011	3.962 ± 0.012	0.004

Comparison at feature granularity. TabNet [65] produces per-feature masks natively, which makes it the natural attribution baseline at the same granularity. We trained it on the identical split and used its `feature_importances_` as a competing ranking through the same removal pipeline, so that only the ranking changes and the retrained model is held fixed. On AQ its ranking is markedly *less* damaging than random (AUC 0.27 vs. 0.98), i.e. its masks do not concentrate on the features the retrained model actually relies on; on EVP by contrast, TabNet's masks are nearly as informative as TabNSM's own attribution (AUC 27.7 vs. 29.5), so the AQ result should not be read as a general deficiency of TabNet's masks. The two rankings are essentially uncorrelated on every dataset (Spearman ρ between -0.05 and $+0.04$; Table 14). The reason is visible in how the mass is distributed: TabNet assigns nonzero importance to only 54 of 390 features on AQ (31 of 392 on EVP) and places essentially all of its mass in the top 10% of features (Gini ≥ 0.96 on every dataset), whereas the TabNSM attribution is diffuse (all features nonzero; Gini 0.25 on AQ, 0.45 on EVP). One consequence should be stated plainly in TabNet's favor: because most of its features are tied at zero importance, its ordering beyond the nonzero set is arbitrary, which is a structural disadvantage at the larger removal fractions. This comparison evaluates the masks as an attribution method under a fixed downstream model and is not a statement about TabNet's own accuracy.

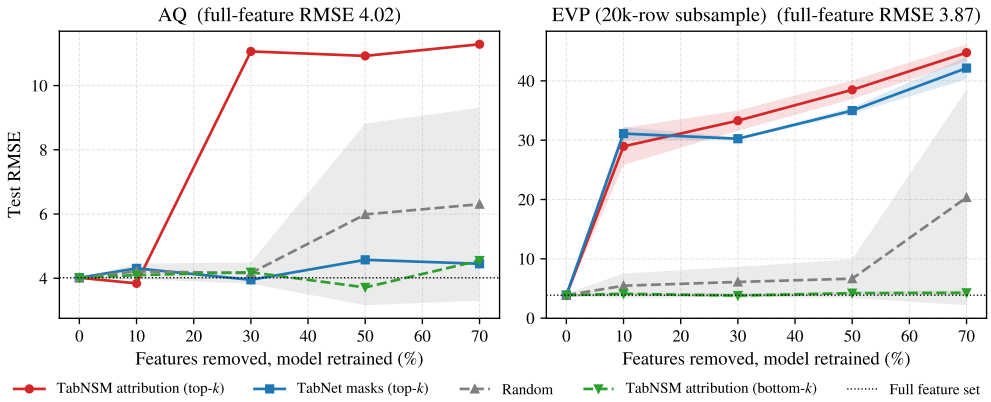


Figure 8: **Feature-level remove-and-retrain on AQ and EVP.** Test RMSE after deleting a fraction of the input features and retraining from scratch. Removing the features the model's own attribution ranks highest (red) degrades accuracy far faster than removing random features (gray; bands are one standard deviation over seeds), while removing its lowest-ranked features (green) tracks the full-feature reference (dotted). TabNet's native masks (blue) are shown through the same pipeline. TO and SC are omitted because every curve is flat at the reference (Table 13).

G.3 Sanity checks

Faithfulness alone does not rule out an attribution that is driven by the input or the architecture rather than by what was learned. We therefore run the two standard sanity checks [53] on every dataset (Table 14).

Model randomization. Re-initializing the trained weights module by module from the output backwards should progressively destroy agreement with the original attribution. On AQ the Spearman ρ falls from 0.972 (fusion head randomized) to 0.267 once the first-stage decoder is reached and -0.011 once ASIM itself is randomized; on EVP it falls from 0.850 to 0.030. On TO and SC, by contrast, ρ plateaus near 0.5 and 0.3 even with the whole network re-initialized: the attributions of a model that has learned nothing are a function of the input and the architecture, which is exactly the failure mode this check is designed to expose, and it exposes it.

Label randomization. Retraining on shuffled training targets should give attributions unrelated to the trained model's. On AQ the shuffled-label model cannot fit the task (test RMSE 25.66 vs. 4.02) and its attributions have $\rho = -0.01$ with the trained model's; on EVP the shuffled-label model has test RMSE 90.4 vs. 4.1 and $\rho = 0.05$. On TO and SC the shuffled-label model reaches the same test RMSE as the trained one and the two attributions correlate at $\rho = 0.75$ and 0.47, again reflecting that the trained model there is not distinguishable from an untrained one.

Table 14: Sanity checks on every dataset. Model randomization: Spearman ρ between the attribution of the trained model and that of a copy whose weights are re-initialized from the output backwards through the named module. Label randomization: test RMSE of a model trained on shuffled targets, and the ρ of its attribution with the trained model's. The last column is the rank agreement between TabNet's masks and TabNSM's attribution.

	Model randomization: ρ after randomizing through			Label randomization		
Dataset	head	first decoder	ASIM	test RMSE (shuffled / trained)	attribution ρ	TabNet vs. TabNSM ρ
AQ	+0.972	+0.267	-0.011	25.663 / 4.016	-0.014	-0.054
EVP (20k-row subsample)	+0.850	+0.358	+0.030	90.428 / 4.119	+0.049	+0.038
TO	+0.581	+0.527	+0.475	0.030 / 0.029	+0.752	+0.025
SC	+0.776	+0.369	+0.283	3.979 / 3.942	+0.474	-0.031

Scope. These tests concern the feature-level attributions of the trained model as a whole. On the datasets the model learns, they establish that those attributions are faithful and weight-dependent; they are agnostic to which internal component produces them, and do not by themselves isolate a mechanism inside any single module.

H Dataset Descriptions

This section provides detailed descriptions of the datasets used in our experiments, along with the associated prediction tasks.

Table 15: Dataset Statistics

Dataset	Abbreviation	# Samples	# Features
Sarcos	SA	48,933	27
CPU	CP	8,192	12
Topo	TO	8,885	266
SCAN	SC	4,355	386
Crime Data	CD	1,004,991	401
U.S. Chronic Disease	ChD	309,215	405
Electric Vehicle Population Data	EVP	264,628	392
Air Quality	AQ	18,862	390
Real Estate Sales	RES	1,048,575	389

Scale and feature structure of the suite. The suite spans 12–405 features and 4,355–1,048,575 examples, but it is not uniformly large-scale or high-dimensional: CP (8,192 examples), TO (8,885), and SC (4,355) are small by sample count, SA and AQ are medium-sized, and only CD, ChD, EVP, and RES are large; likewise SA and CP are not wide. An audit of the feature columns shows that "high feature count" is also not uniform in origin. Five of the nine datasets (AQ, ChD, CD, EVP, RES) derive 95–99% of their columns from the 384-dimensional sentence-embedding block produced by the preprocessing of Section 3; only 6–22 columns per dataset are original structured fields. Table 16 reports, for each dataset, the embedding-derived fraction and a redundancy score defined as the participation ratio of the feature correlation matrix normalized by the feature count (1.0 for statistically independent features, near 0 for highly redundant ones; computed on a 20,000-row sample for datasets above that size). Every dataset above 250 features, embedding-derived or not, has a score below 0.14; the two non-embedding wide datasets (TO, SC) are the most redundant, since spatially or anatomically adjacent raw features correlate strongly. The wide datasets in this suite are therefore highly redundant rather than intrinsically high-dimensional, and we state scalability claims for that regime only.

Table 16: Feature structure of the nine-dataset suite. Redundancy score is the normalized participation ratio of the feature correlation matrix (lower = more redundant).

Dataset	Features	Embedding-derived	Redundancy score
CP	12	0%	0.499
TO	266	0%	0.016
SA	27	0%	0.316
SC	385	0%	0.031
AQ	390	98%	0.042
ChD	405	95%	0.089
CD	401	96%	0.073
EVP	392	98%	0.063
RES	389	99%	0.134

Does redundancy predict TabNSM's per-dataset margin? The feature structure audit above lets us test, within this suite, the same relationship that Appendix E tests against TabArena: does a dataset's redundancy score predict how well TabNSM does on it? For each dataset we take TabNSM's rank among all 42 methods of Table 3 (ties by average rank, methods that failed to run on a dataset excluded from its ranking) and correlate it with the redundancy score of Table 16. Figure 9 shows the result: TabNSM ranks first on 5 of 9 datasets (TO, SC, EVP, ChD, RES; redundancy 0.016–0.134), second on AQ (0.042, behind TabFM), tied third on CD (0.073, with TabICLv2 indistinguishable at the reported precision), third on SA (0.316) and 10.5th on CP (0.499, the least redundant dataset in the suite). The Spearman correlation between redundancy score and rank is $\rho = 0.58$ ($p = 0.10$, $n = 9$)—directionally consistent with less-redundant data being harder for TabNSM, but not statistically significant, and driven largely by the two narrow datasets rather than a graded trend. This mirrors the null finding on TabArena (Appendix E, $\rho = -0.08$, $p = 0.79$): across both suites, redundancy alone does not reliably predict TabNSM's relative performance, so we do not use it as a leading explanation beyond the qualitative pattern already noted (Section 4).

topo_2_1. The Topo dataset was used, with 1,143 molecular descriptors. Molecular structures and response values were taken directly from the original studies. The final attribute in each file was used as the prediction target; for the Topo dataset, this target corresponds to oz267.

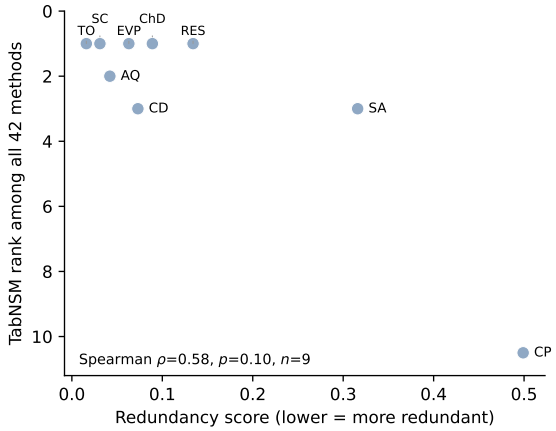


Figure 9: TabNSM's rank among all 42 methods (Table 3), by dataset, against the redundancy score of Table 16. Lower rank is better; lower redundancy score means more redundant features. Five of nine datasets are at rank 1, all in the more redundant part of the range; CP (least redundant) is the clearest exception.

Sarcos Robot Arm Dataset. The Sarcos dataset consists of measurements from a 7-degree-of-freedom robotic arm performing various motions. Each instance includes joint positions, velocities, and accelerations, and the prediction task is to estimate the torque of a specific joint (V22). This dataset is commonly used to evaluate regression models for learning inverse dynamics.

Electric Vehicle Population Dataset. This dataset contains records of battery electric and plug-in hybrid vehicle registrations in Washington State. Features include vehicle attributes and registration information, and the task is to predict the electric driving range.

Air Quality. The dataset consists of New York City air quality surveillance indicators measured across neighborhoods and time. Features describe pollutant exposure and related environmental health metrics, while the prediction target, Data Value, represents the quantitative measurement of each air quality indicator used for supervised learning.

U.S. Chronic Disease Dataset. The Chronic Disease dataset provides standardized public health indicators collected across U.S. states and territories. The prediction task involves estimating numeric values associated with chronic disease metrics.

Crime Dataset. The Crime dataset consists of crime incident reports from Los Angeles since 2020. Each record includes temporal, categorical, and spatial features, and the task is to predict the geographic area associated with each incident.

Real Estate Sales Dataset. This dataset contains property transaction records reported by municipalities in Connecticut. Given property and transaction features, the task is to predict the sales ratio, a commonly used indicator in real estate valuation and property tax assessment.

Standardized Centralized Alzheimer's Neuroimaging (SCAN) Dataset. The SCAN dataset integrates multimodal data including neuroimaging, fluid biomarkers, and demographic variables collected across Alzheimer's Disease Research Centers. The prediction task is to estimate Mini-Mental State Examination (MMSE) scores.

H.1 Dataset Links

All datasets used in our benchmark are publicly available. We provide direct links to each dataset in Table 17.

Table 17: Benchmark dataset sources.

Dataset	URL
CPU	https://www.openml.org/search?type=data&status=active&id=42841
Topo	https://www.openml.org/search?type=data&status=active&id=422
Sarcos Robot Arm Data	https://www.openml.org/search?type=data&status=active&id=43873&sort=runs
Electric Vehicle Population Data	https://catalog.data.gov/dataset/electric-vehicle-population-data
U.S. Chronic Disease Data	https://catalog.data.gov/dataset/u-s-chronic-disease-indicators
Air Quality	https://catalog.data.gov/dataset/air-quality
Crime Data from 2020 to Present	https://catalog.data.gov/dataset/crime-data-from-2020-to-present
Real Estate Sales Data	https://catalog.data.gov/dataset/real-estate-sales-2001-2018
Standardized Centralized Alzheimer's Neuroimaging (SCAN) Data	https://scan.naccdata.org/

Note: All datasets were accessed by August 2025.

I SEED VARIATION

To quantify variability with respect to random initialization, data partitioning, and training stochasticity, we retrained the full TabNSM model (with the corrected GridLoss of Eq. (4)) under five seeds (42–46) on the five datasets loadable by the released pipeline. Each seed also draws a new 72/8/20 train/validation/test split, so the reported spread includes split variability. Table 18 reports mean and sample standard deviation of test RMSE. Because repeating every dataset in Table 3 across many seeds is computationally expensive, we restrict this seed study to the five datasets loadable by the released pipeline and label the main-table entries as single runs.

Table 18: Test RMSE over 5 seeds (42–46), full model with the corrected GridLoss. Each seed draws a new train/validation/test split. The single-run entries of Table 3 are shown for reference.

Dataset	**Mean RMSE**	**Std. dev.**	**Rel. std.**	**Table 3 entry**
CP	2.989	0.070	2.3%	2.842
TO	0.0342	0.0033	9.6%	0.021
SA	0.1660	0.0043	2.6%	0.137
SC	5.150	0.677	13.2%	3.322
AQ	4.049	0.472	11.7%	3.999

Relative variation is low for CP and SA (2–3%), moderate for TO (9.6%), and substantial for SC and AQ (13.2% and 11.7%). The 5-seed means are also above the single-run entries of Table 3 on TO, SA, and SC, which were obtained with the tuned per-dataset configuration and the original split rather than the fixed default configuration and resampled splits used here; the two protocols are therefore not directly comparable, but the spread indicates that point estimates on SC and AQ in particular should not be read as precise without this uncertainty attached.

Table 19: Matched-device comparison on one RTX A4000 GPU (single seed). The ratio is TabNSM time divided by CatBoost time; values above one mean CatBoost is faster.

	Test RMSE		Fit time (s)		
Dataset	**CatBoost**	**TabNSM**	**CatBoost**	**TabNSM**	**Time ratio**
CP	**2.924**	3.106	6.98	246.79	35.3×
TO	**0.0276**	0.0308	1.67	368.86	220.5×
SA	0.1803	**0.1715**	68.50	1763.10	25.7×
SC	**3.530**	3.990	3.02	210.52	69.8×
AQ	7.059	**4.887**	57.37	676.01	11.8×
QSAR-TID-11 (1,024 feat.)	0.8103	**0.8071**	10.06	156.07	15.5×

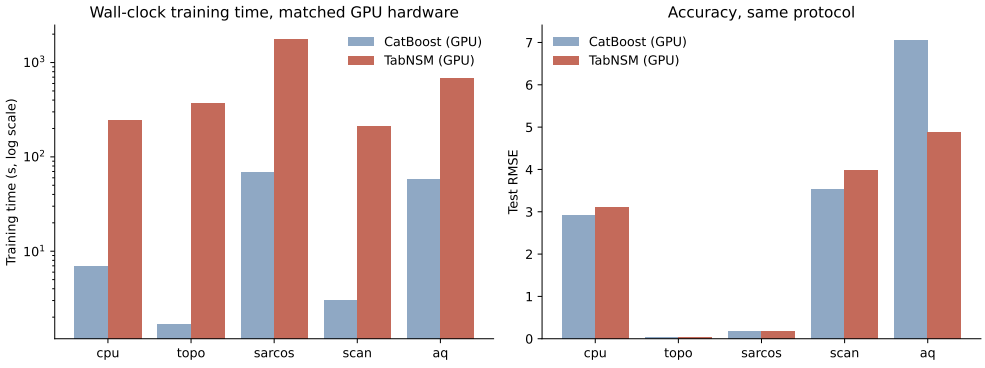


Figure 10: Table 19 in figure form: wall-clock training time (log scale, left) and test RMSE (right) on matched GPU hardware.

J Efficiency and Memory Footprint

To avoid confounding hardware with method, we compare CatBoost and TabNSM on the same NVIDIA RTX A4000 GPU (CatBoost with `task_type="GPU"`). The timing boundary covers model fitting after data loading; both methods use validation monitoring, but their patience and checkpoint-restoration logic are not identical, so this comparison controls for the hardware confound rather than serving as an intrinsic speed benchmark. Table 19 and Figure 10 report one seed on the five datasets loadable by the released pipeline plus TabArena's 1,024-feature QSAR-TID-11.

Under these settings CatBoost is 11.8–220.5× faster and has lower RMSE on three of the five datasets from the nine-dataset suite. On QSAR-TID-11, TabNSM wins narrowly on accuracy (0.4% margin) while CatBoost remains 15.5× faster; since that dataset also has fewer rows than four of the five others, it does not show the wall-clock gap narrowing with feature count. We therefore make no empirical wall-clock efficiency claim for TabNSM; the complexity analysis in Section 3 describes the dependence of the architecture on D and is kept separate from measured end-to-end training time.

Tables 20 and 21 report memory usage on the three largest datasets from the original (CPU CatBoost) protocol; they are retained as descriptive measurements only and are not used to support an efficiency claim. We additionally measure two pretrained tabular foundation models, TabDPT [17] and TabPFNv2 [14], on the same train/test tensors under identical instrumentation (`torch.cuda.max_memory_allocated` around `fit+predict`; process RSS delta for CPU) on the same RTX A4000. TabDPT uses its default retrieval-based context (size 2048, 8 ensemble members) rather than the full training set, so it completes on all three datasets; TabPFN attends over the full in-context training set with no such subsampling and receives `ignore_pretraining_limits=True` to bypass its built-in sample/feature caps, which lets it attempt – and exhaust available memory on – all three.

Table 22 lists the exact per-dataset values summarized in Figure 2 of the main text.

Table 20: Peak GPU memory usage in MB. CatBoost is run on CPU, so GPU memory is not applicable. TabPFN's entries are the peak reached immediately before the out-of-memory failure (Table 21 note), not a completed run.

Dataset	CatBoost	TabNSM	TabDPT	TabPFN
EVP	N/A	**477**	1939	OOM (5873)†
ChD	N/A	**529**	1586	OOM (4654)†
CD	N/A	**1621**	4607	OOM‡

Table 21: CPU memory increase in MB during training. Lower is better.

Dataset	CatBoost	TabNSM	TabDPT	TabPFN
EVP	3763.77	3006.73	**265.26**	OOM†
ChD	4325.94	3408.70	**196.60**	OOM†
CD	12080.61	8524.16	**195.80**	OOM‡ (≥43,396)

† CUDA out-of-memory on the 15.72 GiB A4000: ChD tried to allocate 33.73 GiB, EVP 42.66 GiB.

‡ On CD, TabPFN's host-side preprocessing was killed by the Linux OOM-killer (RSS 43,396 MB) before any tensor reached the GPU; the machine has 62 GB RAM. This is a lower bound, not a completed measurement, and GPU peak is therefore N/A on CD.

TabDPT keeps a fixed-size (2048-row) in-context subsample on GPU regardless of training-set size, so its CPU increase stays small and roughly constant across datasets, trading GPU memory (Table 20) for CPU memory – a different trade-off from TabNSM/CatBoost rather than a strictly smaller footprint. TabPFN has no such subsampling: it holds the entire training set in its attention context, which is what causes the failures here even with pretraining limits disabled.

K Sparse Feature-wise Attention: Full Formulation

Within ASIM, we treat the D input features as a sequence of token embeddings $X_{\text{tok}} \in \mathbb{R}^{D \times d_{\text{tok}}}$ and apply a structured sparse self-attention operator $\mathcal{A}_{\text{sparse}}$ [3; 81]. Rather than computing all D^2 pairwise feature interactions, each feature token attends only to a compact, instance-adaptive subset of other features. This subset is constructed from three complementary components designed to jointly cover local, global, and instance-specific interaction patterns.

Local neighborhood. Features adjacent in the input ordering often encode correlated measurements (e.g., related embedding dimensions or grouped sensor readings). For feature d, we retain the s nearest neighbors as a sliding window:

$$\mathcal{K}_d^{\text{win}} = \mathbf{k}_{d-s:d+s}, \quad \mathcal{V}_d^{\text{win}} = \mathbf{v}_{d-s:d+s}. \tag{10}$$

Compressed global summary. To give each query feature coarse awareness of the full feature set, we partition the D features into blocks of size b (stride $r \le b$) and compress each block to a single representative key–value pair via a learnable MLP ϕ:

$$\mathcal{K}_d^{\text{cmp}} = \left\{ \phi(\mathbf{k}_{ib+1:\,ib+b}) \;\middle|\; 0 \le i \le \left\lfloor \tfrac{D-b}{r} \right\rfloor \right\}, \tag{11}$$

with an analogous $\mathcal{V}_d^{\text{cmp}}$. This reduces global context to $\mathcal{O}(D/r)$ keys.

Instance-adaptive block selection. The compressed attention scores in Equation (11) provide a low-cost proxy for identifying which feature groups are most relevant to the current query. We compute:

$$\mathbf{p}_d^{\text{slc}} = \text{Softmax}\big(\mathbf{q}_d^\top \mathcal{K}_d^{\text{cmp}}\big), \tag{12}$$

rank all blocks by $\mathbf{p}_d^{\text{slc}}$, and retrieve the original full-resolution keys and values for the top-m blocks:

$$\mathcal{I}_d = \{i \mid \text{rank}(\mathbf{p}_d^{\text{slc}}[i]) \le m\}, \quad \mathcal{K}_d^{\text{slc}} = \text{Cat}\big[\{\mathbf{k}_{ib+1:(i+1)b} \mid i \in \mathcal{I}_d\}\big]. \tag{13}$$

This recovers fine-grained interactions for the most relevant feature groups with a fixed budget of mb keys per query.

Table 22: Values behind Figure 2 for the four largest datasets: test RMSE (single run, from Table 3) and, in parentheses where measured, peak GPU memory in MB on an RTX A4000. OOM: out of memory; n/a: outside the model's hard-coded sample limit. CatBoost runs on CPU.

Model	**CD** (1.0M×401)	**ChD** (309k×405)	**EVP** (265k×392)	**RES** (1.05M×389)
TabPFNv2	n/a (>10k rows)	n/a	n/a	n/a
TabPFN, limits disabled	OOM (host RAM)	OOM (33.7 GiB req.)	OOM (42.7 GiB req.)	–
TabFM	OOM (80 GB)	OOM (80 GB)	OOM (80 GB)	OOM (80 GB)
TabDPT	0.675 (4607)	23006 (1586)	8.353 (1939)	3164
TabICLv2	0.007	22771	1.897	3164
CatBoost	0.009	19000	3.291	3144
Best transformer	0.007	4475	1.507	3163
TabNSM	0.007 (**1621**)	**4460** (**529**)	**1.199** (**477**)	**3103**

Gated aggregation. The three branch outputs are merged via input-dependent gating scalars $g_d^c \in [0,1]$, produced by a small MLP with sigmoid activation applied to the query embedding:

$$\tilde{X}_d = \sum_{c \in \{\mathrm{win},\,\mathrm{cmp},\,\mathrm{slc}\}} g_d^c \cdot \mathrm{Attn}(\mathbf{q}_d,\, \mathcal{K}_d^c,\, \mathcal{V}_d^c)\,. \tag{14}$$

The total attended context per feature is $L = 2s + \lfloor (D-b)/r \rfloor + mb \ll D$, giving overall complexity $\mathcal{O}(D\, d_{\mathrm{tok}}\, L)$, sub-quadratic in D for fixed sparse hyperparameters (s, b, r, m).

L Feature-Token Mixing (FTM): Full Formulation

Within ASIM, the FTM operator $\mathcal{M}(\cdot)$ acts on the combined representation $X_{\mathrm{tok}} + \tilde{X} \in \mathbb{R}^{D \times d_{\mathrm{tok}}}$ produced after sparse attention, where D feature tokens each carry a d_{tok}-dimensional embedding. Its role is to propagate information across both the feature axis and the embedding axis using lightweight MLPs, complementing the sparse attention output with global mixing at low computational cost.

The operator applies two parallel mixing branches whose outputs are fused by element-wise gating:

$$U_1 = \mathrm{GELU}\big(W_2\big[\mathrm{ReLU}(W_1\,\mathrm{LN}(X)^\top + b_1)\big] + b_2\big)^\top \tag{15}$$

$$U_2 = W_4\big(\mathrm{ReLU}(W_3 \cdot \mathrm{LN}(X) + b_3)\big) + b_4 \tag{16}$$

$$U_3 = \mathrm{SiLU}(\mathrm{LN}(U_1 \odot U_2)) \tag{17}$$

$$Z = X + U_3, \tag{18}$$

where $W_1, W_2 \in \mathbb{R}^{d_{\mathrm{tok}} \times d_{\mathrm{tok}}}$ and $W_3, W_4 \in \mathbb{R}^{D \times D}$ are learnable weights; $\mathrm{LN}(\cdot)$ denotes Layer Normalization; and $\odot$ is element-wise multiplication.

The two branches serve distinct mixing roles within ASIM. The *embedding-wise* branch (U_1, Equation (15)) transposes the input so that the MLP operates across the d_{tok} embedding dimensions for each feature independently, capturing intra-token structure. The *feature-wise* branch (U_2, Equation (16)) applies the MLP directly along the D feature tokens, capturing cross-feature dependencies that may fall outside the sparse attention mask. Their outputs are combined via element-wise multiplication and SiLU gating (U_3, Equation (17)), and added back through a residual connection (Z, Equation (18)), enabling stable gradient flow while enriching the representation with globally mixed context.

M Additional Ablation Studies and Sensitivity Analyses

M.1 Matched Component Ablation: Full Table

Table 23 gives the full result of the component-removal protocol described in Section 4.2.

M.2 Operator-Swap Comparison from an Earlier Pipeline

Figure 12 compares, within TabNSM, the sparse attention operator against dense multi-head attention and a Mamba block on SA, EVP, ChD, and CD. These are single runs from an earlier version of the training pipeline: the ASIM bar on ChD lies well above the corresponding entry of Table 3 (4460),

Table 23: Matched component removal (single seed, terminal checkpoint). Positive values mean removal increases RMSE. The −GridLoss column is reported for both an unnormalized implementation and the normalized implementation of Eq. (4).

Dataset	−ASIM	−RISE	−GridLoss (unnormalized)	−GridLoss (corrected)
CP	+9.6%	+17.3%	−1.3%	+2.9%
TO	+18.1%	−2.5%	−5.2%	+3.2%
SA	+0.8%	−1.8%	−9.1%	−3.1%
SC	+0.6%	−0.4%	−1.5%	−0.8%
AQ	+1.5%	−4.4%	−17.0%	+8.4%
Mean	**+6.1%**	**+1.6%**	**−6.8%**	**+2.1%**

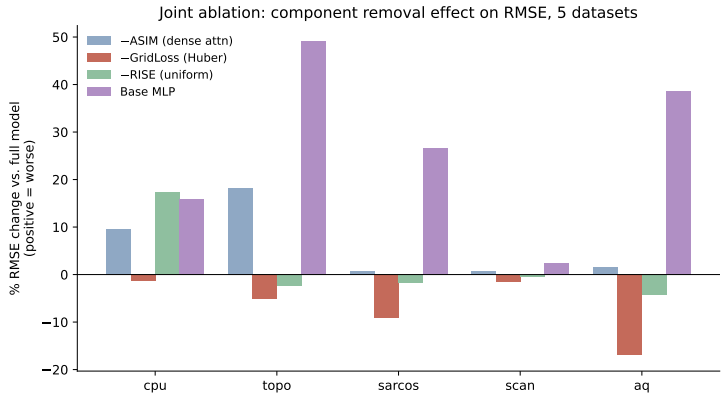


Figure 11: Percentage RMSE change from removing each component under the unnormalized GridLoss implementation (Table 23, first three data columns).

and on ChD the three operators are within a factor of 1.2 of one another with Mamba lowest. We therefore read the figure as showing that dense attention is worst on every dataset and unstable on the wide ones (EVP, CD), and that a linear-time sequence mixer does not close the gap on SA, EVP, and CD, not as a precise ranking; the matched, same-pipeline comparison against dense attention is the −ASIM column of Table 23.

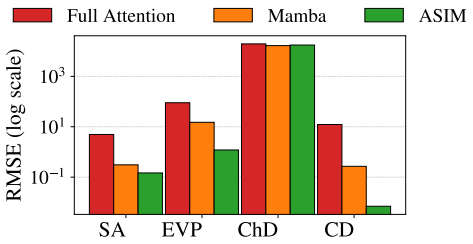


Figure 12: Operator swap inside TabNSM on four datasets (single runs, earlier pipeline; log scale). ASIM has the lowest RMSE on SA, EVP, and CD; on ChD the three operators are nearly tied.

M.3 Ablation of the FTM Pathway

We ablate the FTM pathway by removing it while keeping the sparse attention backbone and training protocol fixed. Table 24 shows that ablating FTM degrades RMSE on CP, TO, SA, and EVP, with particularly large drops on EVP (570.09 vs. 1.199). This indicates that the FTM pathway contributes to the overall model performance, though sparse attention alone provides the primary architectural innovation.

M.4 Ablation on Loss Function

We compare standard pointwise losses, including MSE, MAE, and Huber, a logarithmic robust variant, LogCosh, and GridLoss across three representative datasets with different scales and noise characteristics. Figure 13 reports RMSE on a log scale, with relative changes annotated with respect

Table 24: Ablation of the FTM pathway. Lower RMSE is better.

	CP	TO	SA	EVP
w/o FTM	3.966	0.034	0.176	570.09
Full TabNSM	**2.842**	**0.021**	**0.137**	**1.199**

to GridLoss. This figure was produced with the unnormalized GridLoss implementation described in Section 3 and is retained for completeness; the matched removal ablation in Table 23 (Section 4.2), which uses the corrected implementation of Eq. (4), is the result we rely on.

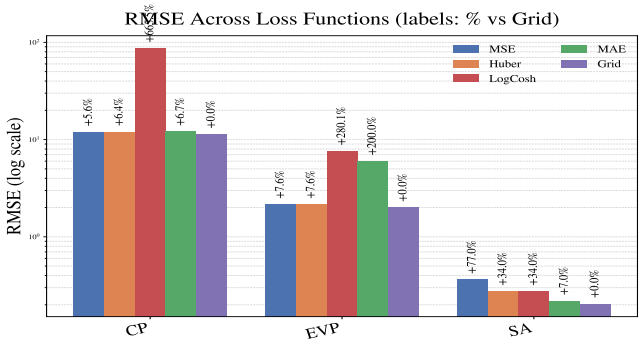


Figure 13: **Loss-function ablation across three datasets.** RMSE is reported on a log scale. Values above bars indicate relative change with respect to GridLoss. GridLoss achieves the lowest or comparable RMSE across the evaluated datasets, while pointwise losses such as MAE and LogCosh can degrade substantially in high-error regimes.

In Figure 13, GridLoss is competitive with Huber and less sensitive than MAE and LogCosh to scale variation and large residuals. Taken together with Table 23, however, the evidence for a consistent GridLoss benefit is mixed: with the corrected implementation, removing GridLoss increases RMSE on CP, TO, and AQ and decreases it on SA and SC, all single-seed. The full model uses the blended objective

$$\mathcal{L} = \gamma \mathcal{L}_{\text{Huber}} + (1 - \gamma)\mathcal{L}_{\text{Grid}},$$

with γ fixed across all experiments; sensitivity to γ and τ is analyzed in Appendix M.9.

Comparison with a histogram-based objective. To further contextualize GridLoss, we compare it against an established histogram-based regression objective, HL-Gauss [41]. Table 25 and Figure 14 report a diagnostic run of an HL-Gauss-style objective with the same architecture, split, initialization seed, and epoch budget on five datasets, alongside the Huber–GridLoss blend. The diagnostic is numerically lower on TO, SA, and SC, tied on CP, and higher on AQ. It is *not* a protocol-matched comparison: the histogram run standardizes the target and disables RISE whereas the blended run does not, it uses percentile-clipped support and a different smoothing-width convention from the original HL-Gauss recipe, and both scripts evaluate the terminal rather than the best-validation checkpoint. We therefore do not use it to claim that either objective is superior; a controlled comparison must hold preprocessing, sampling, checkpoint selection, bin support, and validation tuning fixed.

Table 25: Blended Huber–GridLoss versus an HL-Gauss-style diagnostic (test RMSE, one seed; not a protocol-matched comparison, see text).

Dataset	**Huber + GridLoss**	**HL-Gauss (diagnostic)**
CP	3.106	3.109
TO	0.0308	0.0277
SA	0.1715	0.1631
SC	3.990	3.978
AQ	4.887	8.550

M.5 Ablation on effect of RISE

We evaluate the impact of RISE by disabling it and reverting to uniform sampling. In the single-dataset multi-run comparison of Figure 16, removing RISE increases both the mean validation error and the run-to-run variance. The matched test-set ablation across five datasets in Table 23 shows

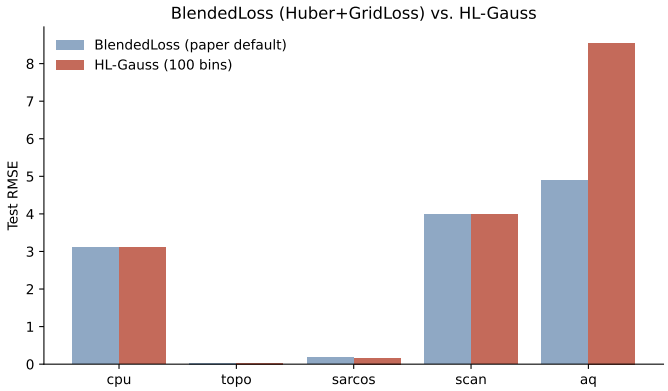


Figure 14: Table 25 in figure form.

where this matters: removing RISE increases test RMSE substantially on CP ($+17.3\%$), while on the other four datasets the change (-0.4% to -4.4%, single seed) lies within the seed variability of Appendix I. We therefore describe RISE as a low-cost refinement that is neutral where it is not helpful, rather than as a consistent improvement.

M.6 Column-Order Sensitivity

Because ASIM's sliding-window branch is defined with respect to the input column ordering, we tested how sensitive the trained model is to that ordering. Column order is a fixed, deterministic property of a table, not a random nuisance at inference time, so the practical question is whether the model is *stable* under a fixed ordering and whether ordering can be *chosen* well; both are answered below. These experiments use the released pipeline's default configuration and terminal checkpoint (the protocol of Section 4.2), so the reference RMSE values differ from the tuned, best-validation entries of Table 3. First, we retrained TabNSM under five independent random column permutations (applied consistently to train, validation, and test), with the training seed fixed, on CP (12 features) and TO (266 features). Table 26 shows an average change of $\approx 5\%$, within the 2–13% seed-to-seed variability of Appendix I, with substantial spread across permutations; on CP one permutation outperforms the original order. Second, we compared the original index order with a correlation-based order (hierarchical-clustering leaf order on the training-split correlation matrix, so that correlated features become adjacent) on all five loadable datasets (Table 27, Figure 15). Four of five datasets show the same small, sign-mixed pattern ($\leq 5.2\%$ either direction); AQ is the notable case, where correlation-based ordering *improves* RMSE by 19.2%. These tests characterize the whole model rather than the sparse-attention branch in isolation. They show that order sensitivity is bounded (comparable to seed variability under arbitrary permutations) and that ordering is a lever rather than a liability: a one-line, data-driven preprocessing step (clustering correlated columns together, which requires no labels) either leaves RMSE unchanged or improves it, in the AQ case substantially. We do not claim permutation invariance, and note that GBDTs and MLPs are not order-dependent while dense- and sparse-attention models with positional structure generally are. Randomly reshuffling columns at every forward pass would change feature identity unless all associated parameters were permuted consistently, so we do not present per-forward reshuffling as an invariance mechanism.

Table 26: Fixed-column permutation test (training seed fixed; mean $\pm$ standard deviation over five column permutations; single seed per permutation).

Dataset	Original RMSE	Permuted RMSE	Mean change
CP (12 features)	3.1065	3.2612 ± 0.3673	+5.0%
TO (266 features)	0.03076	0.03221 ± 0.00343	+4.7%

M.7 Embedding-Block Ablation: PCA Projection and Noise Padding

Five of the seven wide datasets take 95–99% of their columns from a 384-dimensional sentence-embedding block (Appendix H). We tested whether TabNSM's advantage depends on the structure of that block or on input width and redundancy in general. Only the columns named `emb_*` are

Table 27: Index-based (original) versus correlation-based column ordering, five datasets, single seed.

Dataset	Index-based RMSE	Correlation-based RMSE	Change
CP	3.1065	3.2212	+3.7%
TO	0.03076	0.03033	−1.4%
SA	0.1715	0.1803	+5.2%
SC	3.9900	3.9940	+0.1%
AQ	4.8873	3.9479	−19.2%

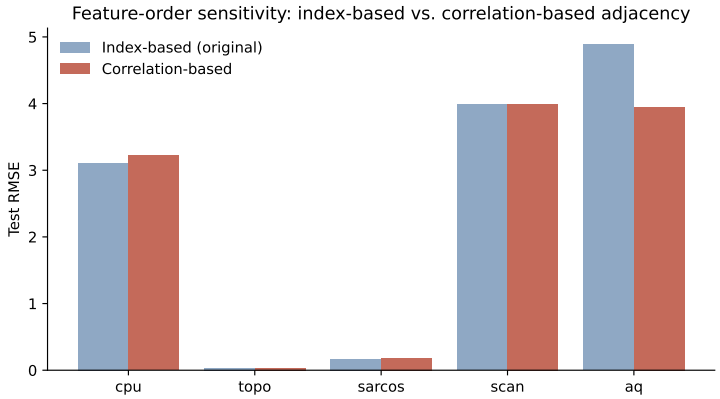


Figure 15: Table 27 in figure form.

changed; the 6–8 structured columns are untouched and stay first. The block is replaced by one of the following, with every transform fitted on the training split only (Table 28): its top-k principal components ($k \in \{64, 128\}$, retaining 95% and 99% of the block's variance on AQ, 91% and 97% on EVP); the same components plus $384 - k$ i.i.d. $\mathcal{N}(0, 1)$ noise columns, which restores the original D (the setting posed by the reviewer); the same components plus $384 - k$ *redundant* columns, each a random unit-norm mixture of the components plus noise (unit variance, correlated with the signal); or the original block plus 384 noise columns (D doubled). Because every padded condition also shuffles the block's column order, we include the original block in a random order as a control. Splits, preprocessing, and the default TabNSM configuration are those of Section 4.2.

Protocol and what differs from the main tables. We ran AQ and EVP. The component-ablation harness does not produce a usable predictor on ChD, CD, or RES under the released configuration (training loss becomes NaN and the predictions are constant on all three), so those datasets are not included and this experiment covers two of the five embedding-derived datasets. Seeds change the noise draw, the column permutation, the mixing weights, and the training initialization: three seeds for AQ, and two seeds for EVP TabNSM and CatBoost (three for TabICLv2); dense-attention TabNSM (ASIM replaced by dense multi-head attention, as in Section 4.2) is a single seed. TabFM is reported on AQ only, since it exhausts GPU memory on EVP (Table 3). CatBoost is run with default GPU settings and validation early stopping, and TabICLv2 with defaults. *Precision and hardware:* AQ TabNSM runs use the released fp16 mixed-precision setting on H100 GPUs without incident. On EVP the released fp16 trainer diverged to NaN in 15 of the 18 runs that completed, including the original-input control for both seeds; a diverged run returns constant predictions (RMSE 90.5). We therefore report EVP TabNSM and dense-attention runs with bfloat16 autocast and no loss scaling, on L40S GPUs, for every condition; none of these 27 runs diverged. RMSE values are terminal-state test RMSE of the default configuration, so, as in Appendix M.6, they differ from the tuned, best-validation entries of Table 3 (for example, TabNSM on EVP is 1.50 here against 1.199 there), and comparisons should be read within the table.

Findings. *(i) Compression alone is mostly benign.* On AQ, TabNSM is unchanged under PCA-64 (4.15 vs. 4.18 with the original block) and PCA-128 (3.87). On EVP it is worse by 22% at $k = 128$ (1.83 vs. 1.50) and by 79% at $k = 64$ (2.68), retaining its lead over TabICLv2 (1.94) only at $k = 128$. The random column order by itself is within seed variability (AQ 4.25 vs. 4.18; EVP 1.68±0.40 vs. 1.50±0.12). *(ii) TabNSM does not retain its advantage under the reviewer's setting, PCA followed by noise padding to the original D.* Its RMSE rises 2.7× (PCA-64) and 2.0× (PCA-128) on AQ

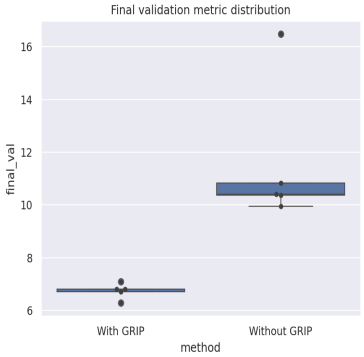


Figure 16: Final validation metric distributions across multiple runs for models trained with and without RISE.

Table 28: Test RMSE (lower is better; mean ± standard deviation over seeds, none for the single-seed dense-attention column) when the 384-dimensional embedding block is replaced as described in the text; D is the resulting feature count. Best per row in bold. "Noise" and "redundant" pad back to the original D. EVP TabNSM and dense-attention entries use bfloat16 on L40S GPUs; all other entries use the released settings. TabFM does not run on EVP.

Input condition	D	**TabNSM**	**Dense attn.**	**CatBoost**	**TabICLv2**	**TabFM**
AQ						
Original 384-d block	390	4.18±0.51	6.87	6.99±0.07	6.42±0.00	**3.58±0.00**
Original, columns permuted	390	4.25±0.21	5.43	6.97±0.16	6.23±0.10	**3.58±0.02**
PCA-64	70	4.15±0.21	4.64	6.83±0.30	5.64±0.04	**3.60±0.01**
PCA-128	134	3.87±0.35	3.89	6.93±0.53	5.52±0.07	**3.76±0.01**
PCA-64 + noise	390	11.16±0.71	15.20	8.16±0.04	5.61±0.09	**3.31±0.02**
PCA-128 + noise	390	8.34±0.34	10.39	7.83±0.09	5.42±0.08	**3.45±0.02**
PCA-64 + redundant	390	7.95±0.50	8.60	8.96±0.31	10.40±0.48	**3.80±0.08**
PCA-128 + redundant	390	5.86±0.17	7.84	8.22±0.12	7.55±0.03	**3.67±0.08**
Original + 384 noise	774	8.08±0.41	9.57	7.58±0.14	6.79±0.04	**3.46±0.03**
EVP						
Original 384-d block	392	**1.50±0.12**	2.61	3.21±0.01	1.90±0.00	–
Original, columns permuted	392	**1.68±0.40**	1.99	3.22±0.01	1.88±0.05	–
PCA-64	72	2.68±0.10	2.39	3.16±0.00	**1.94±0.03**	–
PCA-128	136	**1.83±0.00**	2.18	2.87±0.02	1.94±0.01	–
PCA-64 + noise	392	3.98±0.38	4.36	3.80±0.03	**2.10±0.01**	–
PCA-128 + noise	392	2.27±0.17	2.73	3.29±0.11	**1.93±0.06**	–
PCA-64 + redundant	392	3.20±0.14	3.77	3.69±0.00	**3.19±0.03**	–
PCA-128 + redundant	392	**2.15±0.10**	3.24	3.21±0.02	2.72±0.03	–
Original + 384 noise	776	**2.21±0.13**	2.87	3.39±0.01	2.37±0.02	–

and 2.7× and 1.5× on EVP. The baselines move much less: CatBoost by $+2\%$ to $+18\%$, TabICLv2 by -16% to $+11\%$, and TabFM on AQ by -7% to -4%. On EVP, TabICLv2 is best in both noise conditions (2.10 and 1.93 against TabNSM's 3.98 and 2.27), and TabNSM falls behind CatBoost at $k = 64$ (3.98 vs. 3.80). On AQ, where TabNSM was already behind TabFM with the original block, it also falls behind CatBoost and TabICLv2. *(iii) Redundant padding is less harmful.* TabNSM's RMSE rises 1.4–2.1×; on EVP it remains the best model at $k = 128$ (2.15 vs. 2.72 for TabICLv2 and 3.21 for CatBoost) and ties TabICLv2 at $k = 64$ (3.20 vs. 3.19). TabICLv2 itself degrades under this padding (up to $+68\%$). Adding 384 noise columns to the intact block (D doubled) raises TabNSM's RMSE 1.9× on AQ and 1.5× on EVP; on EVP it is still ahead of TabICLv2 (2.21 vs. 2.37). *(iv) Sparse versus dense attention.* The sparse module has lower RMSE than dense attention in 17 of the 18 conditions (single-seed dense runs; the exception is EVP with PCA-64), so the sparse-versus-dense comparison of Section 4.2 persists under these transforms, although the margin over the other baselines does not.

Reading. On the two datasets tested, TabNSM's lead over the strongest baselines is tied to the intact embedding block rather than to width or redundancy in general: on EVP it holds for the original block, for its random reordering, at PCA-128, and under redundant or additive padding, but not once the block is compressed and padded with independent noise. The result does not support reading the

wide-table advantage as robustness to noise features, and Section 5 scopes it accordingly. A plausible contributor is that ASIM's sliding-window and block-selection branches are defined over column positions (Appendix M.6) and the padded columns place independent noise between informative ones; we did not test placing the components contiguously, so this is a hypothesis. Other limits: two of the five embedding-derived datasets, two to three seeds per cell, one embedding encoder, linear (PCA) compression only, and precision and hardware that differ between AQ and EVP for TabNSM.

M.8 Sensitivity to Feature Dimensionality and Sample Size

We evaluate robustness to changing feature dimensionality and training-set size. For feature-dimensionality analysis, we vary the *feature fraction*, defined as the proportion of retained input features for each dataset, while keeping the number of rows fixed. For each fraction, we subsample feature columns using a fixed random seed and keep all other training settings identical to Section C. For sample-size analysis, we vary the fraction of training samples while keeping the full feature set fixed.

Figure 17 reports MSE for TabNSM and baseline models on the TO and SC datasets. In the feature-fraction setting, TabNSM maintains stable performance across increasing dimensionality, whereas the baseline exhibits higher error and more pronounced non-monotonic behavior. In the sample-fraction setting, TabNSM remains competitive under reduced data availability. These trends suggest that TabNSM is robust to both high-dimensional feature spaces and limited training data. GridLoss and RISE further contribute to stability by providing structure-aware gradients and emphasizing difficult examples during training.

M.9 Sensitivity to GridLoss Hyperparameters

We analyze the sensitivity of GridLoss to the temperature parameter τ and the blending coefficient γ in the combined objective:

$$\mathcal{L} = \gamma \mathcal{L}_{\text{Huber}} + (1-\gamma)\mathcal{L}_{\text{Grid}}.$$

Figure 18 reports RMSE as a function of τ across several datasets (CP, TO, and SCAN-2, abbreviated as SC). Performance varies smoothly over multiple orders of magnitude, indicating that GridLoss is not overly sensitive to the choice of τ within a broad range. In the main experiments, we use a fixed default setting for both τ and γ, while tuning the number of grid bins during training based on dataset-specific target statistics. The effects of varying τ and γ are analyzed separately below.

M.10 Sensitivity of τ to Target Distribution Properties

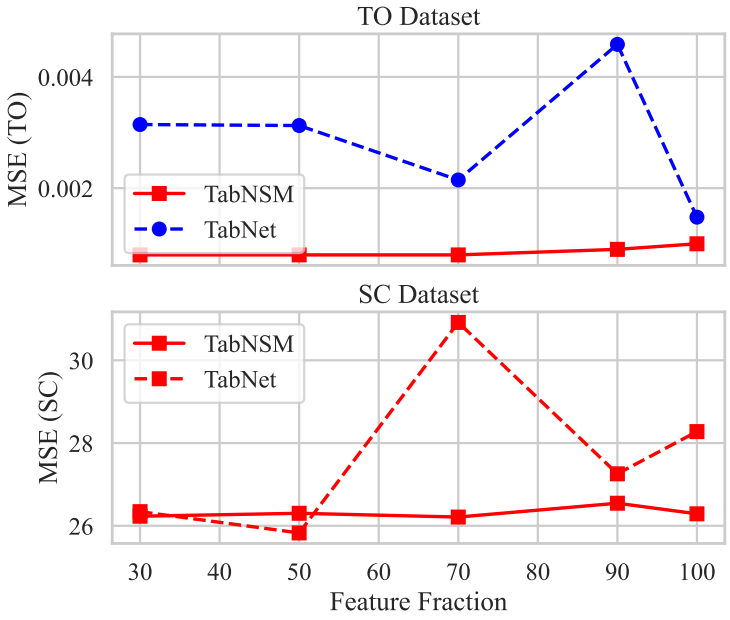
Table 29 summarizes the relationship between target distribution characteristics and the observed sensitivity of RMSE to the regularization parameter τ. We find that the optimal choice of τ is primarily governed by the concentration of the bulk of the target distribution, as quantified by the ratio $\mathrm{IQR}/\mathrm{Std}$, rather than by tail heaviness alone.

For CP, the target distribution exhibits a moderately dispersed bulk with heavy tails. Increasing τ consistently improves robustness, resulting in a monotonic decrease in RMSE. In contrast, TO has a tightly concentrated bulk with rare but extreme outliers, favoring small τ values that preserve precision on the dominant mass while avoiding excessive smoothing. Finally, SCAN-2 is characterized by a degenerate bulk ($\mathrm{IQR} = 0$) and extremely rare but large outliers, leading to a highly non-monotonic RMSE–τ relationship in which very small τ values perform best.

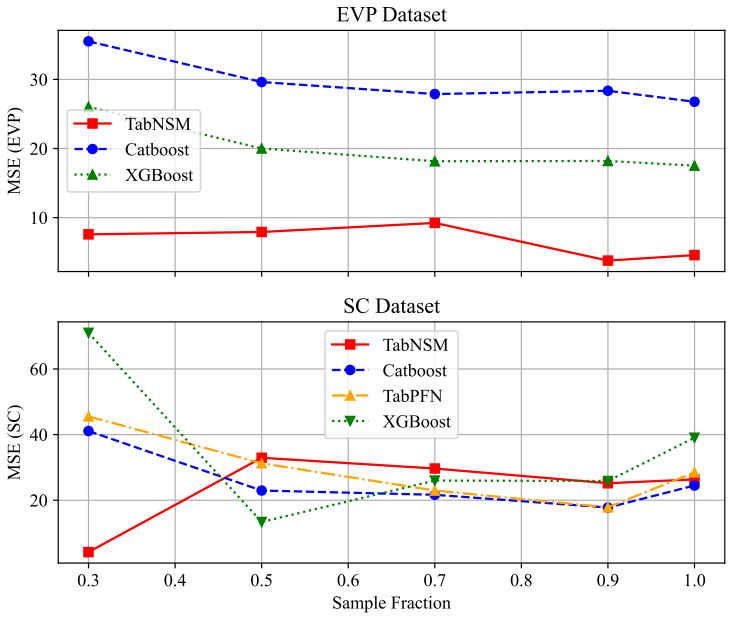
These observations suggest that τ should be adapted to the effective bulk structure of the target distribution. Relying solely on tail statistics such as skewness or kurtosis is insufficient for selecting an appropriate regularization strength.

Table 29: Target distribution statistics and their relationship to τ sensitivity.

Dataset	Std	IQR	IQR/Std	Tail Type	Preferred τ	RMSE–τ Behavior
CP	18.40	13.00	0.71	Heavy-tailed	Large	Monotonic decrease
TO	0.03	0.02	0.67	Rare extremes	Small	Sharp peak
SCAN-2	4.99	0.00	0.00	Degenerate bulk	Very small	Highly non-monotonic



(a) MSE versus feature fraction on the TO and SC datasets.



(b) MSE versus training sample fraction on the TO and SC datasets.

Figure 17: **Sensitivity to feature dimensionality and sample size.** The top panel evaluates robustness under varying numbers of retained input features, while the bottom panel evaluates robustness under varying fractions of training samples. Separate y-axes are used to accommodate differences in error scale.

M.11 Effect of the Blending Coefficient γ

We next study the impact of the blending coefficient γ, which controls the trade-off between the Huber loss and the grid-based loss. As shown in Figure 19, relying on either loss alone ($\gamma = 0$ or $\gamma = 1$) leads to degraded performance. In contrast, intermediate values of γ consistently achieve lower normalized RMSE across all datasets, with optimal performance observed for $\gamma \in [0.4, 0.6]$. These results indicate that the two loss terms are complementary and that balancing robustness with structural regularization is critical for optimal performance.

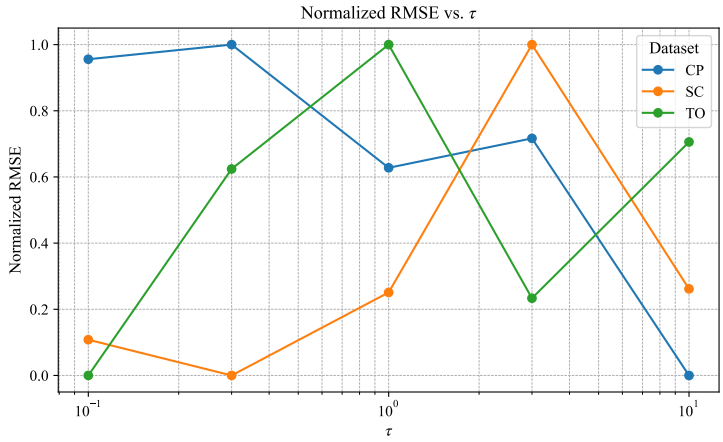


Figure 18: Normalized RMSE as a function of τ across datasets.

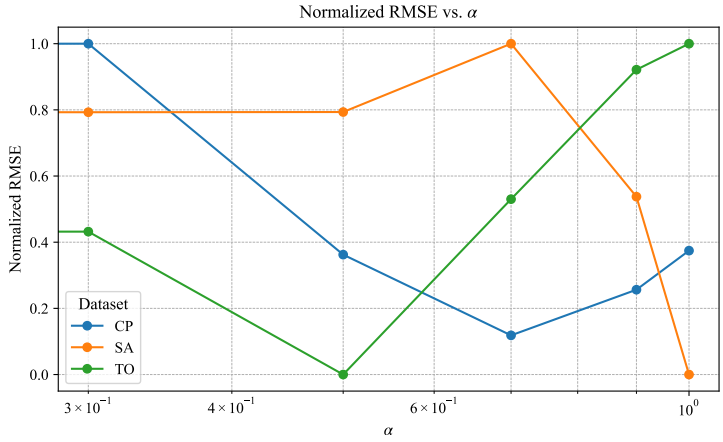


Figure 19: Normalized RMSE as a function of γ across datasets.

M.12 Component-wise Ablation of Sparse Attention Parameters

We conduct a component-wise ablation study to assess the contribution of each sparse attention hyperparameter to model performance (Figure 20). In each experiment, a single hyperparameter is varied while all others are held fixed, and performance is evaluated using average validation RMSE across four validation sets. Lower RMSE indicates better performance. For the *compress block size*, a value of 10 achieves the best average validation RMSE, with performance degrading for both smaller and larger values. The *number of selected blocks* performs best at 3, while deviations in either direction reduce effectiveness. Similarly, a *selection block size* of 10 yields the lowest validation RMSE among the tested values. Finally, a *sliding window size* of 6 provides the best performance, suggesting the importance of selecting an appropriate local context size. Based on these findings, we restrict the hyperparameter search space in the first-stage optimization to improve search efficiency. Nevertheless, joint hyperparameter optimization remains necessary to achieve optimal performance across datasets.

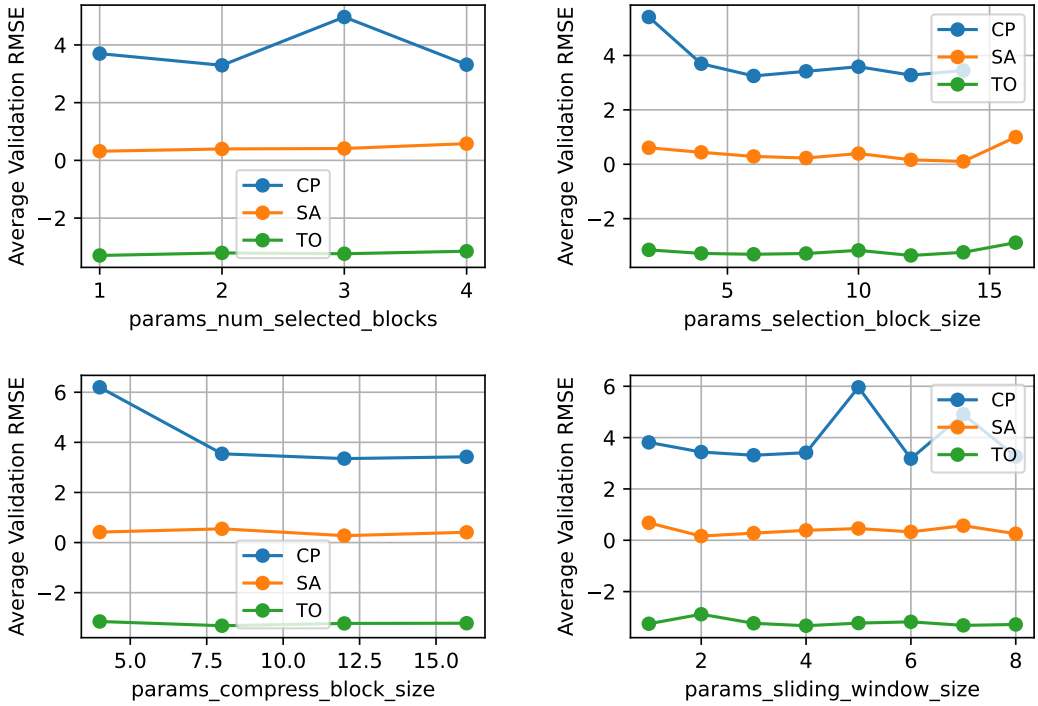


Figure 20: Component-wise ablation of sparse attention parameters. Each plot shows the effect of varying one parameter: *number_of_selected_blocks* (top-left), *selection_block_size* (top-right), *compress_block_size* (bottom-left), and *sliding_window_size* (bottom-right) on the average validation RMSE across four datasets.