TABSQRT: SELF-QUERY REPRESENTATION TRANSFER FOR TABULAR IN-CONTEXT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Tabular foundation models (TFMs) predict query labels from labeled context examples; during training, the query labels themselves serve only as loss targets. We ask a different question: if a frozen TFM is allowed to see the answers, can it teach a student how to represent examples whose answers are hidden? TabSQRT (Self-Query Representation Transfer) turns this idea into supervision. TabSQRT constructs targets by supplying each labeled training block to a frozen TFM as both context and queries. A conditional diffusion student learns to generate the resulting decoder-input representations from standard in-context inputs, followed by classification post-training through the frozen decoder. In diagnostics with students trained separately on each dataset, larger target-construction blocks improve diffusion-student accuracy even as target decoding accuracy declines. This finding motivates assessing supervision through downstream student performance. Trained across 100,000 real-world tables, TabSQRT achieves 88.65% mean accuracy on 38 TabArena classification datasets without dataset-specific fine-tuning, compared with its LimiX teacher's 88.30%, and ties the highest reported mean accuracy among current frontier TFMs. Ablations support combining self-query targets, diffusion-based representation learning, and classification post-training for transfer to unseen tables.

1 INTRODUCTION

Tabular foundation models (TFMs) perform in-context learning (ICL) to predict on unseen datasets from labeled examples, without dataset-specific parameter updates. TabPFN acquires this capability through pretraining on synthetic tasks (Hollmann et al., 2023), while TabDPT learns from real-world tables (Ma et al., 2025). TabICL further extends tabular ICL to larger datasets (Qu et al., 2025). In these models, the labeled context specifies the prediction task and shapes the representation of each query. The same query example can therefore receive different representations depending on the examples and labels supplied alongside it. The context specifies the task and shapes each query's representation. This dependence makes a frozen TFM a configurable source of supervision: changing its context changes the representation target assigned to the same training example. We investigate how to use this flexibility to improve prediction across tables.

Knowledge distillation transfers teacher predictions (Hinton et al., 2015), while representation distillation uses intermediate features as supervision (Romero et al., 2015). Generalized distillation further allows teachers to use information unavailable to students at inference (Lopez-Paz et al., 2016). For tabular ICL, query labels already available during supervised training can enter the teacher's context and shape its representations. This suggests a question beyond their use in a classification loss: *how should a frozen TFM construct label-informed representation targets that remain useful to a student without access to query labels?*

We construct targets through **self-querying**. A labeled training block serves as the teacher's context, and the same feature vectors are presented again as unlabeled queries. Each query can thus access its own label through the context. We extract representations at the frozen decoder's input, where they depend on both the query and its labeled block. Transfer is challenging because the student must predict from standard inputs that omit query labels and may contain a different context. Consequently, decoding a privileged target correctly does not establish that it provides effective supervision. Earlier distillation work identified limitations of teacher accuracy as a guide to student

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.

performance(Cho & Hariharan, 2019). Here, we study target construction by varying the context of a fixed teacher.

Our framework, **Tabular Self-Query Representation Transfer (TabSQRT)**, trains a conditional diffusion student to predict these cached targets from the frozen TFM's standard encoder embeddings. The denoising objective trains the student to reconstruct targets across noise levels using the EDM formulation (Karras et al., 2022). Classification post-training then uses the frozen decoder to penalize errors in the resulting class predictions. At inference, the student generates query representations from labeled context and unlabeled queries, and the decoder maps them to predictions. Only the diffusion model is optimized; the pretrained TFM supplies the encoder, target extractor, and decoder.

We train TabSQRT on 100,000 real-world datasets selected from TabLib (Eggert et al., 2023) and evaluate transfer to unseen tables on TabArena (Erickson et al., 2025). The complete model improves on its LimiX teacher across all four reported metrics. Ablations show complementary benefits from representation learning and classification post-training: neither the denoising checkpoint alone nor CE-only fine-tuning of LimiX exceeds the teacher's mean accuracy. Our contributions are:

- **Self-query supervision for tabular ICL.** We use a frozen TFM's context to construct label-informed representation targets and transfer them through a conditional diffusion student with a frozen prediction interface.
- **Evidence on target selection.** Per-dataset diagnostics reveal opposing block-size trends for target decoding and diffusion-student accuracy, identifying a limitation of decoding accuracy as a criterion for constructing supervision.
- **Cross-dataset transfer.** TabSQRT improves over its teacher on all four reported TabArena metrics without dataset-specific fine-tuning, with ablations supporting the combined training procedure.

2 Related Work

Tabular prediction and tabular foundation models. Gradient-boosted decision trees remain strong dataset-specific predictors on many tabular benchmarks (Chen & Guestrin, 2016; Ke et al., 2017; Prokhorenkova et al., 2018). Neural architectures such as TabNet, NODE, SAINT, TabTransformer, and FT-Transformer provide alternative inductive biases but are also typically trained or tuned separately for each dataset (Arik & Pfister, 2021; Popov et al., 2020; Somepalli et al., 2021; Huang et al., 2020; Gorishniy et al., 2021). Tabular foundation models instead cast prediction as in-context inference over labeled support examples and unlabeled queries. TabPFN established this paradigm with synthetic pretraining (Hollmann et al., 2023), while TabICL and related systems extend it to larger or alternative regimes (Qu et al., 2025; Mueller et al., 2025). The source of pretraining tasks also varies: TabDPT learns directly from real-world tables (Ma et al., 2025), which are also the source of our representation-supervision corpus.

Privileged supervision and distillation. Can information available only during training help a student predict from less informative inputs? Self-query supervision poses this question for tabular models by giving the teacher access to query labels when constructing targets. Privileged learning and knowledge distillation provide the broader framework (Vapnik & Vashist, 2009; Hinton et al., 2015). Recent LLM studies similarly use additional training-time context to produce teacher supervision for students that do not receive that context (Zhao et al., 2026; Penaloza et al., 2026). Our targets are representations at the frozen decoder's input.

Diffusion models for tabular data. TabSQRT uses diffusion for cell-level prediction in a TFM's representation space, generating query label representations conditioned on feature and context-label tokens. TabSyn and TabDiff primarily model complete tables for data synthesis (Zhang et al., 2024; Shi et al., 2025). TabSyn also adapts its unconditional latent diffusion model to missing-value imputation through RePaint-style conditional sampling (Zhang et al., 2024; Lugmayr et al., 2022), while DiffPuter directly targets missing-data imputation (Zhang et al., 2025a). TabFORGE brings pretrained PFN representations into tabular generation and fine-tunes its diffusion Transformer and decoder for each new dataset (Jiang et al., 2026). This dataset-specific adaptation differs from our cross-dataset training and direct inference on unseen tables.

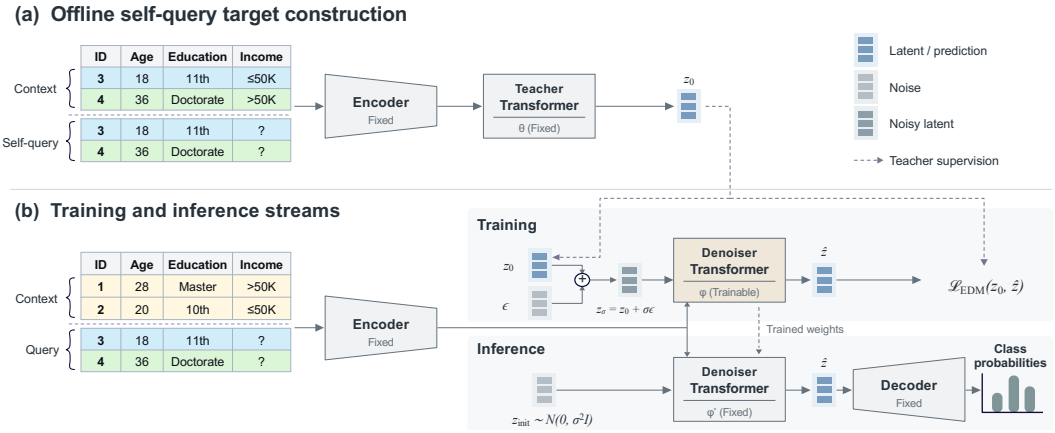


Figure 1: Overview of TabSQRT. (a) A frozen encoder and teacher Transformer produce offline self-query targets z_0 from labeled context–query pairs. (b) The frozen encoder provides the standard context-and-query condition for the trainable denoiser. During training, the denoiser maps a noisy target z_σ to $\hat{z}$ and minimizes the EDM loss against z_0; during inference, it maps a Gaussian initialization to a generated representation, which the frozen decoder converts to class probabilities.

3 METHOD

TabSQRT combines a frozen pretrained TFM, self-query representations used as privileged targets, and a conditional diffusion model that predicts those representations from standard in-context inputs.

3.1 PRETRAINED TFM AND REPRESENTATION EXTRACTION

For each classification task, let $\mathcal{C} = \{(x_i, y_i)\}_{i=1}^n$ denote a labeled context set. We write $X_\mathrm{c} = (x_i)_{i=1}^n$ and $y_\mathrm{c} = (y_i)_{i=1}^n$ for its features and labels, and let $X_\mathrm{q} = \{x_j^\mathrm{q}\}_{j=1}^m$ denote unlabeled query inputs. Standard in-context prediction takes $I^\mathrm{std} = [X_\mathrm{c}, y_\mathrm{c}, X_\mathrm{q}]$, with query examples excluded from the labeled context.

We use pretrained LimiX (Zhang et al., 2025b) as the TabSQRT backbone and distinguish an encoder E_θ, a Transformer T_θ, and a decoder g_θ. Their computation is

$$\begin{aligned} H^\mathrm{std} &= E_\theta(I^\mathrm{std}), \\ Z_\mathrm{q}^\mathrm{std} &= \mathrm{SelectQuery}\big(T_\theta(H^\mathrm{std})\big) \in \mathbb{R}^{m\times d}, \\ L_\mathrm{q}^\mathrm{std} &= g_\theta(Z_\mathrm{q}^\mathrm{std}) \in \mathbb{R}^{m\times|\mathcal{Y}|}. \end{aligned} \tag{1}$$

Here d is the representation dimension, $\mathcal{Y}$ is the label set, and prediction selects the largest logit in each row of $L_\mathrm{q}^\mathrm{std}$. The parameters θ of all three components remain frozen; ϕ denotes the trainable diffusion parameters introduced below.

3.2 SELF-QUERY TARGET CONSTRUCTION

We construct training targets by allowing the frozen TFM to access query labels. Given a labeled block $\mathcal{B} = \{(x_i, y_i)\}_{i=1}^k$, let $X_\mathcal{B} = (x_i)_{i=1}^k$ and $y_\mathcal{B} = (y_i)_{i=1}^k$. We use its labeled examples as context and present the same feature vectors again as queries:

$$I_\mathcal{B}^\mathrm{SQ} = [X_\mathcal{B}, y_\mathcal{B}, X_\mathcal{B}]. \tag{2}$$

We call this construction *self-querying*: each query has a corresponding labeled example in the teacher's context. The labels are supplied through the context, while the repeated query inputs occupy the usual unlabeled query positions.

The frozen encoder and Transformer process this input. We extract the query label token representations after the Transformer's final normalization, at the input interface of the frozen decoder. Here, T_θ includes this final normalization, and $\mathrm{SelectQuery}$ selects the label token at each query position:

$$H_{\mathcal{B}}^{\mathrm{SQ}} = E_\theta(I_{\mathcal{B}}^{\mathrm{SQ}}), \qquad Z_{\mathcal{B}}^{\mathrm{SQ}} = \mathrm{SelectQuery}\Big(T_\theta(H_{\mathcal{B}}^{\mathrm{SQ}})\Big) \in \mathbb{R}^{k\times d}. \tag{3}$$

Each row of $Z_{\mathcal{B}}^{\mathrm{SQ}}$ is the self-query representation for one example and depends on its features and the entire labeled block, including its own label. The decoder g_θ maps these representations directly to class logits.

Blockwise target construction. The size and composition of a self-query block affect the target assigned to each example. We study the effect of block size in Section 4.4.

We shuffle each training dataset and partition it into blocks containing at most K examples, where K denotes the self-query context size. Datasets with at most K examples are processed as a single block. Before diffusion training, we extract one self-query representation per example within its assigned block and cache it under the original sample index. These cached targets are reused for episodes drawn from the same table view. For each augmented table view with a different column order or label encoding, we extract and cache the corresponding self-query targets offline before training; details are given in Appendix A.2.

Pairing targets with standard conditions. During diffusion training, disjoint context and query sets are resampled from the entire training table for each episode, without restriction to the self-query blocks. The query targets in a single episode may therefore come from different blocks. We retrieve the cached representations by sample index and arrange them in query order as $Z_{\mathrm{q}}^{\mathrm{SQ}} = (z_{i_1}^{\mathrm{SQ}}, \ldots, z_{i_m}^{\mathrm{SQ}}) \in \mathbb{R}^{m\times d}$. Together with the standard encoder sequence defined above, this gives the training pair $(H^{\mathrm{std}}, Z_{\mathrm{q}}^{\mathrm{SQ}})$. Each target remains the representation extracted from its original labeled block, even as the student's context changes. K controls the labeled neighborhood used for target construction and does not constrain the student's episode size. Query labels are used only during training, for target construction and post-training classification supervision; they are not included in H^{std}.

3.3 Conditional Diffusion for Representation Transfer

Training pairs in the frozen TFM space. Given the training pairs constructed in Section 3.2, we set $s = H^{\mathrm{std}}$ and $z_0 = Z_{\mathrm{q}}^{\mathrm{SQ}}$, and learn the conditional distribution $p_\phi(z_0 \mid s)$ across datasets.

Denoiser architecture. The denoiser F_ϕ operates on a single token tensor formed from H^{std}, which contains feature tokens and a label token for each row. We replace only the query label placeholders with the preconditioned noisy target representations, while preserving all feature tokens and context label tokens as the in-context condition. Each query's noise level is encoded using a sinusoidal embedding of dimension d. The resulting tensor passes through a stack of Transformer layers that combine within-row feature attention with across-row sequence attention: each layer contains two feature-attention sublayers and one sequence-attention sublayer, each followed by an MLP, with the noise-level embedding added to the query label tokens before every sublayer. We read out the final query label tokens to form F_ϕ, which we combine with the noisy input to obtain a denoised estimate, detailed next. The frozen decoder maps this estimate to class logits.

EDM noising and denoising. We specify this noising and denoising scheme using the EDM continuous-noise formulation (Karras et al., 2022). During training, a noise level is sampled from a log-normal distribution,

$$\sigma = \exp(\mu + \tau\epsilon_\sigma), \qquad \epsilon_\sigma \sim \mathcal{N}(0,1),$$

where μ and τ are the mean and standard deviation of the log noise level. The target is perturbed as

$$z_\sigma = z_0 + \sigma\epsilon, \qquad \epsilon \sim \mathcal{N}(0, I).$$

Using the token construction described above, the preconditioned denoiser output is

$$D_\phi(z_\sigma, \sigma, s) = c_{\mathrm{skip}}(\sigma) z_\sigma + c_{\mathrm{out}}(\sigma) F_\phi(c_{\mathrm{in}}(\sigma) z_\sigma, c_{\mathrm{noise}}(\sigma), s).$$

Here, $c_{\mathrm{in}}(\sigma) z_\sigma$ supplies the query label tokens, and s supplies the preserved feature and context label tokens within the same tensor. The standard EDM coefficients are given in Appendix B.3.

Pretraining and post-training. We train the diffusion model in two stages. During pretraining, we optimize only the EDM-weighted denoising objective:

$$\mathcal{L}_{\text{pre}} = \mathcal{L}_{\text{MSE}} = \mathbb{E}_{z_0,s,\sigma,\epsilon}\Big[w(\sigma)\ \text{MSE}\big(D_\phi(z_0+\sigma\epsilon,\sigma,s),z_0\big)\Big].$$

The EDM weighting is

$$w(\sigma) = \frac{\sigma^2+\sigma_{\text{data}}^2}{(\sigma\sigma_{\text{data}})^2}.$$

During post-training, we initialize the denoiser from the pretrained checkpoint and add a classification loss through the frozen decoder:

$$\mathcal{L}_{\text{post}} = \mathcal{L}_{\text{MSE}} + \lambda\,\mathcal{L}_{\text{CE}},$$

where

$$\mathcal{L}_{\text{CE}} = \mathbb{E}_{z_0,s,y_{\text{q}},\sigma,\epsilon}\Big[\text{CE}\Big(g_\theta\big(D_\phi(z_0+\sigma\epsilon,\sigma,s)\big),y_{\text{q}}\Big)\Big].$$

Here λ weights the classification loss, which uses query labels to supervise the frozen decoder's predictions. Both stages update only the diffusion parameters ϕ. During post-training, gradients propagate through the frozen decoder to the denoiser, while the MSE term continues to supervise target reconstruction.

Sampling and prediction. At inference, we sample $\hat{z} \sim p_\phi(\cdot \mid H^{\text{std}})$ with the EDM Euler solver described in Appendix B.3, and obtain class logits as $g_\theta(\hat{z})$.

4 Experiments

4.1 Experimental Setup

Training data Our training data are selected from TabLib (Eggert et al., 2023) and cover 100,000 real-world datasets, totaling 580M rows and 14B cells. The datasets span domains including finance, education, psychology, biology, and medicine. Unless otherwise specified, we use the full training set for self-query target construction when it contains at most 10,000 examples, and partition larger datasets into blocks of at most $K_{\max} = 10{,}000$ examples. Dataset selection and deduplication are detailed in Appendix A.1.

Evaluation data For evaluation, we use TabArena (Erickson et al., 2025), a public benchmark comprising 38 classification datasets. We remove overlapping training tables using the name, metadata, and content-fingerprint checks described in Appendix A.1. We evaluate each method on five random train–test splits and report the mean and standard deviation of the resulting benchmark scores.

Backbone and model configuration. We use a pretrained LimiX model as the teacher, comprising a feature encoder, a 12-layer Transformer with 6 attention heads and embedding dimension 192, and a two-layer MLP decoder. All teacher components remain frozen. The diffusion denoiser follows the teacher's Transformer architecture, incorporates noise-level conditioning, and is randomly initialized. Only the denoiser is trained, accounting for nearly all of the model's approximately 16M parameters. The teacher Transformer is used only for offline target extraction. Unless otherwise specified, inference uses two denoiser evaluations. Training settings are given in Appendix A.2.

Baselines We compare TabSQRT with leading baselines under the shared TabArena evaluation protocol. Dataset-specific baselines include tree ensembles—XGBoost, LightGBM, and CatBoost (Chen & Guestrin, 2016; Ke et al., 2017; Prokhorenkova et al., 2018)—and neural methods TabM, TabR, FT-Transformer, and ResNet (Gorishniy et al., 2025; 2024; 2021). These methods are tuned separately for each dataset and split using their recommended search spaces; we select the configuration with the strongest validation performance for testing. We further compare with pretrained TFMs—LimiX, TabPFN-v3, Real-TabPFN-v2.5, TabICL-v2, and TabDPT (Zhang et al., 2025b; Grinsztajn et al., 2026; 2025; Qu et al., 2026; Ma et al., 2025)—and diffusion-based tabular methods TabSyn, TabDiff, and DiffPuter (Zhang et al., 2024; Shi et al., 2025; Zhang et al., 2025a). For pretrained methods, we use the authors' released inference implementations and do not apply dataset-specific fine-tuning. All methods are evaluated on the same benchmark splits and summarized by Accuracy, AUC, F1-macro, and Accuracy Rank.

Table 1: Results on TabArena. Accuracy, AUC, and F1-macro are reported as mean ± standard deviation where available. Higher is better for these three metrics; lower is better for Accuracy Rank. Best and second-best means are highlighted in bold and with underlining, respectively. Missing entries ("–") indicate unavailable results.

Family	Model	Accuracy ↑	AUC ↑	F1-macro ↑	Acc. rank ↓
Diffusion	DiffPuter	0.8411 ± 0.0037	0.7564 ± 0.0048	0.6314 ± 0.0094	11.61
	TabSyn	0.7738 ± 0.0064	0.6625 ± 0.0083	0.5958 ± 0.0076	14.53
	TabDiff	0.7549 ± 0.0035	–	0.6075 ± 0.0083	15.24
Deep Learning	TabM	0.8682 ± 0.0010	0.8369 ± 0.0008	0.6830 ± 0.0030	7.74
	TabR	0.8648 ± 0.0017	0.8352 ± 0.0015	0.6917 ± 0.0029	8.73
	FTT	0.8577 ± 0.0017	0.8255 ± 0.0018	0.6789 ± 0.0026	10.14
	ResNet	0.8479 ± 0.0009	0.7667 ± 0.0009	0.6228 ± 0.0005	11.80
Tree	XGB	0.8709 ± 0.0014	0.8542 ± 0.0026	0.6996 ± 0.0010	8.76
	CatBoost	0.8698 ± 0.0015	0.8273 ± 0.0016	0.6869 ± 0.0019	8.58
	LightGBM	0.8692 ± 0.0014	0.8492 ± 0.0014	0.6975 ± 0.0021	8.28
TFM	TabPFN-v3	**0.8865** ± 0.0006	<u>0.8677</u> ± 0.0013	0.7284 ± 0.0014	**3.95**
	TabICL-v2	<u>0.8861</u> ± 0.0013	0.8672 ± 0.0017	**0.7321** ± 0.0018	4.70
	Real-TabPFN-v2.5	0.8859 ± 0.0010	<u>0.8677</u> ± 0.0013	0.7287 ± 0.0008	4.62
	LimiX	0.8830 ± 0.0009	0.8640 ± 0.0013	0.7184 ± 0.0034	5.62
	TabDPT	0.8799 ± 0.0010	0.8558 ± 0.0007	0.6996 ± 0.0027	7.12
TabSQRT		**0.8865** ± 0.0010	**0.8686** ± 0.0009	<u>0.7293</u> ± 0.0019	<u>4.58</u>

4.2 Main performance

TabSQRT ties for the highest mean accuracy and leads on AUC in Table 1; it ranks second on F1-macro and mean accuracy rank. Against its LimiX teacher, its mean accuracy rank improves from 5.62 to 4.58. This comparison uses the full model with CE post-training. Additional evaluation on CC18 is reported in Appendix D.1.

The tree and neural baselines trail the leading TFMs in mean accuracy, and the evaluated diffusion baselines are further behind.

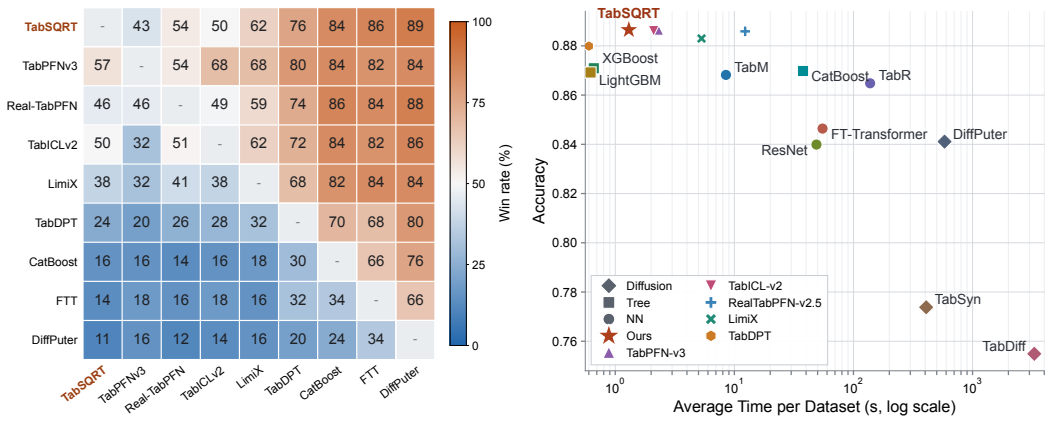
Pairwise win rates. Figure 2(a) shows that TabSQRT wins on 62% of datasets against LimiX. Against TabPFN-v3, TabSQRT wins on 43% and TabPFN-v3 on 57%, despite their equal mean accuracy. TabSQRT's wins are more common on datasets with smaller training sets. TabPFN-v3 tends to retain an advantage on larger training sets and on datasets where it already achieves high accuracy. Comparisons with TabICL-v2 and Real-TabPFN-v2.5 are roughly balanced.

Model size and inference time. TabSQRT uses approximately 16M inference parameters, compared with 28M for TabICL-v2 and 53M for TabPFN-v3. In Figure 2(b), it matches TabPFN-v3's mean accuracy with a lower average runtime. It also runs faster than the evaluated diffusion baselines.

4.3 Ablation Study

We examine the training choices in TabSQRT using the variants in Table 2, and compare student performance using self-query targets from two pretrained TFMs in Table 3.

Training ablations. Table 2 compares four variants with TabSQRT. FT-LimiX fine-tunes pretrained LimiX, while Scratch-LimiX trains it from random initialization. Both optimize cross-entropy alone, so each has a single accuracy entry. SQ-Transformer uses the LimiX Transformer architecture to predict the same self-query targets directly, without diffusion. CQ-TabSQRT retains



(a) Pairwise win rates (b) Accuracy–runtime trade-off

Figure 2: Main-performance comparisons on TabArena. (a) Pairwise win rates among selected top methods; rows indicate the evaluated method and columns its opponent. (b) Mean accuracy versus average inference time per dataset, with the horizontal axis shown on a logarithmic scale.

Table 2: Ablation study on TabArena. Accuracy is reported for each training variant. CE Yes adds classification supervision to the representation objective; CE No uses the representation objective alone. †FT-LimiX and Scratch-LimiX use classification supervision alone and are shown in the CE Yes row for compactness.

	CE-only baselines		Representation predictors		
CE post-training	**FT-LimiX†**	**Scratch-LimiX†**	**CQ-TabSQRT**	**SQ-Transformer**	**TabSQRT**
No	—	—	0.8590	0.8297	0.8826
Yes	0.8829	0.8749	0.8663	0.8442	**0.8865**

the diffusion predictor but replaces self-query targets with representations extracted while query examples are absent from the teacher's labeled context. These comparisons examine the effects of initialization, target construction, and the representation-learning objective.

Self-query targets outperform cross-query targets with and without CE post-training. CE post-training raises TabSQRT's accuracy by 0.39 percentage points, taking it from slightly below standard LimiX to above it. Fine-tuning LimiX with CE alone does not reproduce this gain. MSE encourages reconstruction of the self-query targets, while CE post-training aligns the generated representations with the frozen decoder's decision function. Errors that are small in representation space can still affect class predictions; the classification objective directly penalizes these errors.

TabSQRT outperforms direct regression on the same self-query targets both before and after CE post-training. This comparison supports the use of noise-conditioned denoising over direct target regression.

Table 3: Student performance using self-query targets from different TFM backbones. The student column reports accuracy after training on the corresponding backbone's representations. Δ denotes the change from standard backbone inference.

Backbone	**Mean accuracy**		
	Standard	**Student**	Δ
LimiX	0.7383	0.7444	+0.0061
TabPFN-v2	0.7394	0.7447	+0.0053

Self-query across backbones. Table 3 compares standard backbone predictions with students trained on self-query representations from LimiX and TabPFN-v2 (Hollmann et al., 2025). For each backbone, we train a separate student on each of the 20 datasets used in Section 4.4—ten large and ten small—and average accuracy with equal weight across datasets. Students improve mean accuracy by roughly half a percentage point for both backbones. The gain with TabPFN-v2 shows that self-query representation supervision can also benefit student learning with a teacher other than LimiX.

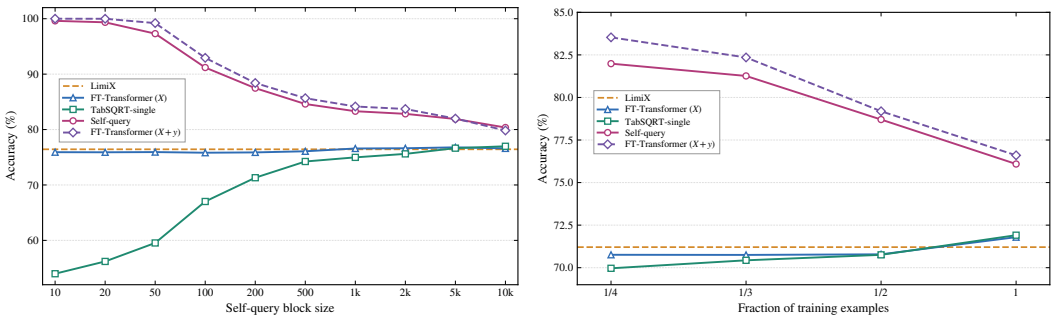
4.4 Context Size

We vary the self-query block size and measure target decoding accuracy and student test accuracy. We train a separate TabSQRT student on each dataset, denoted as TabSQRT-single. The FT-Transformer student uses the same architecture as the FTT classification baseline in Table 1, but is trained to predict self-query representations; the SQ-Transformer in Table 2 instead uses the LimiX Transformer architecture. We also evaluate a diagnostic FT-Transformer that receives the true query labels alongside the features, regresses the same cached self-query targets, and obtains predictions through the frozen decoder. This label-informed control is excluded from standard prediction comparisons. We use ten large datasets with more than 10,000 training examples and ten small datasets with 500–8,000 training examples, averaging accuracy equally across datasets. Within each group, the datasets, student training data, and inference-time context protocol stay the same; only the self-query block size changes.

Large datasets. Figure 3(a) shows that the smallest blocks produce nearly perfectly decoded targets but poor TabSQRT-single predictions. As block size increases, target decoding accuracy falls while TabSQRT-single accuracy rises, with diminishing gains near the largest blocks. FT-Transformer (X) accuracy varies much less across block sizes. At the largest block size, TabSQRT-single and FT-Transformer (X) both perform close to standard LimiX.

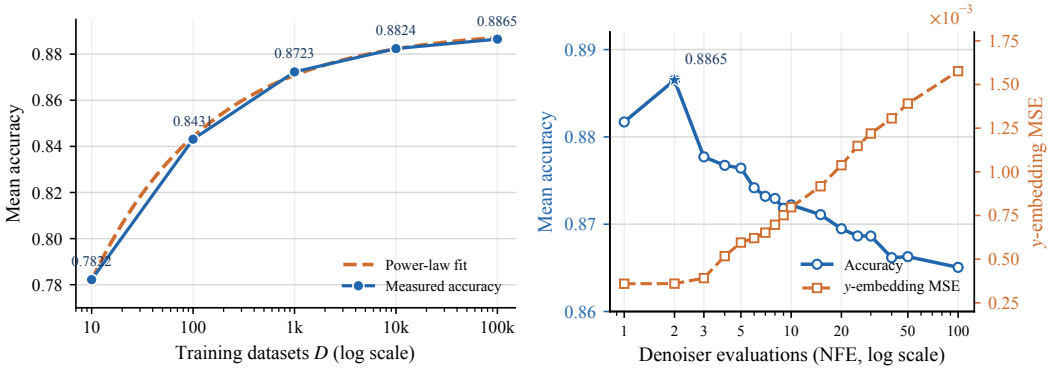
Small datasets. For small datasets, we construct targets using blocks containing approximately one quarter, one third, one half, or all training examples (Figure 3(b)). TabSQRT-single mean accuracy rises by about 1.9 percentage points between the smallest and largest settings. TabSQRT-single and FT-Transformer (X) each reach their highest average accuracy with full-table targets; the gain is smaller for FT-Transformer (X).

Choice of context size. The label-informed FT-Transformer follows the target-decoding trend, while TabSQRT-single receiving standard inputs benefits from larger blocks. One interpretation is that small-block targets rely more on the example's own label, while larger-block targets capture more contextual structure that the student can recover from standard inputs.



(a) Large tables (b) Small tables

Figure 3: Effect of self-query context size on target decoding and student test accuracy. (a) Ten large datasets with block sizes from 10 to 10,000. (b) Ten small datasets with block sizes set as fractions of the training set. Self-query denotes target decoding accuracy. TabSQRT-single and FT-Transformer (X) use standard inputs; FT-Transformer ($X + y$) additionally receives true query labels as a diagnostic control. The horizontal dashed line shows standard LimiX inference. Results are averaged equally across datasets.



(a) Training data scale (b) Sampling behavior

Figure 4: Effects of training-data scale and denoiser evaluations. (a) Mean TabArena accuracy as the number of training datasets increases; the dashed curve is the least-squares fit of Equation 4. (b) Mean TabArena accuracy and MSE between generated and self-query y-embeddings as the number of denoiser evaluations (NFE) varies.

We cap self-query blocks at 10,000 examples to limit memory use and extraction time, using full-table targets below the cap (Section 4.1). This choice follows the student-accuracy curves; choosing the block size by target decoding accuracy would favor the smallest blocks.

4.5 Training Data Scale

We train TabSQRT on corpora containing $D \in \{10, 100, 1{,}000, 10{,}000, 100{,}000\}$ datasets, where D counts tables rather than rows. Accuracy rises with corpus size, but the gains are concentrated at the smaller scales: the first tenfold increase adds about six percentage points, whereas the last adds about 0.4 (Figure 4(a)).

We summarize the curve with a saturating power law, following the data-scaling form of Rosenfeld et al. (2020) and related scaling analyses in TabDPT (Ma et al., 2025):

$$\widehat{\mathrm{Acc}}(D) = a_\infty - bD^{-\alpha}, \tag{4}$$

where a_∞ is the fitted asymptote and $b, \alpha > 0$ control the gap and its decay. Fitting the five measurements gives $a_\infty \approx 0.8909$.

4.6 Sampling Behavior

We select two denoiser evaluations (NFE=2) using an independent validation set. Figure 4(b) reports the corresponding NFE sweep on TabArena. Accuracy also peaks at NFE=2, which uses three noise points and two denoiser calls, then declines as sampling continues. Reconstruction MSE rises beyond this point as well. Additional sampling thus increases inference cost while reducing prediction quality.

A possible explanation is exposure bias (Ning et al., 2024): training inputs are noisy versions of clean targets, whereas intermediate sampling states come from the model itself. Appendix D.3 examines noise controls and the sampling update.

5 Conclusion

The most accurately decoded self-query targets do not give the best student predictions. In our experiments, selecting the target-construction context by teacher decoding accuracy would favor small blocks and sacrifice downstream accuracy. We therefore treat target construction as a choice to be evaluated through student learning. TabSQRT makes that choice actionable: a frozen teacher can supply different supervision from the same labeled table simply by changing its context.

AI USE STATEMENT

We used AI tools for manuscript drafting and editing, literature searches, figure preparation, plotting code, and LaTeX formatting, as well as feedback on experimental design and result interpretation. Verification included checking references against the original sources and checking reported values and figures against experimental results. The authors reviewed the AI-assisted material and take responsibility for the final manuscript and its scientific claims.

REFERENCES

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

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*, pp. 785–794, 2016.

Jang Hyun Cho and Bharath Hariharan. On the efficacy of knowledge distillation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 4794–4802, 2019.

Gus Eggert, Kevin Huo, Mike Biven, and Justin Waugh. TabLib: A dataset of 627m tables with context. *arXiv preprint arXiv:2310.07875*, 2023.

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*, volume 38, 2025.

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, pp. 18932–18943, 2021.

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

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

Léo Grinsztajn, Klemens Flöge, Oscar Key, Felix Birkel, Brendan Roof, Philipp Jund, Benjamin Jäger, Adrian Hayler, Dominik Safaric, Simone Alessi, et al. TabPFN-2.5: Advancing the state of the art in tabular foundation models, 2025. URL https://arxiv.org/abs/2511.08667.

Léo Grinsztajn, Klemens Flöge, Oscar Key, Felix Birkel, Philipp Jund, Brendan Roof, Mihir Manium, Shi Bin Hoo, Magnus Bühler, Anurag Garg, et al. TabPFN-3: Technical report, 2026. URL https://arxiv.org/abs/2605.13986.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*, 2015.

Noah Hollmann, Samuel Müller, Katharina Eggensperger, and Frank Hutter. TabPFN: A transformer that solves small tabular classification problems in a second. In *International Conference on Learning Representations*, 2023.

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. doi: 10.1038/s41586-024-08328-6.

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

Xiangjian Jiang, Mingxuan Liu, Nikola Simidjievski, Tassilo Klein, and Mateja Jamnik. Tabular foundation model for generative modelling, 2026. URL https://arxiv.org/abs/2605.09424.

Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design space of diffusion-based generative models. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

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*, volume 30, 2017.

David Lopez-Paz, Léon Bottou, Bernhard Schölkopf, and Vladimir Vapnik. Unifying distillation and privileged information. In *International Conference on Learning Representations*, 2016.

Andreas Lugmayr, Martin Danelljan, Andres Romero, Fisher Yu, Radu Timofte, and Luc Van Gool. RePaint: Inpainting using denoising diffusion probabilistic models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11461–11471, 2022.

Junwei Ma, Valentin Thomas, Rasa Hosseinzadeh, Alex Labach, Jesse C. Cresswell, Keyvan Golestan, Guangwei Yu, Anthony L. Caterini, and Maks Volkovs. TabDPT: Scaling tabular foundation models on real data. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-5748.

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

Mang Ning, Mingxiao Li, Jianlin Su, Albert Ali Salah, and Itir Onal Ertugrul. Elucidating the exposure bias in diffusion models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=xEJMoj1SpX.

Emiliano Penaloza, Dheeraj Vattikonda, Nicolas Gontier, Alexandre Lacoste, Laurent Charlin, and Massimo Caccia. Privileged information distillation for language models. *arXiv preprint arXiv:2602.04942*, 2026.

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

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*, volume 31, 2018.

Jingang Qu, David Holzmüller, Gaël Varoquaux, and Marine Le Morvan. TabICL: A tabular foundation model for in-context learning on large data. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 50817–50847. PMLR, 2025.

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.

Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. FitNets: Hints for thin deep nets. In *International Conference on Learning Representations*, 2015.

Jonathan S. Rosenfeld, Amir Rosenfeld, Yonatan Belinkov, and Nir Shavit. A constructive prediction of the generalization error across scales. In *International Conference on Learning Representations*, 2020. URL https://arxiv.org/abs/1909.12673.

Juntong Shi, Minkai Xu, Harper Hua, Hengrui Zhang, Stefano Ermon, and Jure Leskovec. TabDiff: A mixed-type diffusion model for tabular data generation. In *International Conference on Learning Representations*, 2025.

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

Vladimir Vapnik and Akshay Vashist. A new learning paradigm: Learning using privileged information. *Neural Networks*, 22(5–6):544–557, 2009.

Hengrui Zhang, Jiani Zhang, Balasubramaniam Srinivasan, Zhengyuan Shen, Xiao Qin, Christos Faloutsos, Huzefa Rangwala, and George Karypis. Mixed-type tabular data synthesis with score-based diffusion in latent space. In *International Conference on Learning Representations*, 2024.

Hengrui Zhang, Liancheng Fang, Qitian Wu, and Philip S. Yu. DiffPuter: Empowering diffusion models for missing data imputation. In *International Conference on Learning Representations*, 2025a.

Xingxuan Zhang, Gang Ren, Han Yu, Hao Yuan, Hui Wang, Jiansheng Li, Jiayun Wu, Lang Mo, Li Mao, Mingchao Hao, et al. LimiX: Unleashing structured-data modeling capability for generalist intelligence. *arXiv preprint arXiv:2509.03505*, 2025b.

Siyan Zhao, Zhihui Xie, Mengchen Liu, Jing Huang, Guan Pang, Feiyu Chen, and Aditya Grover. Self-distilled reasoner: On-policy self-distillation for large language models. *arXiv preprint arXiv:2601.18734*, 2026.

A Experimental Setup and Reproducibility

A.1 Training Data Curation and Deduplication

We curate the TabLib training corpus before target construction. We discard tables with fewer than 100 examples and subsample tables with more than 50,000 examples to 50,000 rows. Retained tables have at most 500 feature columns and between two and ten classes. Tables containing date or time fields, free-form comments, diagnostic narratives, or similar unstructured text fields are excluded. We retain one version of each dataset after deduplication.

For tables with 11–15 classes, we construct a smaller classification task. When the class distribution is approximately balanced, we randomly select between two and ten classes and retain their examples. For strongly imbalanced tables, we first discard the rarest classes and then retain at most ten classes. These operations are applied only to construct the training corpus; evaluation tables are not altered.

Deduplication. We remove duplicates in three stages. First, we normalize dataset and source names and remove records that refer to the same named dataset. Second, we use metadata—the number of examples and features, normalized column schema, number of classes, and class-frequency profile—to identify candidate duplicates. Matching metadata alone is not sufficient for removal: distinct tables can share the same dimensions and label structure.

For each candidate pair, we compute a content fingerprint. We canonicalize missing values, value representations, and column order; hash each normalized row; sort the row hashes to make the comparison invariant to row order; and hash the resulting sequence into a dataset-level digest. A pair is removed only when these content fingerprints match exactly. We use the same name, metadata, and content-fingerprint procedure against the TabArena evaluation tables, removing any overlapping training table before training.

A.2 Training Settings

We train TabSQRT's diffusion denoiser in two stages. During MSE pretraining, the learning rate is 8×10^{-4}. During CE post-training, it is 5×10^{-5}, and we set the classification-loss weight to $\lambda = 0.08$. We select λ through preliminary experiments. MSE pretraining and CE post-training each last 100 epochs, with a 10-epoch warmup followed by cosine learning-rate decay in each stage (Loshchilov & Hutter, 2017).

Optimization and checkpoint selection. Both stages use AdamW (Loshchilov & Hutter, 2019) with $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$, and decoupled weight decay 10^{-3}. At the transition to CE post-training, we load the pretrained denoiser weights and reinitialize the optimizer moments. We clip the gradient ℓ_2 norm to 1 and use no gradient accumulation. An exponential moving average of the model parameters is updated every 10 optimization steps with decay 0.995. The diffusion loss is computed in FP32; autocast is enabled when materializing frozen encoder outputs. For each dataset, we hold out 10% of its training rows for validation. Checkpoints are selected separately in each stage by the lowest validation loss: EDM-weighted MSE during pretraining and $\mathcal{L}_{\mathrm{MSE}} + 0.08\mathcal{L}_{\mathrm{CE}}$ during post-training.

Epoch definition. Each epoch processes 10,000 tables, and ten consecutive epochs cover the full corpus of 100,000 datasets once. The number of episodes contributed by each table depends on its context-size grid and query batches, as described below.

Episode construction. The DataLoader batch size is one episode per GPU, so with gradient accumulation set to one, each DDP optimization step aggregates one episode from each GPU. Each episode contains a labeled context and a query batch of at most 1,024 rows. Context and query rows are disjoint and are sampled from the entire training table.

Let R denote the number of rows in the training table. For $R \leq 1{,}000$, context sizes start at 30 and increase in increments of approximately $0.08R$, stopping below $0.85R$. For $1{,}000 < R \leq 10{,}000$, sizes start at approximately $\min(0.4R, 1{,}000)$ and increase in increments of approximately $0.1R$, stopping strictly below $\min(\lfloor 0.85R \rfloor, 3{,}000)$. For $R > 10{,}000$, the context-size grid is

$\{1{,}000, 1{,}200, 1{,}600, 2{,}000, 2{,}400, 2{,}800, 3{,}000\}$. For each column ordering, we draw five context subsets at every grid size when $R \leq 1{,}000$, and four otherwise. These correspond to five or four sampling seeds, each traversing the entire grid. For tables with more than 10,000 rows, we retain at most $6 \times 1{,}024$ query rows for each sampled context and split them into query batches.

For wide tables with $p > 50$ feature columns, we also impose a budget of 200,000 cells per encoder forward pass. Defining $B = \lfloor 200{,}000/p \rfloor$, the context size is capped at $\min(3{,}000, \lfloor 2B/3 \rfloor)$, and the query batch size is at most $\min(1{,}024, B - n)$, where n is the context size. Thus $p(n + m) \leq 200{,}000$ for a query batch of m rows. Tables with at most 50 feature columns do not use this additional cell-budget cap.

Diffusion configuration. The log-normal training noise distribution uses $\mu = 0$ and $\tau = 1.2$, with $\sigma_{\text{data}} = 0.1$ for EDM preconditioning and loss weighting. During post-training, CE is computed by passing a single denoising output $D_\phi(z_0 + \sigma\epsilon, \sigma, s)$ through the frozen decoder, with the representation rescaling described in Appendix B.2. The inference schedule uses $\sigma_{\min} = 0.002$, $\sigma_{\max} = 80$, and $\rho = 7$.

Augmentation. We augment tables by permuting feature-column order and shifting class-label encodings. These transformations change the frozen teacher's token sequence or label embeddings, so each augmented view requires its own self-query targets. We prepare and cache both the augmented tables and their corresponding self-query representations offline. During training, episodes are constructed from these cached views and targets.

A.3 Main Evaluation Protocol

Data splits and reported variability. We evaluate each method on five train–test splits: the original split and four random splits generated with seeds 42, 43, 44, and 45. The training-to-test ratio is $2:1$; random sampling is not stratified by class. For each split, we average dataset-level scores with equal weight for each dataset. We then report the mean and sample standard deviation ($\text{ddof} = 1$) of the five benchmark averages. The reported standard deviation therefore measures variation across splits after benchmark aggregation, rather than variation across datasets or an average of dataset-level standard deviations.

Metrics and ranking. Accuracy is the fraction of correctly classified test examples, and F1 is macro-averaged over classes. Binary AUC uses the positive-class probability; multiclass AUC uses one-vs-rest comparisons with macro averaging. For Accuracy Rank, methods are ranked by accuracy separately on each dataset and split, and these ranks are then averaged across datasets and splits.

Model configurations. The TFM baselines are run with their default configurations. Preprocessing follows each model's default implementation. Context caching is enabled for TabSQRT.

Timing measurements. The runtime measurements for Figure 2(b) are collected on a PRO 5000 GPU during the same five split-based evaluations used to compute predictive performance. Timing starts at the prediction call and ends after metric computation, excluding CSV reading, preprocessing, and model-weight loading. We report the arithmetic mean of the measured durations across datasets and splits. TabSQRT uses context caching in these measurements.

A.4 Classification with Diffusion-Based Imputation

We use TabSyn, TabDiff, and DiffPuter to predict labels by treating the target column as missing at inference. The training table includes both features and labels. For each test example, only the observed features are provided to the imputation procedure; the true label is withheld and used solely for evaluation. The imputed target is then interpreted as a class prediction, without training an external classifier on synthetic tables.

TabSyn. TabSyn learns a VAE representation of the table and a diffusion model in this latent space (Zhang et al., 2024). The label is treated as a categorical column alongside the feature columns. To predict it, we mask its latent token and perform conditional latent completion using the observed

feature tokens, following the RePaint-style imputation procedure (Lugmayr et al., 2022). The VAE decoder maps the completed label token to class logits. Our evaluation applies softmax to these logits to obtain class probabilities; their argmax gives the predicted class. These outputs support Accuracy, F1-macro, and AUC.

TabDiff. TabDiff models numerical and categorical columns, including the target, through mixed-type diffusion (Shi et al., 2025). At prediction time, the target column is masked and conditionally sampled with the observed features retained as conditioning information. The generated target is decoded to a class label. Our evaluation pipeline saves only the final hard labels, which suffice for Accuracy and F1-macro. It does not retain class probabilities or continuous prediction scores, so we leave AUC unreported. This is a limitation of the saved evaluation outputs, not an intrinsic inability of TabDiff to support AUC evaluation.

DiffPuter. DiffPuter combines diffusion-based density estimation with iterative missing-value imputation (Zhang et al., 2025a). We mark the target column as missing and the feature columns as observed. Conditional diffusion sampling restores the missing target while preserving the observed entries, and the restored categorical value is mapped back to its class label. Probability-based evaluation additionally requires class scores or a probability estimate from repeated categorical imputations; a final hard label alone does not retain this information.

B Architecture and Diffusion Details

B.1 Denoising Network Architecture

The denoiser follows the axis-wise Transformer design of the first LimiX technical report (Zhang et al., 2025b), but has independently initialized, trainable parameters. It uses 12 blocks, embedding dimension $d = 192$, and six attention heads, matching the LimiX-16M backbone described in Section 4.1.

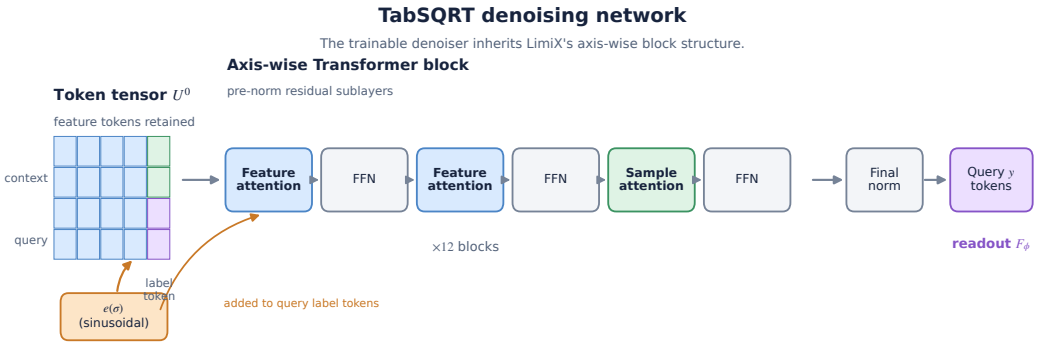


Figure 5: TabSQRT denoising network. The input retains standard encoder tokens, replacing only query label tokens with preconditioned noisy self-query representations. Each of 12 LimiX-style blocks applies two feature-attention passes and one sample-attention pass, interleaved with feed-forward networks. A sinusoidal noise embedding is added to query label tokens at every sublayer; final normalized query label tokens provide the denoiser output F_ϕ.

Let $r = n + m$ be the total number of context and query rows and let f be the number of feature tokens per row. The encoder produces a token tensor $H^{\text{std}} \in \mathbb{R}^{r\times(f+1)\times d}$, where the final token in each row is its label token. The denoiser input U^0 retains the feature tokens for all rows and the label tokens for context rows. At query label-token positions, we replace the encoder placeholders by $c_{\text{in}}(\sigma)z_\sigma$. Thus, the noisy target is the only input that replaces teacher encoder tokens; all remaining tokens provide the standard in-context condition.

Each block contains two feature-level attention passes and one sample-level attention pass, each followed by a two-layer GELU feed-forward network. Feature attention operates across the $f+1$ tokens within every row, whereas sample attention operates across rows for a shared token position and allows query rows to use the labeled context. All sublayers use pre-normalization and residual connections. Writing

$$\mathcal{A}_a(U) = U + \mathrm{MHA}_a(\mathrm{LN}(U)), \qquad \mathcal{M}(U) = U + \mathrm{FFN}(\mathrm{LN}(U)),$$

where $a \in \{\mathrm{F}, \mathrm{S}\}$ denotes the feature or sample axis, the ℓ-th block has the form

$$U^{\ell+1} = \mathcal{M}_{\ell,3}\big(\mathcal{A}_{\ell,\mathrm{S}}\big(\mathcal{M}_{\ell,2}\big(\mathcal{A}_{\ell,\mathrm{F},2}\big(\mathcal{M}_{\ell,1}\big(\mathcal{A}_{\ell,\mathrm{F},1}(U^{\ell})\big)\big)\big)\big)\big).$$

For each noise level σ, we form a sinusoidal embedding $e(\sigma) \in \mathbb{R}^d$ and add it to every query label token before each attention or feed-forward sublayer. After the final block and output normalization, we read out the query label tokens to obtain $F_\phi(c_{\mathrm{in}}(\sigma) z_\sigma, c_{\mathrm{noise}}(\sigma), s)$. The EDM skip and output terms in Section 3.3 then convert this prediction into $D_\phi(z_\sigma, \sigma, s)$; the frozen LimiX decoder maps the final denoised query representations to class logits.

B.2 Representation Normalization and Scaling

Extraction and normalization. The teacher applies a final LayerNorm (Ba et al., 2016) without learned affine parameters to each token's 192-dimensional vector, followed by elementwise clipping to $[-15, 15]$. We cache the query label-token representations after these operations.

Clipping and fixed rescaling. For non-affine LayerNorm using the population variance and a non-negative numerical stabilizer, each normalized coordinate has absolute value at most $\sqrt{d-1}$. With $d = 192$, this bound is approximately 13.82, below the clipping threshold of 15. Thus clipping is inactive for finite, exactly normalized vectors in this configuration, consistent with the measurements. A fixed divisor applied to the raw output would rescale all channels equally and retain differences in scale across tokens. LayerNorm instead centers and rescales each token individually. The distribution analysis characterizes this distinction; it does not measure the downstream accuracy of alternative normalization schemes.

Interface with the diffusion model and decoder. The cached representations are divided by ten when loaded for diffusion training, reducing their pooled standard deviation to 0.1 and matching $\sigma_{\mathrm{data}} = 0.1$. Before applying the frozen decoder, generated representations are multiplied by ten. Explicitly, if $Z_{\mathrm{q}}^{\mathrm{SQ}}$ denotes the cached teacher representation, the implementation uses

$$\widetilde{Z}_{\mathrm{q}}^{\mathrm{SQ}} = Z_{\mathrm{q}}^{\mathrm{SQ}}/10, \qquad \widehat{L}_{\mathrm{q}} = g_\theta(10\widehat{\widetilde{Z}}_{\mathrm{q}}).$$

Here $\widehat{\widetilde{Z}}_{\mathrm{q}}$ is the denoiser output in diffusion coordinates. The same inverse scaling is used when computing classification supervision through the decoder. The main-text equations suppress this fixed interface conversion. Unlike clipping, this rescaling is invertible and does not discard representation information.

B.3 EDM Details

Following EDM (Karras et al., 2022), we use the preconditioning coefficients

$$c_{\mathrm{skip}}(\sigma) = \frac{\sigma_{\mathrm{data}}^2}{\sigma^2 + \sigma_{\mathrm{data}}^2}, \qquad c_{\mathrm{out}}(\sigma) = \frac{\sigma\sigma_{\mathrm{data}}}{\sqrt{\sigma^2 + \sigma_{\mathrm{data}}^2}},$$
$$c_{\mathrm{in}}(\sigma) = \frac{1}{\sqrt{\sigma^2 + \sigma_{\mathrm{data}}^2}}, \quad c_{\mathrm{noise}}(\sigma) = \frac{1}{4}\log\sigma,$$

where σ_{data} sets the characteristic scale of the clean target representations.

For sampling, N denotes the number of denoiser evaluations (one per Euler step) and ρ controls the curvature of the schedule between endpoints $\sigma_{\max}$ and $\sigma_{\min}$. For $N \geq 2$, we set

$$t_i = \left[\sigma_{\max}^{1/\rho} + \frac{i}{N-1}(\sigma_{\min}^{1/\rho} - \sigma_{\max}^{1/\rho})\right]^\rho, \quad i = 0, \ldots, N-1, \qquad t_N = 0.$$

For $N = 1$, the schedule consists of $t_0 = \sigma_{\max}$ and $t_1 = 0$. We denote sampling states by u_i to distinguish them from the clean target z_0. Starting from $u_0 \sim \mathcal{N}(0, t_0^2 I)$, the Euler updates for $i = 0, \ldots, N-1$ are

$$d_i = \frac{u_i - D_\phi(u_i, t_i, s)}{t_i}, \qquad u_{i+1} = u_i + (t_{i+1} - t_i) d_i.$$

The final sample is $\hat{z} = u_N$.

C Analysis of Self-Query Representations

C.1 Additional Context-Size Trends

We supplement the accuracy curves in Figure 3 with two diagnostics. First, an FT-Transformer receives only the query labels and is trained to regress the cached self-query representations. Its predictions isolate how much of each target can be recovered from the label alone, without access to the query features. Second, we report the mean-squared error between the predicted and cached self-query representations across block sizes. This representation-space measure complements decoder accuracy: two predictions can have similar MSE yet lead to different class decisions, while a change in MSE need not imply the same change in downstream accuracy.

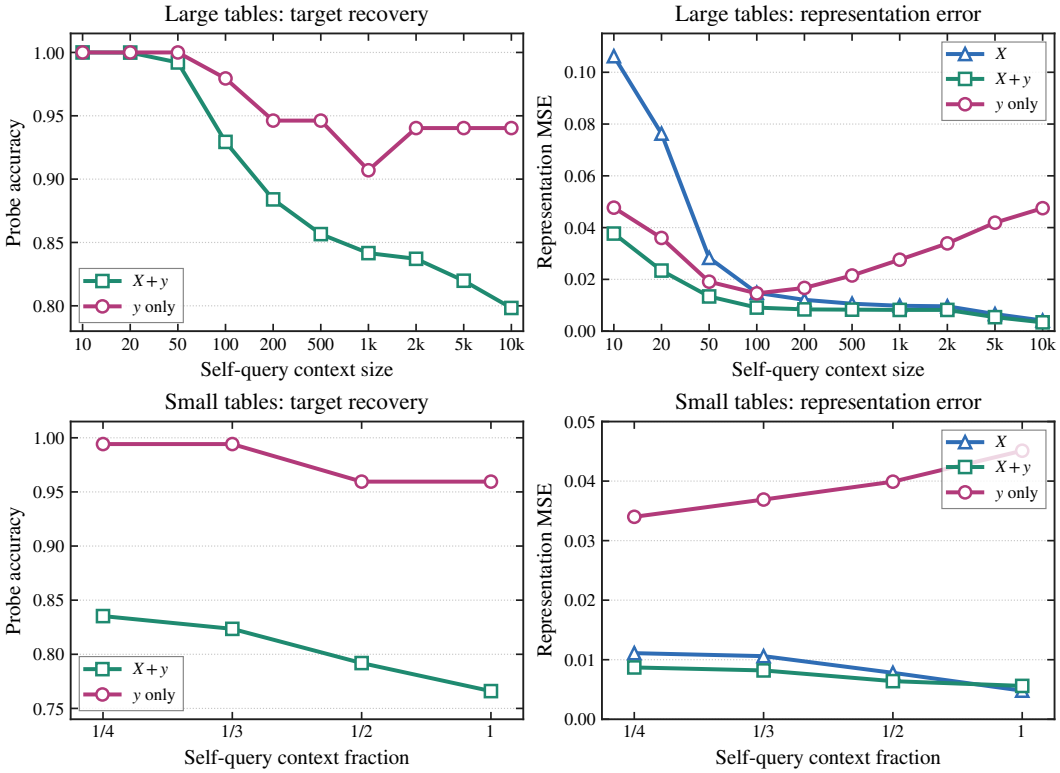


Figure 6: Additional context-size diagnostics on the large- and small-dataset groups used in Figure 3. The left column reports frozen-decoder accuracy after an FT-Transformer regresses cached targets from either features and query labels ($X + y$) or query labels alone (y). The right column reports representation MSE for students receiving X, $X + y$, or y. Results are averaged equally across datasets.

For large tables, label-only decoding remains high as the block grows, even though its representation error increases after the smallest settings. In contrast, the MSE of the X and $X+y$ students decreases with block size. Small tables show the same separation: label-only accuracy remains substantially higher than $X + y$ accuracy, while label-only MSE rises as more of the table enters the self-query

block. The $X + y$ student has the lowest MSE at every large-table setting and at the first three small-table settings; with the full small table, the X-only MSE is slightly lower. These diagnostics show that decoder accuracy and representation distance capture different aspects of target recovery.

C.2 What Do Self-Query Representations Encode?

Following representation-probing analyses (Alain & Bengio, 2017), we test feature recovery from self-query label representations by training a separate neural-network decoder h_j for each feature column x_j. Each probe receives only the cached 192-dimensional representation; the teacher and representations remain frozen. Only the probe decoders are trained in this experiment.

Probe setup. We extract self-query representations with a block size of 1,000, without target correction. For this diagnostic, we pool each dataset's original training and test rows, shuffle them, and form a new 60/10/30 train/validation/test split for the probes, separate from the classification benchmark. Categorical columns use classification decoders; numerical columns are z-normalized and use regression decoders. All metrics are averaged equally over feature columns, excluding the task label y.

Categorical feature recovery. We treat the discrete-valued features in Monks, Jungle, and Balance-scale as categorical targets. Mean recovery accuracy exceeds 97% on all three datasets (Table 4), showing that representations at the label-decoding interface retain substantial information about the input features.

Table 4: Categorical feature recovery from self-query representations. A separate decoder is trained for each feature column. Accuracy is measured on the probe test split; the task label is excluded.

Dataset	Feature columns	Mean accuracy ↑	Column range
Monks	6	0.9816	0.9282–1.0000
Jungle	6	0.9812	0.9634–0.9975
Balance-scale	4	0.9787	0.9521–0.9894

Numerical feature recovery. Table 5 reports RMSE in standardized feature units and R^2 for six feature columns from each of artificial-characters and Electricity.

Table 5: Numerical feature recovery from self-query representations. Metrics are averaged over the evaluated feature columns, excluding the task label. RMSE uses z-normalized feature values.

Dataset	Feature columns	Mean RMSE ↓	Mean R^2 ↑
Electricity	6	0.1454	0.9769
artificial-characters	6	0.0969	0.9898

C.3 Distribution and Channel Structure

We examine the activation distribution and channel structure of the cached self-query representations. Normalization and diffusion scaling are specified in Appendix B.2. The diagnostic sample contains 3,100 query label tokens from eight tables, giving 595,200 scalar activations at embedding dimension $d = 192$. Figure 7 compares the pooled activation distributions before and after the teacher's final LayerNorm.

Observed normalization effects. The raw final-block activations have a pooled standard deviation of 17.67 and range from -184.45 to 198.04. After LayerNorm, the pooled mean is approximately zero, the standard deviation is 1.00, and the range is $[-9.144, 10.752]$. None of the sampled values is changed by clipping. The observed scale reduction therefore comes from LayerNorm.

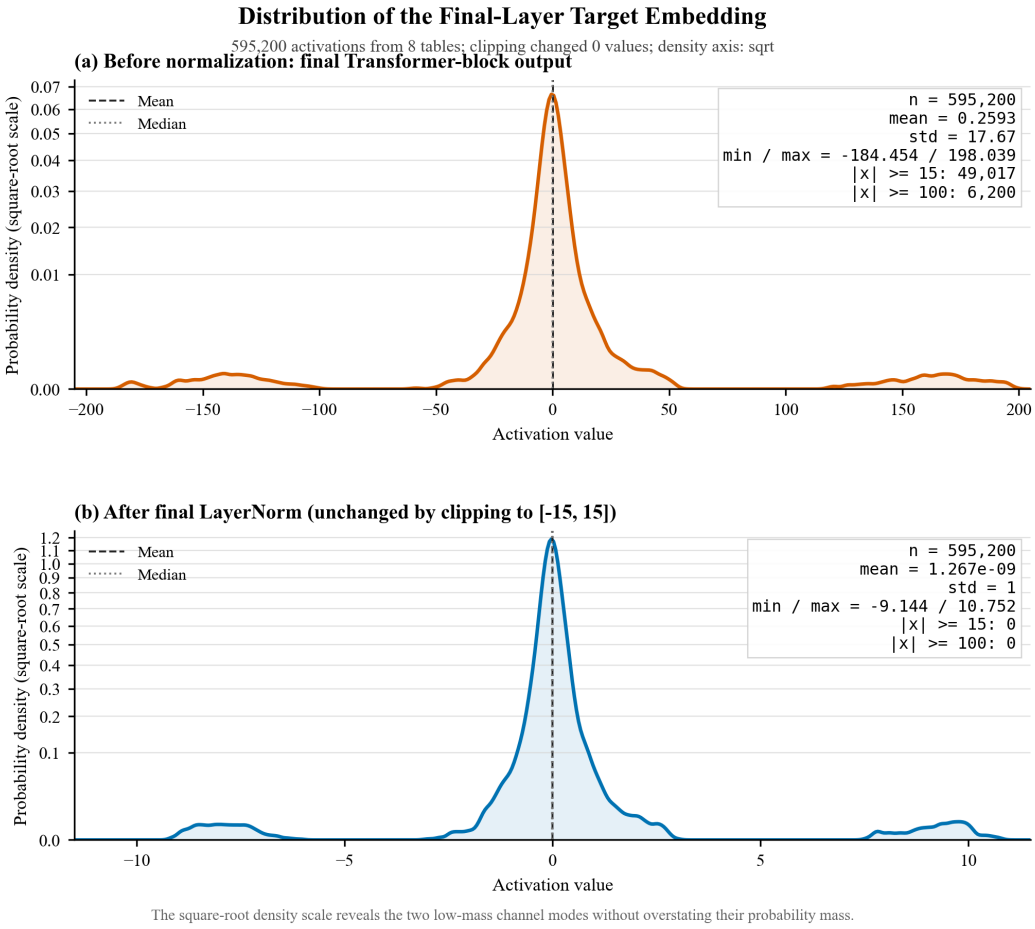


Figure 7: Distribution of self-query label-token activations across eight tables. (a) The final Transformer block output before normalization. (b) The output after the final LayerNorm, which is unchanged by subsequent clipping to $[-15, 15]$. The two low-mass side modes correspond to two fixed channels in the sampled representations. The density axes use a square-root scale to make these modes visible; horizontal scales differ between panels.

Channel structure of the side modes. Two fixed channels account for the extreme raw activations: one is the minimum coordinate in every sampled token, and the other is the maximum. Their mean raw values are approximately -140.10 and 164.91, respectively. Each contributes $1/192 \approx 0.52\%$ of the pooled coordinates; the remaining 190 channels contribute approximately 98.96%. These channel-specific effects explain the two side modes in Figure 7(a). The side modes remain after normalization, with channel means near -7.96 and 9.35. They reflect coordinate structure within the representations, rather than three groups of examples.

C.4 Unsuccessful Target Correction

Larger self-query contexts improve student performance, but also produce more targets that the frozen decoder misclassifies. We therefore explored whether correcting these targets could improve prediction further while preserving their learnability. We kept correctly decoded targets unchanged and re-extracted representations only for examples whose original targets decoded to the wrong class.

Correction procedure. For large datasets, we first kept the self-query context size at 10,000 and resampled the surrounding context, always retaining the query example and its label in the context. We accepted a replacement if its representation decoded to the correct class. If all five attempts failed, we reduced the context size to 90% of its previous value and made five further attempts. If none succeeded, we retained the original cached representation. For small datasets, the initial context already contained the full table, so we reduced its length by a factor of 0.9 to obtain alternative contexts, again retaining the query example. Unsuccessful corrections left the original target unchanged.

Algorithm 1 summarizes the search. Here, $\mathrm{SQ}(\mathcal{C}, i)$ extracts the query label representation for example i using labeled context $\mathcal{C}$, including example i itself; $\mathrm{Decode}(z) = \arg\max_c [g_\theta(z)]_c$ uses the frozen decoder.

Algorithm 1 Self-Query Target Correction by Context Resampling

Require: Labeled table $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$; cached targets $\{z_i\}$
Require: Context cap $K = 10{,}000$; shrink factor $\gamma = 0.9$; attempt budget $B = 5$
Ensure: Updated targets $\{\widetilde{z}_i\}$

$\widetilde{z}_i \leftarrow z_i$ for all i
if $N > K$ **then**
  $\mathcal{S} \leftarrow [K, \lfloor \gamma K \rfloor]$ ▷ Resample, then shrink
else
  $\mathcal{S} \leftarrow [\max(1, \lfloor \gamma N \rfloor)]$ ▷ Shrink full-table context
end if
for each i such that $\mathrm{Decode}(z_i) \neq y_i$ **do**
  accepted $\leftarrow$ false
  for each context size k in $\mathcal{S}$, in order **do**
    for $b = 1, \ldots, B$ **do**
      Sample $k - 1$ rows from $\mathcal{D} \setminus \{(x_i, y_i)\}$ without replacement
      $\mathcal{C} \leftarrow \{\text{sampled rows}\} \cup \{(x_i, y_i)\}$
      $z' \leftarrow \mathrm{SQ}(\mathcal{C}, i)$
      if $\mathrm{Decode}(z') = y_i$ **then**
        $\widetilde{z}_i \leftarrow z'$; accepted $\leftarrow$ true
        break
      end if
    end for
    if accepted **then**
      break
    end if
  end for
end for
return $\{\widetilde{z}_i\}$ ▷ Keep original targets if all attempts fail

Preliminary results. Table 6 compares standard LimiX predictions with student accuracy before and after target correction in joint training on the 23 datasets.

Table 6: Accuracy in the 23-dataset joint target-correction experiment. LimiX uses standard inference; the student is trained with either the original or corrected self-query targets.

LimiX	Student (original targets)	Student (corrected targets)
0.697	0.711	0.686

In joint training on the 23 datasets, corrected targets gave lower student accuracy than the original targets. Selecting replacements by correct teacher decoding therefore did not improve the supervision received by the student. The procedure checked whether a replacement decoded correctly, but did not constrain how readily the student could predict it from standard inputs. These results reinforce the need to evaluate target correction through downstream learning; we use the original, uncorrected targets in TabSQRT.

D Additional Evaluation and Ablations

D.1 Performance on CC18

We additionally evaluate on OpenML-CC18, a curated suite of 72 classification datasets from OpenML (Bischl et al., 2021). The datasets contain between 500 and 100,000 examples and fewer than 5,000 features after one-hot encoding. The suite excludes synthetic datasets and tasks that can be solved perfectly by a single feature or a simple decision tree. Table 7 compares TabSQRT with the five TFM baselines from the main evaluation using Accuracy, AUC, and F1-macro. Our evaluation covers 60 datasets from this suite, excluding tasks with more than ten classes and wide tables that caused out-of-memory errors during evaluation.

Table 7: Mean classification performance on CC18 datasets. Higher values are better for all three metrics. Best and second-best results are highlighted in bold and with underlining, respectively.

Model	Accuracy ↑	AUC ↑	F1-macro ↑
TabPFN-v3	0.8817	0.9310	0.8338
TabICL-v2	**0.8871**	<u>0.9343</u>	<u>0.8419</u>
Real-TabPFN-v2.5	0.8800	0.9292	0.8319
LimiX	0.8846	0.9338	0.8379
TabDPT	0.8755	0.9266	0.8270
TabSQRT	<u>0.8869</u>	**0.9352**	**0.8421**

D.2 Cross-Backbone Ablation Details

This ablation compares student learning from self-query representations supplied by LimiX and TabPFN-v2. Alongside the mean accuracies in Table 3, win-rate comparisons assess how often the trained student outperforms the corresponding backbone's standard predictions across datasets.

Training settings. For each backbone, we train a separate student on each dataset using self-query representations extracted from that backbone. The training configuration largely follows the multi-dataset setup described in Appendix A.2; here, each student's training episodes come from a single dataset. We compare the resulting student with standard inference from its corresponding backbone using identical training splits, preprocessing, and inference-time context.

Table 8 separates large and small datasets before reporting the overall comparison. Each entry counts datasets on which the student wins, ties, or loses against standard inference from the same backbone.

Table 8: Dataset-level wins, ties, and losses of students trained on self-query representations relative to standard inference from the corresponding backbone.

Dataset group	LimiX			TabPFN-v2		
	Win	Tie	Loss	Win	Tie	Loss
Large datasets	7	1	2	6	2	2
Small datasets	7	2	1	7	2	1
Overall	14	3	3	13	4	3

D.3 Additional Analysis of Sampling Behavior

Prediction and reconstruction. NFE=1 yields slightly lower reconstruction MSE than NFE=2, but lower classification accuracy. The decoder's decision depends on the direction of representation error as well as its magnitude, which can account for this difference between the two metrics.

Noise and churn controls. With data, preprocessing, and conditioning inputs held fixed, changing the initial diffusion noise gives a mean class-probability standard deviation of approximately 0.0096 and a predicted-label disagreement rate of approximately 0.6%. Predictions are therefore relatively stable across these seeds. Accuracy still declines with longer trajectories when churn—additional noise injected during sampling—is disabled.

A possible training–sampling mismatch. Training uses noisy clean targets, whereas sampling repeatedly feeds model-generated states back into the denoiser, a form of training–sampling mismatch studied as exposure bias in diffusion models (Ning et al., 2024). Without churn, an Euler update can be written as

$$u_{\text{next}} = \frac{\sigma_{\text{next}}}{\sigma} u + \left(1 - \frac{\sigma_{\text{next}}}{\sigma}\right) D_\phi(u, \sigma, s).$$

A denser schedule changes the weights in this update and evaluates the denoiser at additional intermediate states. Errors at those states can offset a reduction in discretization error. A direct test would compare denoising errors on perturbed clean targets and sampled states at matched noise levels.

E Context Caching and Efficiency

E.1 Caching the Context Branch

During iterative denoising, the labeled context stays unchanged while the query label representations evolve. The context branch of the denoiser does not depend on the query latents or their noise levels. We therefore compute this branch once and cache its sample-attention keys and values at each of the 12 Transformer layers. Subsequent denoiser evaluations reuse these tensors while updating the query branch. This follows the general principle of reusing attention states for repeated static context (Gim et al., 2024). The cache is constructed for a fixed context before sampling and released afterwards. This reduces repeated context computation without changing the model parameters or sampling schedule.

E.2 End-to-End Runtime Evaluation

We compare cached and uncached inference using the same checkpoint, random seeds, and inputs, with two denoiser evaluations (three noise points), one ensemble member, mixed precision, and no additional query chunking. These timing experiments use a separate checkpoint from the main accuracy evaluation, with the clean-test option disabled for the full-suite run. Both implementations use PyTorch scaled dot-product attention on the same GPU with approximately 72 GiB of memory. Caching is enabled only for contexts containing at least 2,048 rows; smaller contexts use the standard implementation.

For the full-suite measurement, we evaluate all 38 TabArena classification datasets, capping both training and test sets at 20,000 rows and using subsampling seed 42. We time the complete evaluation process, including model loading, data reading, preprocessing, inference, metric computation, and output writing. We run each mode twice, reversing their order in the second round, and report the median duration in Table 9.

Table 9: Full evaluation-process time for the 38-dataset TabArena context-cache benchmark. This includes loading and evaluation overhead in addition to model inference.

Context cache	Run 1 (s)	Run 2 (s)	Median (s)	Speedup
Off	70.863	70.249	70.556	1.00×
On	54.335	53.824	54.080	1.30×

Caching reduces total evaluation time by 23.35%, with identical accuracy on all 38 datasets. This comparison measures the implementation change under a fixed checkpoint and evaluation configuration.

E.3 Context Size, Latency, and Memory

We also measure complete `predict()` calls on three tables with 256 query rows each. Each condition is warmed up and repeated five times; Table 10 reports median latency and peak allocated GPU memory, including the feature encoder, diffusion sampler, and decoder. Predicted probability arrays are elementwise identical between the two implementations in all three tests.

Table 10: Per-call latency and memory cost of context caching with 256 query rows and two denoiser evaluations. Memory is peak allocated GPU memory.

Dataset	Context rows	Latency (ms)		Speedup	Memory (MiB)	
		Off	On		Off	On
bank-marketing	2,500	67.94	62.58	1.09×	241.57	262.10
SDSS17	5,000	99.78	80.07	1.25×	338.96	381.66
GiveMeSomeCredit	8,000	149.21	101.11	1.48×	422.96	484.60

Across these workloads, larger contexts yield greater speedups because more repeated computation is avoided. Retaining keys and values across all layers adds approximately 21–62 MiB of peak allocated memory. With only two denoiser evaluations, cache construction and query-branch scheduling can offset the savings for small workloads: a supplementary test with 2,200 context rows and 64 queries was about 4.6% slower with caching. The activation threshold should therefore account for query size and the attention implementation as well as context length.

F Limitations and Future Work

Limitations. The context-size analysis covers ten datasets per group, and the data-scale fit uses five corpus sizes. Table 1 reports standard deviations across five random train–test splits; small differences between leading TFMs should be interpreted alongside this variability. The sampling study measures final predictions and reconstruction errors; explaining the intermediate trajectory requires further experiments.

Future work. Two directions follow from this work. First, we want to correct self-query representations that the teacher decodes incorrectly while preserving the contextual information the student can learn. Second, we will explore whether a generator trained across datasets in a TFM's representation space can synthesize unseen tables without dataset-specific parameter updates. TabFORGE uses pretrained TFM representations for generation but still adapts its diffusion Transformer and decoder to each new dataset (Jiang et al., 2026); eliminating this adaptation is an open direction.

APPENDIX REFERENCES

Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes, 2017. URL https://arxiv.org/abs/1610.01644.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016. URL https://arxiv.org/abs/1607.06450.

Bernd Bischl, Giuseppe Casalicchio, Matthias Feurer, Pieter Gijsbers, Frank Hutter, Michel Lang, Rafael G. Mantovani, Jan N. van Rijn, and Joaquin Vanschoren. OpenML benchmarking suites. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/c7e1249ffc03eb9ded908c236bd1996d-Abstract-round2.html.

In Gim, Guojun Chen, Seung-seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. Prompt cache: Modular attention reuse for low-latency inference. In *Proceedings of Machine Learning and Systems*, volume 6, 2024. URL https://proceedings.mlsys.org/paper_files/paper/2024/hash/a66caa1703fe34705a4368c3014c1966-Abstract-Conference.html.

Xiangjian Jiang, Mingxuan Liu, Nikola Simidjievski, Tassilo Klein, and Mateja Jamnik. Tabular foundation model for generative modelling, 2026. URL https://arxiv.org/abs/2605.09424.

Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design space of diffusion-based generative models. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017. URL https://arxiv.org/abs/1608.03983.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Andreas Lugmayr, Martin Danelljan, Andres Romero, Fisher Yu, Radu Timofte, and Luc Van Gool. RePaint: Inpainting using denoising diffusion probabilistic models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11461–11471, 2022.

Mang Ning, Mingxiao Li, Jianlin Su, Albert Ali Salah, and Itir Onal Ertugrul. Elucidating the exposure bias in diffusion models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=xEJMoj1SpX.

Juntong Shi, Minkai Xu, Harper Hua, Hengrui Zhang, Stefano Ermon, and Jure Leskovec. TabDiff: A mixed-type diffusion model for tabular data generation. In *International Conference on Learning Representations*, 2025.

Hengrui Zhang, Jiani Zhang, Balasubramaniam Srinivasan, Zhengyuan Shen, Xiao Qin, Christos Faloutsos, Huzefa Rangwala, and George Karypis. Mixed-type tabular data synthesis with score-based diffusion in latent space. In *International Conference on Learning Representations*, 2024.

Hengrui Zhang, Liancheng Fang, Qitian Wu, and Philip S. Yu. DiffPuter: Empowering diffusion models for missing data imputation. In *International Conference on Learning Representations*, 2025a.

Xingxuan Zhang, Gang Ren, Han Yu, Hao Yuan, Hui Wang, Jiansheng Li, Jiayun Wu, Lang Mo, Li Mao, Mingchao Hao, et al. LimiX: Unleashing structured-data modeling capability for generalist intelligence. *arXiv preprint arXiv:2509.03505*, 2025b.