LOOPICL: LOOPING A SINGLE TRANSFORMER BLOCK TO SOLVE TABULAR TASKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Tabular foundation models using in-context learning have recently surpassed gradient-boosted trees on predictive tabular tasks. However, recent mechanistic insights suggest that parameters in these models are largely redundant. We introduce *LoopICL*, a looped transformer whose core design decouples parameter count from computational depth. *LoopICL* consists of a single block, processing data through two coupled streams: a cell stream capturing per-cell feature representations and a row stream capturing in-context example representations, jointly refined through within-column and cross-column attention. During pre-training, we vary loop counts, allowing the block to be unrolled for a varying number of iterations at test-time and use a learned exit-gate to automatically exit. In its standard setting, *LoopICL* performs competitively with *TabICLv2* on *TabArena* and *TALENT* at the same computational cost (FLOPs), while using nearly 90% fewer parameters. Furthermore, its recurrent design enables users to also trade off inference cost and performance, providing a resource-aware TFM.

1 INTRODUCTION

Tabular Foundation Models (TFMs) such as *TabICLv2* (Qu et al., 2026), *TabPFN-3* (Grinsztajn et al., 2026), and TabFM (Research, 2026) have surpassed gradient-boosted trees in predictive performance across a wide range of benchmark tasks (Liu et al., 2025; Erickson et al., 2025; Landsgesell et al., 2026). This highlights the potential of In-Context Learning (ICL) for supervised tabular tasks: given labeled training examples as context, these Transformer-based models can directly predict labels for new examples, without requiring task-specific training (Hollmann et al., 2023).

Current frontier TFMs rely on ICL and use a transformer-based model pre-trained on synthetic data generated from a prior by minimizing the loss on a given ICL task. As a result, these prior-data-fitted networks (Müller et al., 2022) amortize predictive inference and generalize to real-world tasks out of the box. Specifically, they rely on deep stacks of independently parameterized transformer layers, resulting in fixed inference costs regardless of task complexity or computational budget. Recent progress has come from better input encoding and re-designing the prior (e.g., *TabICLv2*; Qu et al. (2026)), but is mostly based on continuously scaling up the parameter count (e.g., *TabPFN-3* is $7\times$ larger than *TabPFN v2*; Grinsztajn et al. (2026)). Mechanistic analyses suggest that these models are depth-wise redundant and mainly work through iterative refinement, e.g., increasing separability between classes in the latent space (Balef et al., 2026). This finding is further strengthened by extensive compressibility via task-aware block pruning (Koshil et al., 2026).

Here, we challenge the scaling-up trend and use the insight that a transformer's computational depth is not necessarily tied to the number of unique weights and layers. So-called looped transformers replace a conventional stack of independently parameterized layers with repeated applications of the same transformer block (Mostafa et al., 2019), thus decoupling parameter count from depth. This parameter sharing enables the model to increase its effective computational depth through iterative refinement while keeping the parameter count fixed, with the number of iterations adjustable at inference time to trade computational cost for predictive performance (Geiping et al., 2026).

This mechanism seems well suited for tabular tasks: Once tokenized, a supervised tabular task can be approached through iterative refinement of the latent representation, followed by a lightweight decoder to label query points. This differentiates tabular ICL from knowledge-intensive NLP tasks, which rely on retrieving learned facts and thus benefit from larger parameter counts (Morris et al.),

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.

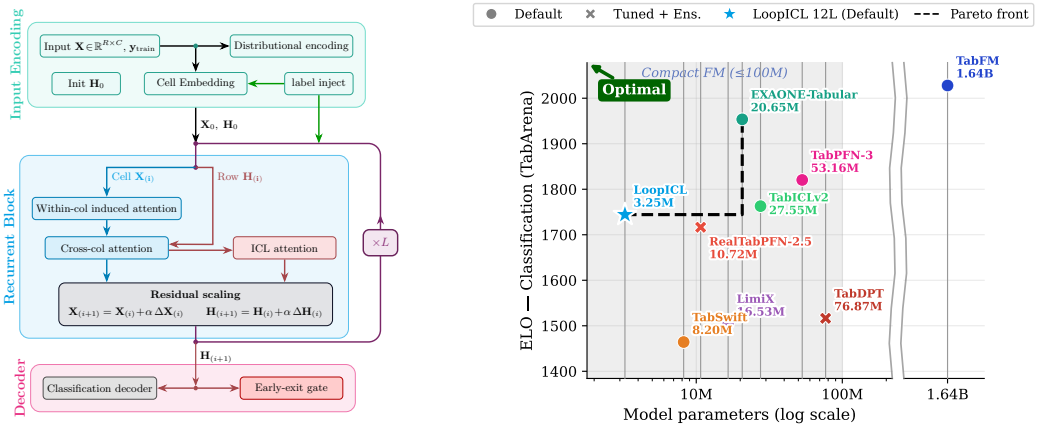

Figure 1: Left: *LoopICL* architecture overview. The blue boxes operate on the **cell stream** (per-cell representations); the red box operates on the **row stream** (per-row ICL representations). Right: *LoopICL* is highly parameter-efficient by reusing the same parameters across recurrent steps.

and is consistent with the observation that existing TFMs exhibit different latent-space dynamics and redundancy compared to LLMs (Balef et al., 2026).

Contributions. We propose *LoopICL*, to the best of our knowledge, the first recurrent Transformer architecture demonstrated at scale for tabular ICL, combining parameter sharing with iterative computation. At its core is a *single transformer block* that contains a cell stream capturing per-cell feature representations and a row stream capturing in-context example representations, which is repeatedly applied to refine internal representations. A novel distributional conditioning on the input tokens stabilizes training and further improves predictive performance. On standard benchmarks, our model pushed the Pareto front of the accuracy–parameter trade-off, matching or exceeding the performance of 9-times larger TFMs (e.g., *TabICLv2*).

To understand the behavior and capabilities of *LoopICL*, we conduct a detailed empirical analysis addressing the following research questions.

(RQ1) Can a single looped Transformer block match or exceed the predictive performance of the respective fixed-depth TFM? We first establish recurrent models as a novel and competitive alternative for solving predictive tabular tasks.

(RQ2) How does *LoopICL* refine its representations across recurrent steps? We show that explicit residual scaling enables smooth refinement and extrapolation to high loop counts, and implicitly learned scaling enables the model to exit early at the minimum loops needed.

(RQ3) How does *LoopICL*'s performance compare to substantially larger, fixed-depth frontier TFMs? Finally, we position our contribution in the context of the cost-efficiency of frontier TFMs.

2 Related Work and Background

LoopICL connects three lines of prior work. First, as a *tabular foundation model* (TFM), it builds on transformer architectures for ICL on tabular data. Second, its recurrent design relates to *looped* and weight-tied Transformers, which replace independently parameterized layers with repeated applications of the same block. Third, we study the behavior and scaling of recurrent computation, where varying the number of recurrent iterations provides an inference-time compute budget, connecting *LoopICL* to *adaptive computation* and compute-efficient inference.

Related TFM Architectures. TFMs are pretrained Transformer-based models that leverage ICL for predictive tabular tasks. Given a set of labeled examples as context, the models predict labels for new examples directly from these examples, without requiring task-specific parameter updates (Müller et al., 2022). *TabPFN* (Hollmann et al., 2023) pioneered this paradigm using a vanilla encoder-only architecture. Building on this design, *TabSwift* (Liu & Ye, 2026) added gated attention for

built-in support for Early Exit (EE). In 2025, Hollmann et al. (2025) introduced *TabPFNv2*, which use 2D-attention transformers (Kossen et al., 2021) applying attention along the row and column axes. *TabICL* (Qu et al., 2024) uses dedicated compression stage, followed by an encoder-only transformer to scale to larger contexts. *TabICL v2* (Qu et al., 2026) further introduces query-aware scalable softmax attention for better generalization to larger datasets, an architecture subsequently adopted and improved by *TabPFN-3* (Grinsztajn et al., 2026). *EXAONE* (Eo et al., 2026) introduces feature-summary and item-summary tokens, achieving state-of-the-art performance.

Recurrent tabular models. The idea of iterative refinement in tabular prediction dates back to classical ensemble methods such as gradient boosting (Friedman, 2001) successively fitting weak learners to the residuals of previous rounds. Their strong performance on tabular benchmarks (Grinsztajn et al., 2022; McElfresh et al., 2023) provides evidence that this mechanism is well suited for tabular tasks. Also deep learning methods have explored iterative computation through recurrent and state-space architectures. *Mambular* (Thielmann et al., 2024) and the SSM-based TFM of Koch et al. (2025) use state-space models for tabular prediction, while Padayachy et al. (2026) applies the *Tiny Recursive Model* (Jolicoeur-Martineau, 2025) to insurance pricing and Komisarczyk et al. (2026) proposes a hybrid attention–state-space architecture. These works show that iterative computation can serve as the main principle for tabular tasks, further reinforced by the observation that even fixed-depth TFMs develop an inductive bias toward iterative refinement (Balef et al., 2026).

Recurrent Transformers. Recurrent Transformers repeatedly apply shared blocks to intermediate representations, increasing computational depth without increasing the number of parameters (Mostafa et al., 2019). Saunshi et al. (2025) show that looped models can achieve strong reasoning performance with substantially fewer parameters and exhibit an inductive bias toward reasoning rather than memorization. Similarly, Yang et al. (2024) demonstrate that weight sharing across iterations enables to efficiently emulate iterative learning algorithms. Fixed-Point Reasoners interpret the process as convergence toward a stable representation and use convergence-based halting to allocate computation adaptively (Movahedi et al., 2026). Similar ideas have also been explored in visual generation (Goyal et al., 2026) and audio processing (Kaloga et al., 2026). In the tabular domain, Balef et al. (2026) provide a small-scale proof of concept for using recurrence to iteratively refine latent representations. Collectively, these studies establish recurrence as a mechanism for increasing effective computation while maintaining parameter efficiency, however, it remains unclear how to scale computation to achieve competitive performance with large contexts.

Understanding recurrent Transformers. Several works suggest that recurrent computation progressively refines and stabilizes representations across iterations (Movahedi et al., 2026; Blayney et al., 2026), while the training-time distribution of recurrent steps shapes representation quality and generalization to unseen depths (Geiping et al., 2026). Mechanistically, Blayney et al. (2026) find the same recurrent block can perform different stages of computation across iterations. This suggests that each iteration can progressively carry out a different part of the overall computation, even though the same block is reused. In the tabular domain, Balef et al. (2026) similarly observe transitions in representations across TFM layers, suggesting distinct stages of computation. However, how representations evolve under recurrent computation in tabular ICL remains unexplored. We therefore investigate the dynamics of *LoopICL* across recurrent steps.

Adaptive computation. Recurrence has traditionally been used to model sequential dependencies in recurrent neural networks (Elman, 1990), but it has also been recognized as a mechanism for adaptive computation (Graves, 2016) and for introducing recurrent depth into Transformers (Mostafa et al., 2019). Adaptive computation can be monitored via representation stability (Movahedi et al., 2026) or prediction convergence (Geiping et al., 2026). Dedicated prediction heads can determine when to halt computation (Zhu et al., 2025). More recently, Jeddi et al. (2026) showed that adaptive-computation strategies can be fragile when applied to looped Transformers and proposed budget-conditioned reasoning, where the user specifies the computational budget at inference time. Küken et al. (2025) propose an early-exit strategy for TFMs that use an entropy threshold as a proxy metric to exit the forward pass. More recently, Liu & Ye (2026) proposed a per-query early-exit inference method, in which each query is processed independently, and the exit decision is made on a per-query basis. Their approach attaches lightweight prediction and exit heads to a subset of Transformer layers, enabling intermediate predictions and input-dependent stopping decisions.

3 ARCHITECTURE

Figure 1 (Left) shows an overview of the architecture; below, we highlight the key architectural innovations and components going through input encoding, residual scaling, and output decoding, while Appendix A provides full architectural details.

Distributional Input Encoding. Raw features are standardized per column and encoded via cyclic feature grouping into E-dimensional cell embeddings (Qu et al., 2026). We inject class labels via an orthogonally initialized embedding matrix (Grinsztajn et al., 2026) and re-inject them at each recurrent step. Before the recurrent loop, we additionally introduce *distributional encoding*, which uses three complementary distribution-aware signals: *marginal histograms*, *discriminative histograms*, and *Fourier quantile encodings*. These histogram-based signals capture marginal and class-discriminative feature distributions, while Fourier quantile encodings capture relative feature rankings and provide information about potential out-of-distribution test samples.

Recurrent Block. *LoopICL* processes data through two asymmetrically coupled streams: a *cell stream* of individual cell representations, $\mathbb{R}^{R\times C\times E}$, and a compact *row stream*, $\mathbb{R}^{R\times D}$, with $E = 128$ and $D = 4E$. This separation preserves fine-grained feature interactions while enabling efficient row-wise ICL, as relying solely on compressed row representations can lead to information loss (Eo et al., 2026). At each recurrent iteration, the cell stream first models local feature interactions and is then read out into the row stream. The row representations are refined through in-context attention, while the updated cell stream is retained and reused in the next iteration.

More specifically, each iteration applies a single shared axial block: within-column self-attention operates independently on each column, while cross-column self-attention with RoPE positional encodings operates across each row (Su et al., 2024). The cross-column self-attention updates the cell stream, followed by a CLS-based readout into the row stream, which is then updated by causally masked ICL attention. All sub-blocks use sandwich normalization (pre- and post-RMSNorm) (Ding et al., 2021).

Residual Scaling. Since *LoopICL* applies the same block for $L \in \mathbb{N}^+$ iterations, residual scaling can improve training stability by controlling the accumulation of updates (Wang et al., 2026; Movahedi et al., 2026). After each iteration, given the block output $\tilde{\mathbf{x}}$, it constrains the update with alpha:

$$\mathbf{x} \leftarrow \mathbf{x} + \alpha(\tilde{\mathbf{x}} - \mathbf{x}), \tag{1}$$

where $\alpha \in (0, 1]$ is a configurable scaling factor, comparable to the step size in an optimization procedure; $\alpha = 1$ means no scaling, and smaller values reduce the magnitude of each update, preventing the residual stream from diverging as L increases. A fixed α may not generalize across highly varying L values and encourages the model to greedily front-load computation into early iterations and actively suppress updates in later iterations. Alternatively, a depth-normalized scaling scheme that sets α as a function of a pre-defined loop depth L encourages the model to spread computation uniformly across the given budget. To observe the impact of this in practice, we empirically study three variants ranging from no decay to linear depth normalization: $\alpha \in \{1, 1/\sqrt{L}, 1/L\}$.

Decoder. Following Grinsztajn et al. (2026), we use an attention-based retrieval decoder that is target-permutation equivariant (Arbel et al., 2026). The normalized row-stream representations of test examples serve as queries and those of training examples serve as keys, with attention scores accumulated separately per class via one-hot label weighting (Koshil et al., 2025). We then average the resulting per-class attention masses across heads to produce class probabilities. Additionally, we introduce an early-exit gate that predicts whether further iterations are likely to improve performance. A compact MLP uses only training-set representations, combining the current state, its rate of change, the iteration index, and dataset size to produce an exit probability $\lambda \in (0, 100)$.[1] Further details are provided in Appendix A.4.

At test time, we use an ensemble of $N{=}8$ members, each receiving a differently transformed view of the data (permuted feature order, shuffled class labels, varied preprocessing); predictions are averaged across members. The early-exit gate operates independently per member, so each member may exit at a different loop step. Full ensemble details are provided in Appendix B.

[1] Query-based early exit, as widely used in LLMs, may no directly transfer to TFMs. Unlike autoregressive LLMs, TFMs process all context elements in a single forward pass. Even if confident predictions are available for some query points, looping continues till all query points can be predicted. Consequently, we base our early-exit decision on the embeddings of the training samples and exit once iterative refinement converges.

3.1 PRETRAINING

We pretrain *LoopICL* in two stages on synthetic tasks generated from the *TabICLv2* graph-based SCM prior (Qu et al., 2026). Here, we summarize the key settings, full details are in Appendix C.

Recurrent loop schedule. The number of recurrent iterations for each training step is sampled from a log-normal Poisson distribution (Geiping et al., 2026) to improve depth generalization and performance retention (Schwarzschild et al., 2021; Yang et al., 2026). We sample $\gamma \sim \text{LogNormal}(\mu, 0.5^2)$ and $L \sim 1 + \text{Poisson}(\gamma)$, with L clipped to $[1, 8]$. We choose μ such that $\mathbb{E}[L] = 6$ before clipping, meaning the model learns on average from 5.32 loops during pretraining (Figure 2). Prior work optimizes the expected loss across recurrent depths (Zhu et al., 2025; Geiping et al., 2026). In contrast, we compute the cross-entropy loss only at the final iteration, training the model to achieve peak performance at the given L. Since our residual scaling in *LoopICL* depends on the number of iterations, this objective explicitly trains the model to use all iterations effectively.

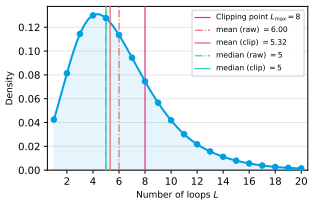

Figure 2: The number of recurrent iterations L is sampled from a log-normal Poisson distribution during pretraining.

Stage 1: Pretraining (500K steps). We train for 500,000 steps with batch size 64 on datasets containing 2-100 features, up to 10 classes, and up to 1,024 rows. Train/test split ratios are uniformly sampled from $[0.3, 0.9]$. We use full backpropagation through all recurrent iterations. Optimization uses Muon (Jordan et al., 2024) with learning rate 8×10^{-4} and momentum 0.9. We use a 5,000-step warmup followed by cosine decay over the final 10% of training, gradient clipping at 1.0, weight decay 0.01.

Stage 2: Long-context continued pretraining (30K steps). We continue training for 30,000 steps on larger synthetic tasks with 1–2,000 features and up to 100,000 rows, subject to a feature×row budget of 10^6. We use truncated backpropagation through time (TBPTT) (Geng et al., 2021) with a window of $k = 5$, propagating gradients only through the final five loops and detaching earlier activations. This reduces memory and computation while preserving parameter sharing across loops, allowing us to pretrain with a larger number of loops than would be feasible with full backpropagation under our GPU memory constraints. We use a learning rate of 10^{-4} and 400 warmup steps, with all other optimization settings kept the same as in Stage 1.

Stage 3: Early-exit gate (4K steps). Starting from the Stage 2 checkpoint, we freeze the backbone and train the early-exit gate for 4,000 steps with the same setting as Stage 2. The gate is trained on per-loop prediction losses: it continues if future iterations improve the loss and exits otherwise. The gate uses only training embeddings and makes its decision based on the training samples, without using information from the test set. When the gate probability exceeds a threshold λ, the recurrent loop terminates, and the decoder is then run to obtain predictions. Notably, we train the early-exit gate with a maximum of 16 loops, which is twice the pretraining limit.

4 EXPERIMENTS

In this section, we first conduct a series of ablation studies on *LoopICL* at computationally feasible intermediate pre-training checkpoints and investigate the model's performance across different layers and computational budgets (extensive results are in Appendix D). Finally, we evaluate *LoopICL* on *TabArena* and *TALENT* benchmarks and compare its performance against state-of-the-art models.

4.1 (RQ1) CAN A SINGLE LOOPED TRANSFORMER BLOCK MATCH OR EXCEED THE PREDICTIVE PERFORMANCE OF THE RESPECTIVE FIXED-DEPTH TFM?

Setup. We evaluate checkpoints from Stage 1 (100K steps). We study classification performance averaged across 1,000 *synthetic tasks* sampled from the prior and across 10 folds of the 54 *development set* used in *TabICLv2* and *TabPFNv2* (Hollmann et al., 2025), without any ensembling. We use up to 2,048 rows per dataset to match the pretraining regime, following (Qu et al., 2026). We compare a fixed-depth TFM comprising six *Stacked blocks* (—), matching the approximately $\mathbb{E}[L] = 5.21$ loops used during pretraining. As recurrent baselines, we consider *LoopICL* with $L = 6$ with dif-

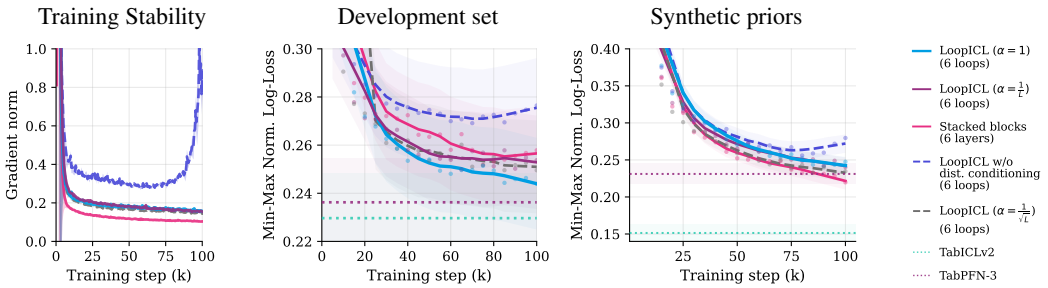

Figure 3: **Ablation study.** Training stability (gradient norm) and predictive performance (normalized log loss averaged across tasks) of recurrent and non-recurrent model variants.

ferent residual scaling schedules $\alpha \in \{1, 1/L, 1/\sqrt{L}\}$. Additionally, we evaluate a variant without distributional input encoding (*w/o dist. conditioning*; - -).

Pretraining stability. The left panel of Figure 3 compares the gradient norms as a proxy to training stability. Comparing the stacked blocks — and the recurrent model —, we observe that recurrent models are more challenging to train, exhibiting generally larger gradient norms. More interestingly, comparing the — and - - curves shows that our *distributional encoding* not only reduces gradient norm and, thus, improves training stability, but also achieves a better predictive performance across both *synthetic priors* and the *development set*.

Finding 1. Recurrence makes optimization challenging due to greater gradient instability.

Finding 2. Distributional conditioning increases training stability and predictive performance.

Performance. The middle and right panel of Figure 3 compares the stacked (—) with recurrent variants (—, - -, - -) on *synthetic prior* and *development set* tasks. On the *synthetic priors* tasks, the stacked model achieves lower in-distribution loss, potentially reflecting its larger parameter count and greater capacity.[2] In contrast, the recurrent variants achieve lower loss on the *development set* tasks, indicating better generalization to unseen tasks. We also observe stronger generalization to large context sizes for our recurrent models (see Appendix D.1)

Residual Scaling. Finally, we compare different residual scaling strategies and revisit Figure 3. Overall, residual scaling has only a minor effect on the gradient norms; however, although we use a fixed L, performance on the *development set* tasks varies. This raises the question of how residual scaling impacts stability during looping, which we study next.

4.2 (RQ2) How does *LoopICL* refine its representations across recurrent steps?

The most intriguing ability of looped models is that we can increase computational depth at inference time. Residual scaling can be used to controls this ability. Thus, we study how different scaling schemes impact generalization performance and internal representations during refinement.

Setup. We continue pretraining for our looped models with different values of α for 10,000 more steps in the Stage 2 setting to enable longer-context processing. We use all available samples in the *development set* tasks and use the setup from the previous experiment otherwise.

Impact on performance. We first study how performance evolves during looping and evaluate predictions at each loop step up to $L = 12$ (see Figure 4). Aggressive fixed residual scaling ($\alpha = 1$, —) leads to slower improvements in performance, however, adaptive scaling ($\alpha = 1/\sqrt{L}$, - -) may achieve better performance at a higher loop count. This suggests that, in settings such as early exit, ($\alpha = 1$, —) may be a preferable choice, as it can achieve stronger performance at earlier loop steps.

[2]We note, that the fixed-depth may require additional training to undergo grokking to transition from memorization to generalization.

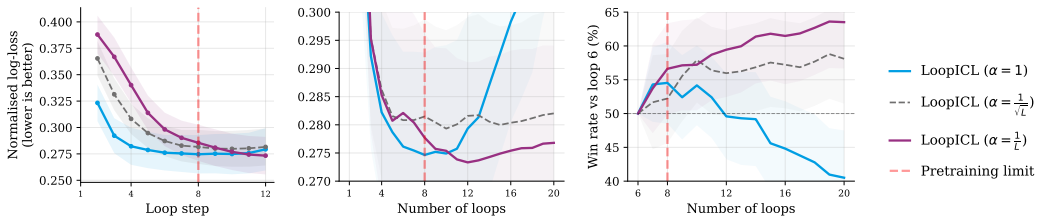

Figure 4: **Residual scaling improves extrapolation.** Left: per-iteration norm. log loss for $L = 12$. Middle: norm. log loss for $L \in [1, 20]$. Right: win rate relative to performance at $L = 6$, for $L \geq 6$.

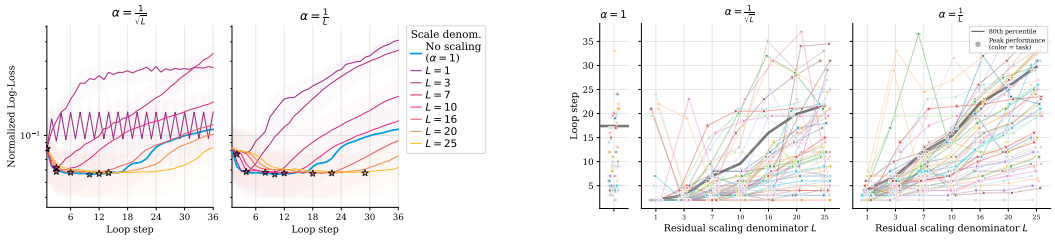

Figure 5: **Residual scaling ablation.** Left: Performance across loop steps for different L and residual scaling schemes. Stars indicate the optimal loop step. Right: Optimal loop steps across datasets, showing dataset-dependent computation and the effect of L on budget allocation.

In the middle and right panel of Figure 4, we compare performance across increasing values for L.[3] ($\alpha = 1$, —) performs best within the range of loop counts sampled during pretraining, but degrades with high loop counts. In contrast, adaptive residual scaling using ($\alpha = 1/L$, —) steadily improves performance with more loops even extrapolates beyond the pre-training limit. However, as the scaling factor depends on the number of loops which, in practice, must be determined in advance. With a fixed scaling $\alpha = 1$, L does not need to be set in advance and the model can exit anytime.

Finding 3. Residual scaling enables stable extrapolation to loop counts beyond pretraining.

Impact on model behavior. Next, we characterize convergence behavior. For this we vary the value of L (impacting residual scaling) and report per-loop performance, including the best-observed performance, across 1-36 loops in the left panel of Figure 5. Overall, smaller values of L lead to faster convergence, i.e. the best iteration, while increasing the number of iterations can cause performance divergence. Interestingly, for $\alpha = 1/\sqrt{L}$ and $L = 1$, we observe substantial fluctuations in performance. This behavior is reminiscent of the oscillatory or non-convergent dynamics that can arise in iterative fixed-point methods, which have recently been studied in the context of looped Transformers (Movahedi et al., 2026). As L increases, convergence becomes slower, while the region of near-optimal performance (error ≤ 0.001) gets broader. Also, we note that the best iteration not necessarily coincides with L, highlighting a possibility for further tuning.

The right panel of Figure 5 shows that the optimal number of loop steps is task-dependent. For the $\alpha = 1$ variant, on average, approximately half of the datasets converge within 8 loops, while the remaining datasets require more iterations. For $\alpha = 1/\sqrt{L}$ and $\alpha = 1/L$ the optimal loop count varies with L, indicating that the model adapts its computation to the given budget: the effective number of iterations required to reach near-optimal performance increases with L. In particular, $\alpha = 1/L$ provides an effective allocation as the optimal number of loop steps approximates L.

Finding 4. With $\alpha = 1$, the model optimally scales updates across all loop steps within the pretraining range. Adaptive scaling using $\alpha = 1/L$ enables budget-aware computation, achieving near-optimal performance around L loop steps, including beyond the pretraining range.

[3] Note that for $\alpha = 1$ the value of L does not change model behavior.

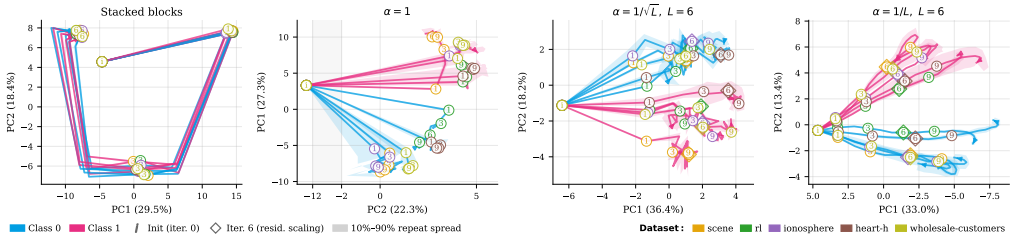

Figure 6: **Latent representation trajectories.** PCA of embeddings across loops for binary tasks. Residual scaling with $\alpha = 1/L$ induces structured and uniform progress toward separating classes.

Internal representations. Finally, we analyze how the internal representations and performance evolve at each loop step (see Appendix D.2 for further details). Figure 6 shows the evolution of the embedding space across loop steps. We track internal embeddings for five binary classification tasks from the *development set* across $L = 6$ loops and visualize the first two PCA components. Starting from the right-most panels, $\alpha = 1/\sqrt{L}$ and $\alpha = 1/L$ exhibit a very structured and similar behavior. PC2 primarily captures class separation across the datasets, while PC1 is strongly correlated with the loop step. The model continuously changes its internal state with more loops. Interestingly, for $\alpha = 1/L$, the loop steps are encoded almost uniformly resulting in linear trajectories, suggesting that $\alpha = \frac{1}{L}$ induces a controlled iterative refinement. For $\alpha = 1$ PC1 (27.3% explained variance) also captures class separation, however, the evolvement of representation through PC2 is less systematic, suggesting that the representations quickly organize according to the target classes. The left panel shows the fixed-depth model as a reference (more details in Appendix D.3).

Finding 5. $\alpha = 1/L$ yields more stable and uniform encoding of loop steps, leading to smoother latent representation trajectories and more consistent iterative refinement.

4.3 (RQ3) How does *LoopICL*'s performance compare to substantially larger, fixed-depth frontier TFMs?

Finally, we study performance on *TabArena* (Erickson et al., 2025) and *TALENT* (Liu et al., 2025). We compare against state-of-the-art TFMs and report improvability vs. inference costs in Figure 7 (detailed results are in Appendix D.4). We evaluate *LoopICL* using an ensemble of 8 members. We choose $\alpha = 1$ to apply our early exit gate to showcase both ways to control the performance-cost tradeoff: by fixing the number of loops L we define the compute budget to be used and with the early-exit threshold $\lambda > 0$ we specify a confidence threshold required to justify another loop.

Specifying L, the performance of *LoopICL* improves with more loops, peaking at around $L = 12$ loop steps, competitive with *TabICLv2*. *TabICLv2* uses 12 ICL blocks and hidden dimensions of the same size as *LoopICL*, thus, the dominant FLOPs cost is the same (runtime measurements might differ due to different hardware (see caption of Figure 7); *LoopICL* incurs only a minor overhead from additional cheap row-interaction and set-transformer iterations. Notably, *LoopICL* achieves this at 88.2% fewer parameters. More importantly, these results demonstrate that our model robustly extrapolates beyond its pretraining limit on real-world tasks and operates effectively at roughly twice the average number of loop steps observed during pretraining.

Results using the early-exit gate (*LoopICL*-EE) confirm this as a reliable alternative to relying on a pre-defined, fixed number of loop steps. The left panel in Figure 8 reports the median speedup relative to the 16-loop baseline (i.e., early exit disabled, 8 ensemble members each running 16 loops), which serves as the upper bound on the number of loops executed. As λ increases, the model exits more conservatively, yielding better performance at the cost of lower speedup. At λ=90 we observe both a speedup and a win-rate gain over the 16-loop baseline on both benchmarks.

The center and right panels in Figure 8 compare λ=90 against the best-performing fixed-loop configuration (12 loops). On *TabArena*, EE is faster on 33 out of 38 datasets, and 12 out of 38 datasets gain both speed and accuracy simultaneously. Some datasets do incur a performance loss, though the maximum relative error increase remains within 6%. On *TALENT*, we observe higher overall

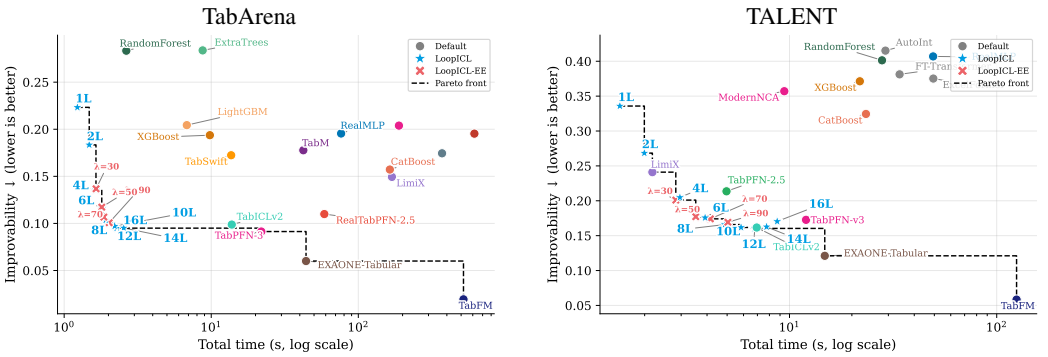

Figure 7: Improvability (lower is better) measures the relative error gap to the best method, averaged across datasets. Time is training + inference. For *TALENT*, all models are evaluated on an A100 GPU under identical conditions, enabling a direct runtime comparison. For *TabArena*, competitor runtimes are taken from the published benchmark and may reflect different hardware; these results should not be used for direct runtime comparisons of *LoopICL* with other baselines.

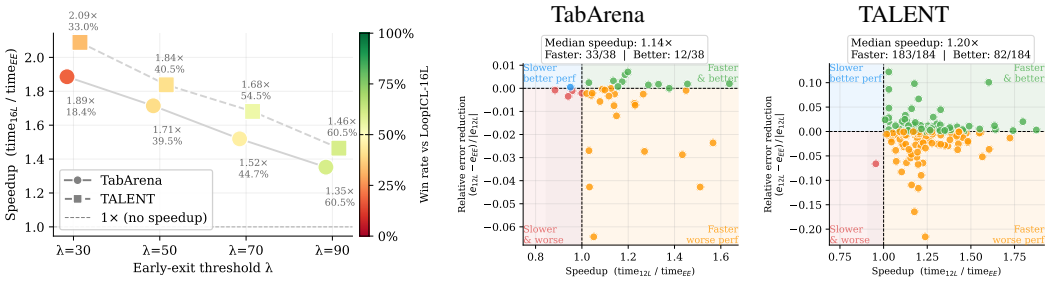

Figure 8: *Left:* Median speedup of *LoopICL*-EE over the full 16-loop baseline across different λ; color indicates the fraction of datasets where EE wins. *Center / Right:* Per-dataset speedup vs. relative error reduction for λ=90 on *TabArena* and *TALENT* , compared to the 12-loop baseline (the best-performing fixed-loop configuration). Each point is one dataset; $x > 1$ means EE is faster, $y > 0$ means EE is more accurate. Points in the green quadrant achieve both simultaneously.

speedup and win rate; however, the magnitude of both gains and losses is larger, with maximum relative error gains and losses reaching approximately 10% and 20%, respectively.

5 Conclusion

Our results revisit the central question: does tabular ICL require *more parameters*, or simply *more computation*? *LoopICL* suggests the latter. A single shared block, applied recurrently, matches substantially larger fixed-depth TFMs on *TabArena* and *TALENT*. This also provides evidence that tabular foundation models may rely more on learning iterative task-solving algorithms than on memorizing and recalling knowledge, as is common in language models. Across RQ1–RQ3, we find that weight sharing improves generalization, recurrent computation enables stable iterative refinement even beyond the pretraining depth, and the number of loops can adapt to dataset difficulty at inference time. **These results suggest that some benefits of model scaling may arise from increased computational depth rather than parameter count, motivating a simple alternative: instead of growing the parameter budget, grow the loop budget.**

Limitations and future directions. The current model supports classification tasks; extending it to regression tasks is left for future work. Future work should scale *LoopICL* in both model size and the number of recurrent iterations, for example by repeating N layers rather than a single layer. Studying scaling laws for tabular foundation models is another promising direction. Finally, inspired by low-rank structure observed in KV caching across iterations in looped language models (Vendrell et al., 2026; Neill & Reid, 2026), it would be interesting to investigate whether similar structure exists in the tabular domain and can be exploited for more efficient inference.

AI USE STATEMENT

In this work, we used generative AI tools for polishing text and identifying errors. AI assistance was used during the development of the project for documentation, code cleaning and editing, development of small helper functions, and plotting. We did not use generative AI tools to propose or refine the method or hypotheses, or to generate data sets. Tasks related to mathematical proofs, translation and qualitative data analysis are not applicable to this work. We have reviewed all AI-assisted work and take full responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Code is available at `https://anonymous.4open.science/r/loopicl/` for now and will be made fully open-source. The repository includes training scripts, evaluation code for reproducing the results reported in the paper.

REFERENCES

Michael Arbel, David Salinas, and Frank Hutter. EquitabPFN: A target-permutation equivariant prior fitted network. *Advances in Neural Information Processing Systems*, 38:62586–62609, 2026.

Thomas Bachlechner, Bodhisattwa Prasad Majumder, Henry Mao, Gary Cottrell, and Julian McAuley. Rezero is all you need: Fast convergence at large depth. In *Uncertainty in artificial intelligence*, pp. 1352–1361. PMLR, 2021.

Amir Rezaei Balef, Mykhailo Koshil, and Katharina Eggensperger. Is one layer enough? understanding inference dynamics in tabular foundation models. In *Forty-third International Conference on Machine Learning*, Proceedings of Machine Learning Research. PMLR, 2026.

Hugh Blayney, Álvaro Arroyo, Johan Obando-Ceron, Pablo Samuel Castro, Aaron Courville, Michael M Bronstein, and Xiaowen Dong. A mechanistic analysis of looped reasoning language models. *arXiv preprint arXiv:2604.11791*, 2026.

Ming Ding, Zhuoyi Yang, Wenyi Hong, Wendi Zheng, Chang Zhou, Da Yin, Junyang Lin, Xu Zou, Zhou Shao, Hongxia Yang, et al. Cogview: Mastering text-to-image generation via transformers. *Advances in neural information processing systems*, 34:19822–19835, 2021.

Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

Moonjung Eo, Min-Kook Suh, Hye-Seung Cho, Jiwon Kim, Seoyoon Kim, Sangjun Nam, and Soonyoung Lee. Exaone tabular 1.0 : Technical report. *arXiv preprint arXiv:2608.25774*, 2026.

N. Erickson, L. Purucker, A. Tschalzev, D. Holzmüller, P. M. Desai, D. Salinas, and F. Hutter. TabArena: A living benchmark for machine learning on tabular data. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*. Curran Associates, 2025.

Ying Fan, Yilun Du, Kannan Ramchandran, and Kangwook Lee. Looped transformers for length generalization. In *International Conference on Learning Representations*, volume 2025, pp. 14502–14520, 2025.

J. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, pp. 1189–1232, 2001.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. *Advances in Neural Information Processing Systems*, 38: 41340–41391, 2026.

Zhengyang Geng, Xin-Yu Zhang, Shaojie Bai, Yisen Wang, and Zhouchen Lin. On training implicit models. *Advances in neural information processing systems*, 34:24247–24260, 2021.

Sahil Goyal, Swayam Agrawal, Gautham Govind Anil, Prateek Jain, Sujoy Paul, and Aditya Kusupati. Elt: Elastic looped transformers for visual generation. *arXiv preprint arXiv:2604.09168*, 2026.

Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

L. Grinsztajn, E. Oyallon, and G. Varoquaux. Why do tree-based models still outperform deep learning on typical tabular data? In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Proceedings of the 35th International Conference on Advances in Neural Information Processing Systems (NeurIPS'22)*. Curran Associates, 2022.

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. *arXiv preprint arXiv:2605.13986*, 2026.

Steffen Herbold. Autorank: A python package for automated ranking of classifiers. *Journal of Open Source Software*, 5(48):2173, 2020. doi: 10.21105/joss.02173. URL https://doi.org/10.21105/joss.02173.

N. Hollmann, S. Müller, K. Eggensperger, and F. Hutter. TabPFN: A transformer that solves small tabular classification problems in a second. In *The Eleventh International Conference on Learning Representations (ICLR'23)*. ICLR, 2023.

N. Hollmann, S. Müller, L. Purucker, A. Krishnakumar, M. 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.

Ahmadreza Jeddi, Marco Ciccone, and Babak Taati. Loopformer: Elastic-depth looped transformers for latent reasoning via shortcut modulation. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=RzYXb5YWBs.

Alexia Jolicoeur-Martineau. Less is more: Recursive reasoning with tiny networks. *arXiv preprint arXiv:2510.04871*, 2025.

Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024. *URL https://kellerjordan. github. io/posts/muon*, 6(3):4, 2024.

Yacouba Kaloga, Shashi Kumar, Shakeel A Sheikh, Driss Khalil, Petr Motlicek, and Ina Kodrasi. Test-time compute scaling for asr with depth-conditioned looped transformers. *arXiv preprint arXiv:2606.04678*, 2026.

Felix Koch, Marcel Wever, Fabian Raisch, and Benjamin Tischler. State-space models for tabular prior-data fitted networks. *arXiv preprint arXiv:2510.14573*, 2025.

Mieszko Komisarczyk, Saurabh Mathur, Maurice Kraus, Sriraam Natarajan, and Kristian Kersting. Tydra: An efficient hybrid model for tabular data. *arXiv preprint arXiv:2608.21199*, 2026.

Mykhailo Koshil, Matthias Feurer, and Katharina Eggensperger. In-context learning of soft nearest neighbor classifiers for intelligible tabular machine learning. In *Proceedings of the 4th Table Representation Learning Workshop*, pp. 182–191, 2025.

Mykhailo Koshil, Matthias Feurer, and Katharina Eggensperger. Tacticl: Task-aware compression of tabular icl models. *arXiv preprint arXiv:2608.10837*, 2026.

Jannik Kossen, Neil Band, Clare Lyle, Aidan N Gomez, Thomas Rainforth, and Yarin Gal. Self-attention between datapoints: Going beyond individual input-output pairs in deep learning. *Advances in Neural Information Processing Systems*, 34:28742–28756, 2021.

J. Küken, L. Purucker, and F. Hutter. Early stopping tabular in-context learning. In *1st International Workshop on Foundation Models for Structured Data (FMSD) @ ICML 2025*, 2025.

Jonas Landsgesell, Pascal Knoll, and Tizian Wenzel. Scoringbench: A benchmark for evaluating tabular foundation models with proper scoring rules. *arXiv preprint arXiv:2603.29928*, 2026.

Si-Yang Liu and Han-Jia Ye. TabSwift: An efficient tabular foundation model with row-wise attention. In *Forty-third International Conference on Machine Learning*, Proceedings of Machine Learning Research. PMLR, 2026.

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.

D. McElfresh, S. Khandagale, J. Valverde, V. Prasad C, G. Ramakrishnan, M. Goldblum, and C. White. When do neural nets outperform boosted trees on tabular data? In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Proceedings of the 36th International Conference on Advances in Neural Information Processing Systems (NeurIPS'23)*, pp. 76336–76369. Curran Associates, 2023.

John Xavier Morris, Chawin Sitawarin, Chuan Guo, Narine Kokhlikyan, G Edward Suh, Alexander M Rush, Kamalika Chaudhuri, and Saeed Mahloujifar. How much can language models memorize? In *Forty-third International Conference on Machine Learning*.

D. Mostafa, G. Stephan, V. Oriol, J. Uszkoreit, and L. Kaiser. Universal Transformers. In *The Seventh International Conference on Learning Representations (ICLR'19)*. ICLR, 2019.

Sajad Movahedi, Vera Milovanović, Shlomo Libo Feigin, Alexander Theus, Thomas Hofmann, Valentina Boeva, T Konstantin Rusch, and Antonio Orvieto. Fixed-point reasoners: Stable and adaptive deep looped transformers. *arXiv preprint arXiv:2606.18206*, 2026.

S. Müller, N. Hollmann, S. Arango, J. Grabocka, and F. Hutter. Transformers can do Bayesian inference. In *The Tenth International Conference on Learning Representations (ICLR'22)*. ICLR, 2022.

James O' Neill and Fergal Reid. Looped latent attention: Cross-loop kv compression for looped transformers. *arXiv preprint arXiv:2607.15456*, 2026.

Kishan Padayachy, Ronald Richman, and Mario V Wüthrich. Tab-trm: Tiny recursive model for insurance pricing on tabular data. *arXiv preprint arXiv:2601.07675*, 2026.

Pascal Pfeiffer, Dmitry Gordeev, Mathias Müller, Laura Fink, Joan Salvà Soler, Mark Landry, Branden Murray, Marcos V Conde, and Sri Satish Ambati. Tabh2o: A unified foundation model for tabular prediction. *arXiv preprint arXiv:2605.18383*, 2026.

J. Qu, D. Holzmüller, G. Varoquaux, and M. Le Morvan. TabICL: A tabular foundation model for in-context learning on large data. In *Proceedings of the 41st International Conference on Machine Learning (ICML'24)*, volume 251 of *Proceedings of Machine Learning Research*. PMLR, 2024.

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.

Google Research. Tabfm: A zero-shot foundation model for tabular data. 2026. URL https://research.google/blog/introducing-tabfm-a-zero-shot-foundation-model-for-tabular-data/.

Nikunj Saunshi, Nishanth Dikkala, Zhiyuan Li, Sanjiv Kumar, and Sashank J Reddi. Reasoning with latent thoughts: On the power of looped transformers. In *International Conference on Learning Representations*, volume 2025, pp. 14855–14881, 2025.

Avi Schwarzschild, Eitan Borgnia, Arjun Gupta, Furong Huang, Uzi Vishkin, Micah Goldblum, and Tom Goldstein. Can you learn an algorithm? generalizing from easy to hard problems with recurrent networks. *Advances in Neural Information Processing Systems*, 34:6695–6706, 2021.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

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.

Victor Conchello Vendrell, Arnau Padres Masdemont, Niccolò Grillo, Jordi Ros-Giralt, Arash Behboodi, and Fabio Valerio Massoli. Memory-efficient looped transformer: Decoupling compute from memory in looped language models. *arXiv preprint arXiv:2605.07721*, 2026.

Shaowen Wang, Bingrui Li, Ge Zhang, Wenhao Huang, Shen Yan, and Jian Li. On the residual scaling of looped transformers: Stability and transferability. *arXiv preprint arXiv:2606.18524*, 2026.

Liu Yang, Kangwook Lee, Robert Nowak, and Dimitris Papailiopoulos. Looped transformers are better at learning learning algorithms. In *International conference on learning representations*, volume 2024, pp. 42195–42214, 2024.

Xiao-Wen Yang, Ziyu Han, Xi-Hua Zhang, Wen-Da Wei, Jie-Jing Shao, Lan-Zhe Guo, and Yu-Feng Li. Stabilizing recurrent dynamics for test-time scalable latent reasoning in looped language models. *arXiv preprint arXiv:2605.26733*, 2026.

Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. Adding conditional control to text-to-image diffusion models. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 3813–3824. IEEE, 2023.

Rui-Jie Zhu, Zixuan Wang, Kai Hua, Tianyu Zhang, Ziniu Li, Haoran Que, Boyi Wei, Zixin Wen, Fan Yin, He Xing, et al. Scaling latent reasoning via looped language models. *arXiv preprint arXiv:2510.25741*, 2025.

A Architecture Details

A.1 Input Encoding

Raw features are standardized per column using training-row statistics, with missing values imputed by the column mean. Following Qu et al. (2026), the model applies *cyclic feature grouping*: each feature is concatenated with its $G-1$ cyclically-shifted neighbors ($G=3$), forming local feature groups projected to E-dimensional cell embeddings, yielding the initial cell stream $\mathbb{R}^{R\times C\times E}$.

Label encoding. Training-row class labels are mapped through a trainable orthogonally-initialized embedding (Grinsztajn et al., 2026) and added to training-row cells.

Distribution conditioning. Three signals characterizing the training distribution are computed once and broadcast across all rows.

- **Marginal histogram.** Adaptive-quantile soft log-histograms (32 bins) with four distributional moments are projected column-wise via a small MLP.
- **Discriminative histogram.** For *classification*, per-class log-histogram residuals (class minus marginal) are projected column-wise.
- **Fourier quantile encoder.** The empirical rank of each cell within its column is Fourier-encoded (8 frequencies) with a binary OOD flag and projected per-cell.

All conditioning MLPs have zero-initialized output layers, making them initially inactive (Bachlechner et al., 2021). This prevents untrained conditioning heads from disrupting the backbone during pretraining and protects pretrained representations during regression continued pretraining (Zhang et al., 2023). The conditioning pathways then gradually activate as their weights move away from zero, allowing the model to learn how strongly to rely on each signal.

A.2 Recurrent Block

Each recurrent iteration applies a single *shared* axial block with weights tied across all L loops. *Axial* here means the block alternates attention over rows (within each column) and attention over

columns (within each row), avoiding the quadratic cost of full joint attention over all cells. For the *cell stream* ($\mathbb{R}^{R\times C\times E}$):

1. **Label re-injection.** Label embeddings are added to training-row cells via a zero-initialized projection.
2. **Within-column attention.** An Induced Set Attention Block (ISAB) (Qu et al., 2024) processes each column independently over rows using $K{=}128$ learned inducing points ($H{=}8$, head dim 16).
3. **Cross-column attention.** Self-attention with RoPE (Su et al., 2024) processes each row over columns ($H{=}8$, head dim 16).

A **readout** via cross-attention then projects cells into the *row stream* ($\mathbb{R}^{R\times D}$) using $C_\ell{=}4$ learned CLS tokens per row. The row stream is updated by a zero-initialized label injection followed by causally-masked ICL attention ($H{=}8$, head dim 64), allowing test rows to attend to training rows. All sub-blocks use sandwich normalization (pre- and post-RMSNorm) (Ding et al., 2021).

Each attention block includes a query-conditioned softmax scaling layer (Qu et al., 2026) that rescales queries before the dot product:

$$q \;\leftarrow\; q\cdot\gamma\cdot\delta, \qquad \gamma = 1 + f\left(\log\tfrac{n}{n_{\text{ref}}}\right), \qquad \delta = 1+\tanh(g(q))\,,$$

where γ is a per-head temperature correction, δ is a per-query modulation, and f, g are small zero-initialized MLPs. We introduce $n_{\text{ref}}{=}512$ to prevent unbounded output growth that can cause instability at longer sequence lengths (Pfeiffer et al., 2026).

A.3 Output Head

Inspired by Grinsztajn et al. (2026), an attention-based retrieval decoder (Arbel et al., 2026) accumulates per-class attention mass between test-row queries and training-row keys via one-hot label weighting (Koshil et al., 2025), averaged across heads to produce class log-probabilities.

A.4 Early-Exit Gate

After each recurrent iteration t, a lightweight gate decides whether to stop or continue. If it exits at step t, the decoder runs once at that step; earlier iterations only update the shared representations. The gate has fewer than 20K (19 521) parameters and runs on pooled row embeddings, so its overhead is negligible.

Gate inputs. The gate takes a 43-dimensional feature vector $\mathbf{f}_t$ formed by concatenating five signals:

Signal	Description	Dim
$\mathbf{c}_t$	Compressed mean state	8
$\Delta_t = \mathbf{c}_t - \mathbf{c}_{t-1}$	Change since last loop	8
$\text{pe}(t)$	Sinusoidal loop-depth encoding	16
$[\lambda_{t-1}, \lambda_{t-2}, \lambda_{t-3}]$	Last three gate outputs	3
$\text{pe}(\log N)$	Sinusoidal dataset-size encoding	8
Total		43

The compressed state $\mathbf{c}_t \in \mathbb{R}^8$ is computed by passing each row embedding through a small MLP ($512 \to 32 \to 8$, ReLU, no bias) and averaging over rows. The change signal Δ_t is near zero when representations have stabilised, providing a natural convergence cue.

Decision network. A two-layer MLP ($43 \to 64 \to 1$) with sigmoid output maps $\mathbf{f}_t$ to an exit probability:

$$\lambda_t = \sigma(\text{MLP}_\phi(\mathbf{f}_t)) \in (0,1).$$

The output layer is zero-initialised, so $\lambda_1 \approx 0.5$ at the start of training.

Inference-time exit rule. We accumulate a running exit probability

$$F_t = 1 - \prod_{s=1}^{t}(1 - \lambda_s)$$

and stop at the first loop t where $F_t > q$ (default $q = 0.5$), following Zhu et al. (2025). A per-sample variant exits each example independently once its $F_t > q$.

Training. The backbone is frozen; only the gate is trained. The oracle label

$$w_t = \sigma\big(50\,(I_t - 0.005)\big), \qquad I_t = \mathcal{L}_t - \min_{s>t} \mathcal{L}_s,$$

encodes whether continuing past loop t reduces the loss. We train with weighted binary cross-entropy, upweighting steps where early exit causes large regret ($r_t = (\mathcal{L}_t - \min_s \mathcal{L}_s)^+$):

$$\mathcal{L}_{\text{gate}} = \frac{1}{T}\left[\sum_{t=1}^{T-1}(1 + r_t)\,\text{BCE}(\lambda_t,\, 1 - w_t) \;+\; \text{BCE}(\lambda_T,\, 1)\right].$$

To discourage late exits we add a gap penalty $\mathcal{L}_{\text{gap}} = 0.02\,\mathbb{E}[\tilde{g}(\bar{t} - t^*)^2]$, where $\tilde{g}(\delta) = 2\delta$ for $\delta > 0$ and δ otherwise (late exits penalised twice as heavily). The total loss is $\mathcal{L} = \mathcal{L}_{\text{gate}} + \mathcal{L}_{\text{gap}}$.

A.5 Parameter Counts

Table 1 lists per-component parameter counts. In total, the model has $\sim$3.25M parameters with $\sim$2.7M parameters in the recurrent block.

Table 1: Parameter counts for *LoopICL* (E=128, D=512).

Component	**Parameters**
Input encoding	
Cell embedding	512
Label encoder	1,280
Log-histogram conditioner	14,848
Discriminative histogram conditioner	14,592
Fourier quantile encoder	9,472
Init readout	131,712
Shared axial block (tied across L loops)	
Within-column attention (ISAB)	290,384
Cross-column attention	142,160
ICL transformer block	2,140,928
Auxiliary (label inject, CLS, norms)	83,456
Output	
Output norm	512
Retrieval decoder	427,392
Total	**3,257,248**

B Inference Details

Preprocessing. *LoopICL* applies a fixed preprocessing pipeline fitted on training rows and reapplied consistently to test rows:

1. **Feature encoding.** Categorical columns are ordinal-encoded (unknown test categories $\to -1$); numeric missing values are mean-imputed from training statistics.
2. **Constant feature removal.** Features with only one unique training value are dropped.
3. **Z-score standardization.** All features are standardized and clipped to $[-100, 100]$.
4. **Outlier clipping.** A two-stage z-score clipper (threshold 4.0) first identifies and removes outliers, re-estimates statistics on the cleaned data, then clips remaining values to $\pm 4\hat{\sigma}$ of the cleaned distribution.

Ensemble inference. At test time, *LoopICL* runs $N{=}8$ ensemble members and aggregates their predictions. Each member receives a differently transformed view of the data, varied along three axes:

- **Feature order.** Columns are permuted using Latin square patterns (Qu et al., 2026), ensuring each feature appears in a distinct input position across members with no two members sharing the same ordering.
- **Class permutation (classification).** Training labels are shuffled using a balanced permutation strategy (Grinsztajn et al., 2026): all unique class orderings are enumerated, deduplicated, and distributed evenly across members, removing positional bias towards specific class indices.
- **Preprocessing.** Members cycle through four distribution normalizations: *none*, Yeo-Johnson, quantile-to-$\mathcal{N}(0,1)$, and robust IQR scaling, to increase diversity.

A fingerprint feature—a per-row hash of the training context—is appended to each member's feature matrix (Grinsztajn et al., 2026), providing a unique row identifier per member. For *classification*, each member's logits are converted to probabilities via softmax and the N probability vectors are averaged.

Many-class support. The model natively supports up to $T{=}10$ classes. For datasets with more than 10 classes, we apply a mixed-radix decomposition (Qu et al., 2026): each class c is encoded as a tuple of digits $(d_1, \ldots, d_L)$ in a mixed base, where each digit position $d_\ell \in \{0, \ldots, b_\ell{-}1\}$ defines a b_ℓ-class sub-problem the model can solve natively. Each digit position is run independently through the full ensemble, producing per-digit class probabilities. Final class probabilities are obtained by multiplying the per-digit probabilities: $P(c) \propto \prod_\ell P(d_\ell{=}\mathrm{digit}_\ell(c))$.

C Pretraining Details

Recurrent loop schedule. At each training step, the number of recurrent iterations L is sampled from a log-normal Poisson distribution (Geiping et al., 2026): $\gamma \sim \mathrm{LogNormal}(\mu, \sigma^2)$ with $\sigma = 0.5$, then $L \sim \mathrm{Poisson}(\gamma)$, clipped to $[1, 8]$. The mean μ is chosen so that $\mathbb{E}[L] = 6$. This distribution concentrates mass near the default depth while maintaining a long tail, enabling generalization to more iterations at inference time (Schwarzschild et al., 2021). During training L is sampled from a log-normal Poisson distribution ($\mu{=}5$, $\sigma{=}0.5$, max 8); at inference $L{=}6$ with loop-residual scaling $\alpha{=}\gamma/(L\sqrt{S})$.

Stage 1: Pretraining. We train for 500,000 steps with a batch size of 64 on synthetic datasets generated by the *TabICLv2* graph-based SCM prior (Qu et al., 2026). Each dataset contains 2–100 features, up to 10 classes, and up to 1,024 rows, with train/test split ratios uniformly sampled in $[0.3, 0.9]$. Full backpropagation through all recurrent iterations is used. We use the Muon optimizer (Jordan et al., 2024) ($\mathrm{lr} = 8 \times 10^{-4}$, momentum $= 0.9$) with an AdamW fallback ($\mathrm{lr} = 3 \times 10^{-4}$, $\beta = (0.9, 0.95)$) for scalar and embedding parameters. A warmup-stable-decay (WSD) schedule applies a 5,000-step warmup followed by cosine decay over the final 10% of training. We use gradient clipping (norm $= 1.0$), weight decay $= 0.01$, an exponential moving average (EMA) with decay $= 0.999$, and `bfloat16` mixed precision.

Stage 2: Long-context fine-tuning. Starting from the Stage 1 checkpoint, we fine-tune for 40,000 steps on a larger-scale prior with 1–2,000 features and up to 100,000 rows (subject to a feature$\times$row budget of 10^6), enabling the model to handle datasets far beyond its Stage 1 training distribution. To make this computationally feasible, we use TBPTT (Geng et al., 2021) with a window of $k = 5$: gradients are propagated only through the last 5 recurrent iterations, and earlier activations are detached. This reduces memory proportionally to k/L, allowing inference-time depth to far exceed training-time depth (Geiping et al., 2026). We use a reduced learning rate of 10^{-4} with 400 warmup steps; all other optimizer settings are inherited from Stage 1.

Training compute. We train *LoopICL* on B300 GPUs. Stage 1 takes 18 GPU-days, Stage 2 takes 10 GPU-days, and training the early-exit gate takes 1 GPU-day.

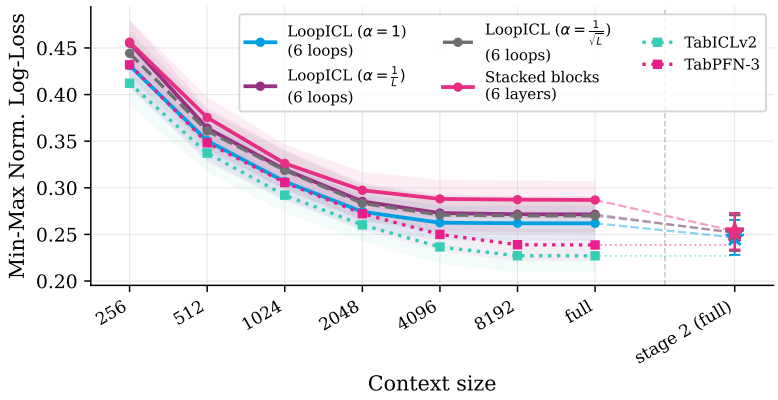

Figure 9: **Length generalization.** Recurrent models demonstrate stronger length generalization compared to the stacked model.

D Experiment Details

D.1 Length Generalization

We also compare the ability of our models to scale with an increasing number of in-context examples. As shown in Figure 9, the recurrent models scale slightly better than the stacked model, with the performance gap between the stacked (—) and recurrent models (—, - -, - -) widening as the context size increases. This effect is particularly pronounced for the recurrent model with $\alpha = 1/L$, consistent with the findings of (Fan et al., 2025), who show that looped Transformers with an adaptive number of steps can substantially improve length generalization.

We further evaluate the models after long-context training in Stage 2. As shown in Figure 9, all models benefit from long-context training. Notably, after this additional training, the stack model nearly matches the performance of the recurrent models, substantially narrowing the gap observed with shorter-context training.

D.2 Representation Analysis

We analyze how embeddings evolve across loop iterations using three complementary measures: embeddings similarity, cross-step probing classifiers (logistic regression trained on step i, tested on step j), and class separation gap, following the approach of Balef et al. (2026).

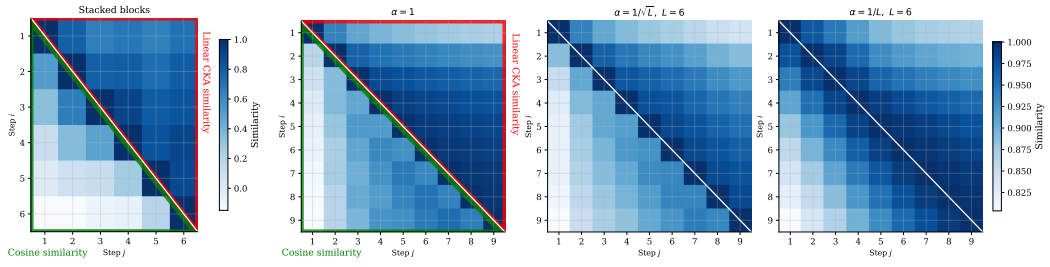

Figure 10: Cross-step embedding similarity for stacked blocks (*left*) and looped models (*right*). Each heatmap entry (i, j) shows the cosine similarity (lower triangle) or linear CKA (upper triangle) between embeddings at loop steps i and j, averaged over all binary-classification datasets.

Embeddings similarity. As shown in Figure 10, stacked blocks exhibit low off-diagonal similarity, indicating that each block produces a qualitatively distinct representation. In contrast, looped models maintain high similarity across step pairs, reflecting the representational stability induced by weight tying.

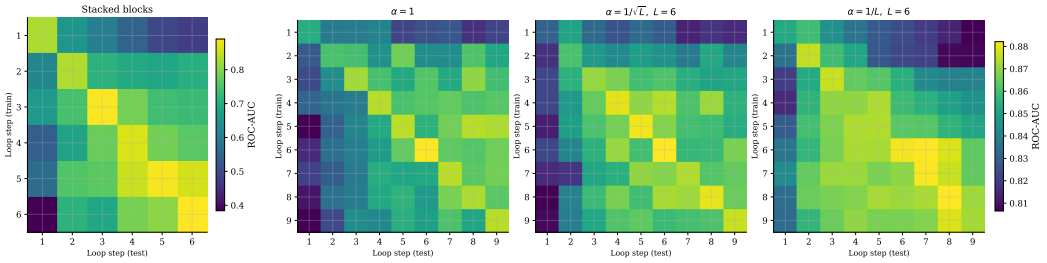

Figure 11: Cross-step probing AUC for stacked blocks (*left*) and looped models (*right*). Each entry (i, j) shows the ROC-AUC of a logistic regression trained on embeddings at step i and evaluated on embeddings at step j, averaged over all binary-classification datasets.

Probing classifiers. As shown in Figure 11, stacked blocks exhibit near-chance off-diagonal AUC, indicating limited transferability of representations across blocks. In contrast, looped models retain high off-diagonal AUC, showing that a classifier trained at one loop step generalizes well to the others. More interestingly, among looped models, the *asymmetry* between the upper ($j > i$, train early/test late) and lower ($j < i$, train late/test early) triangles reveals distinct convergence regimes. For $\alpha = 1$, the upper triangle dominates: early-step features are largely preserved at later iterations, while later steps introduce additional discriminative structure that early classifiers cannot capture. For $\alpha = \frac{1}{L}$, this pattern reverses, becoming particularly pronounced beyond loop step L. Classifiers trained after step L continue to transfer to earlier steps, whereas classifiers trained at earlier steps fail to transfer beyond step L. This asymmetry suggests that the scaled residual updates progressively overwrite features established during the first L loops, providing a possible explanation for the degradation observed when models trained with a fixed L are evaluated for more than L iterations at test time.

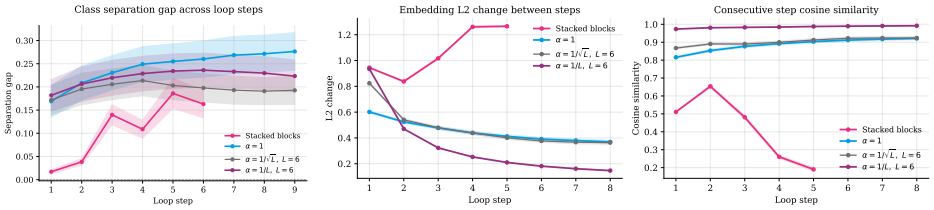

Figure 12: **Left**: class separation gap (mean inter-class minus intra-class cosine distance) at each loop step. **Right**: embedding update magnitude across iterations, measured as the normalised ℓ_2 change between consecutive steps (*left panel*) and as the cosine similarity between consecutive embeddings (*right panel*).

Separation gap. Figure 12 shows that stacked blocks begin with a near-zero separation gap, indicating that the early layers primarily perform *latent mapping* before progressively building class structure across subsequent layers. In contrast, looped models already exhibit a substantial separation gap at the first iteration, suggesting that the subsequent iterations primarily perform *feature engineering*. For $\alpha = 1$, the separation gap grows monotonically across all nine steps ($0.169 \rightarrow 0.276$). For $\alpha = \frac{1}{\sqrt{L}}$, the gap plateaus around step 4, while for $\alpha = \frac{1}{L}$ it peaks at step $L = 6$ before marginally declining. The latter behavior is consistent with the representational overwriting observed in Figure 11.

Residual Scaling. Interestingly, as shown in Figure 13, with residual scaling $\alpha = 1/L$, the optimal number of loops for a fixed residual depth ($L = 16$) exhibits a clear correlation with dataset size. In contrast, we do not observe the same relationship without residual scaling or with $\alpha = 1/\sqrt{L}$. This suggests that residual scaling by $1/L$ may help the model adapt its effective computation depth to the size of the dataset.

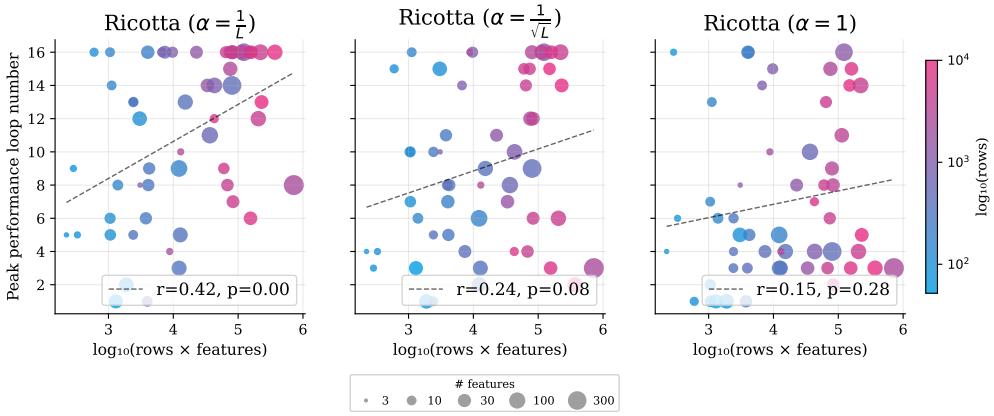

Figure 13: With residual scaling $\frac{1}{L}$ we see a correlation between the optimal number of loops and dataset size.

D.3 PCA ANALYSIS

To characterize how information is organized within the iterative latent space, we apply PCA to the test-row embeddings at each loop step. For each dataset, embeddings are averaged across repeats within a fold and stacked into an $SN \times D$ matrix (loop steps $\times$ test samples), then a PCA is fit retaining the top 10 principal components.

We score each PC k against four candidate roles:

- **Time-step (t):** absolute Pearson correlation between PC scores and the loop-step index,

$$A_k^{(t)} = |\rho(z_k,\, t)|\,.$$

- **Class separation (y):** eta-squared measuring how much variance in PC scores is explained by the binary class label,

$$A_k^{(y)} = \eta^2(z_k,\, y) = \frac{\sum_{c\in\{0,1\}} n_c \left(\bar{z}_{k,c} - \bar{z}_k\right)^2}{\sum_j \left(z_{k,j} - \bar{z}_k\right)^2},$$

  where n_c and $\bar{z}_{k,c}$ are the count and mean PC score for class c, and $\bar{z}_k$ is the overall mean.

- **Confidence (H):** absolute Pearson correlation between PC scores and per-sample prediction entropy,

$$A_k^{(H)} = |\rho(z_k,\, H)|\,, \qquad H_i = -\sum_{c=1}^{C} p_{ic} \log p_{ic},$$

  averaged across repeats and loop steps.

- **Sample identity (i):** eta-squared measuring how much variance is explained by sample membership,

$$A_k^{(i)} = \eta^2(z_k,\, i) = \frac{S \sum_{r=0}^{N-1} \left(\bar{z}_{k,r} - \bar{z}_k\right)^2}{\sum_j \left(z_{k,j} - \bar{z}_k\right)^2},$$

  where $\bar{z}_{k,r}$ is the mean PC score for sample r across its S loop steps.

For each role, the 10 raw scores are L1-normalized to form a probability distribution $p_r \in \Delta^9$ over PC indices, indicating the relative affinity of each PC for that role. The four distributions are computed independently, so a single PC may carry signal for multiple roles.

Within each cross-validation fold, the per-dataset probability vectors are averaged across datasets to yield one vector per role per fold. We report the mean and 95% confidence interval ($\pm 1.96 \times$ SEM) across the three folds (Figure 14).

As shown in Figure 14, in recurrent model variants the time-step role is strongly concentrated in the leading PC, followed by class-separation signal in the next principal components — indicating that the model organizes its iterative dynamics along a small number of structured axes. Interestingly, in the stacked-blocks variant the contributions of all four roles are more diffuse, spreading across higher-order components rather than concentrating in the first few.

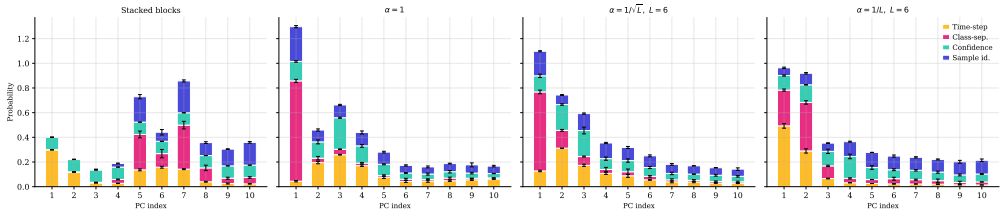

Figure 14: **Structure of information in the iterative latent space.** We apply PCA to query embeddings pooled across loop steps and samples, and measure the affinity of each principal component (PC) with four functional roles: time-step progression, class separation, prediction confidence, and sample identity.

D.4 Performance

Here we provide more details on the performance on *TabArena* and *TALENT*. For *TabArena*, we report Elo ratings (Figure 15), a critical difference diagram (Figure 16), and pairwise win rates among the top-20 models (Figure 17). For *TALENT*, we report the corresponding Elo ratings (Figure 18), critical difference diagram (Figure 19), and pairwise win rates (Figure 20).

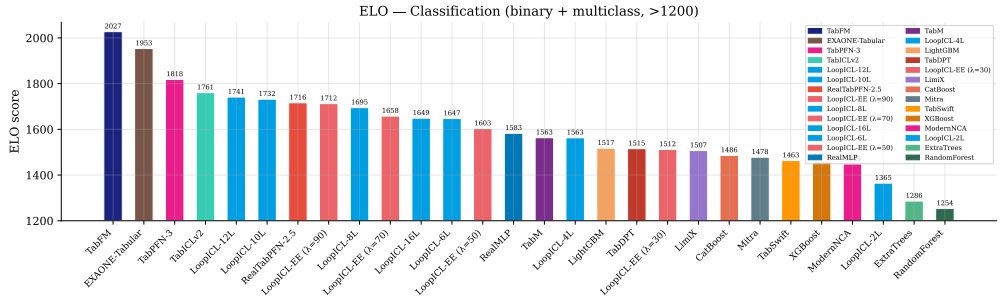

Figure 15: Elo ratings for classification on *TabArena*. Higher is better.

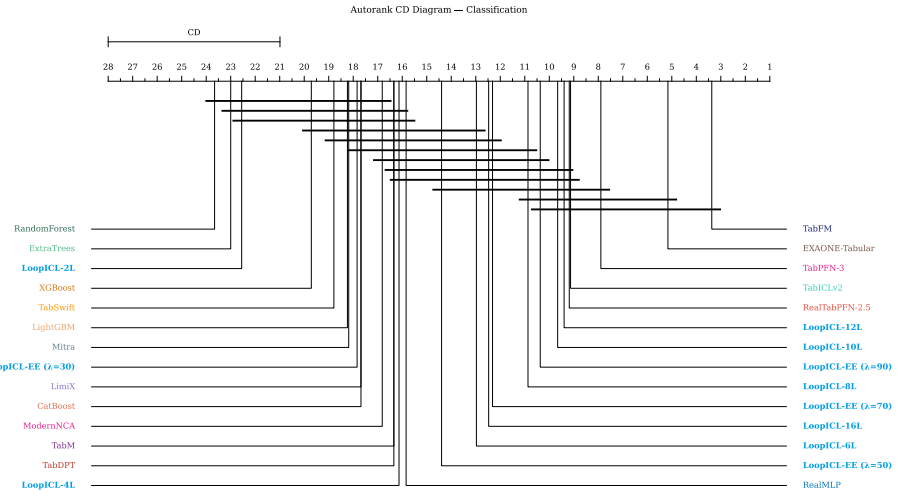

Figure 16: Critical difference diagram for classification on *TabArena*, computed via the `autorank` framework (Herbold, 2020) using a Wilcoxon signed-rank test with Holm correction ($\alpha = 0.05$). Models connected by a horizontal bar are not significantly different.

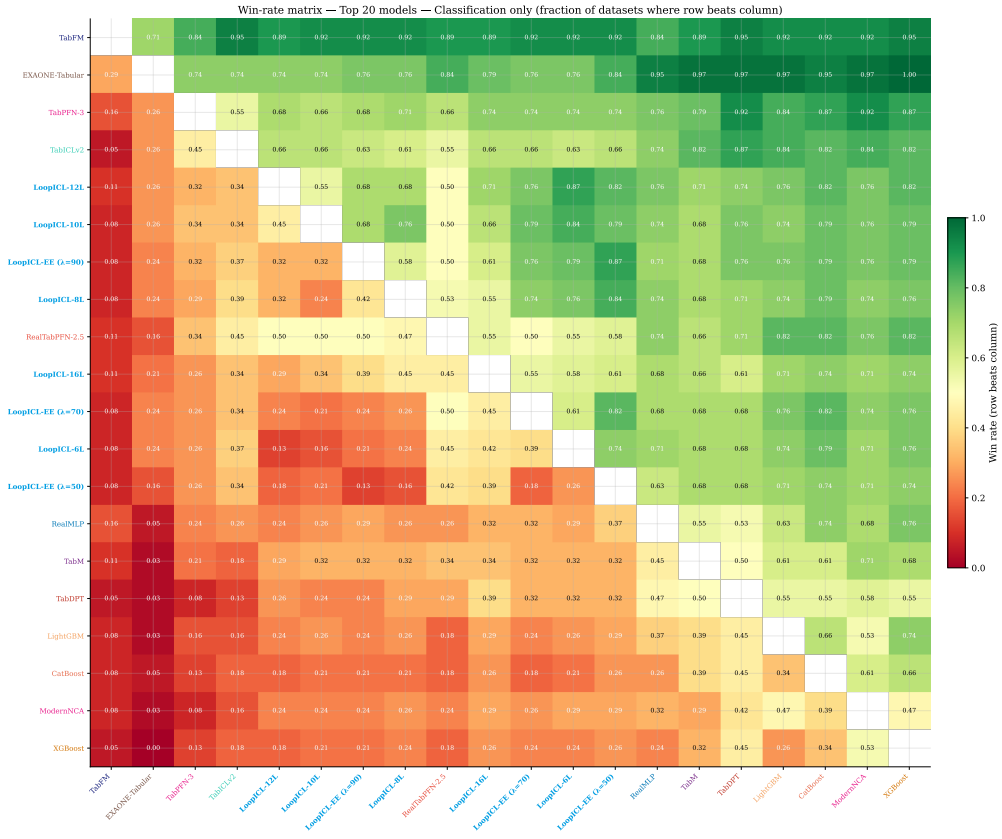

Figure 17: Pairwise win rates on *TabArena* classification (top-20 models). Each cell shows the fraction of datasets where the row model outperforms the column model.

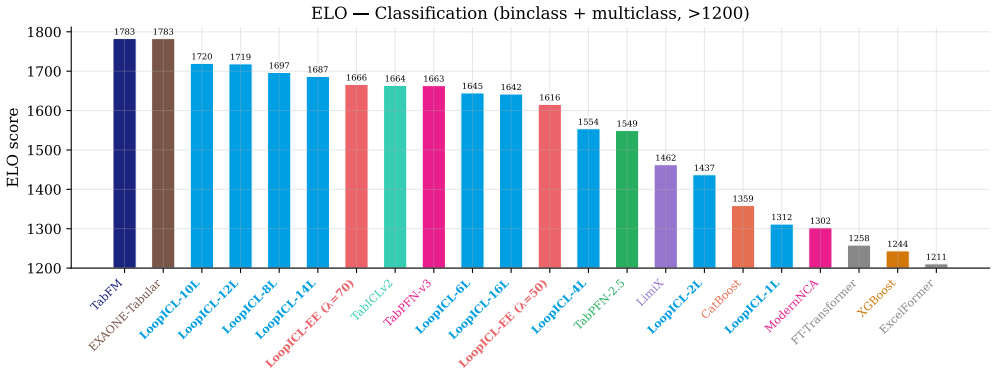

Figure 18: Elo ratings for classification on *TALENT*. Higher is better.

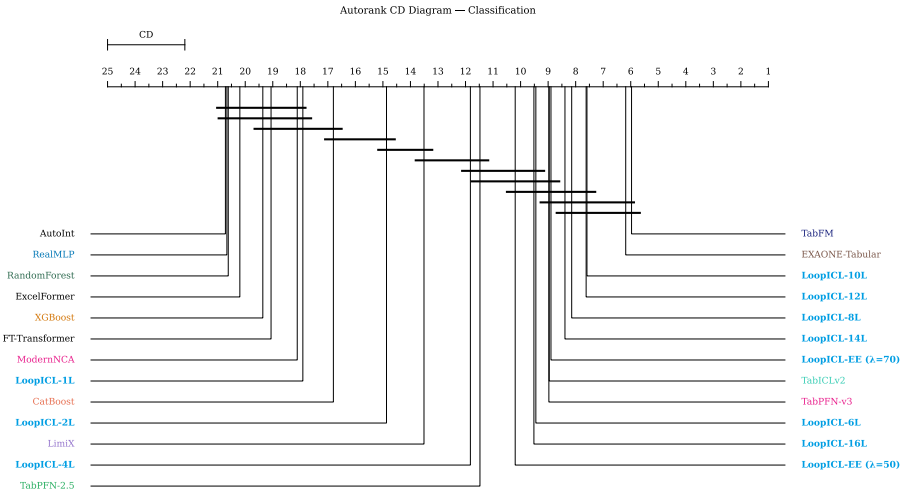

Figure 19: Critical difference diagram for classification on *TALENT*, computed via the `autorank` framework (Herbold, 2020) using a Wilcoxon signed-rank test with Holm correction ($\alpha = 0.05$). Models connected by a horizontal bar are not significantly different.

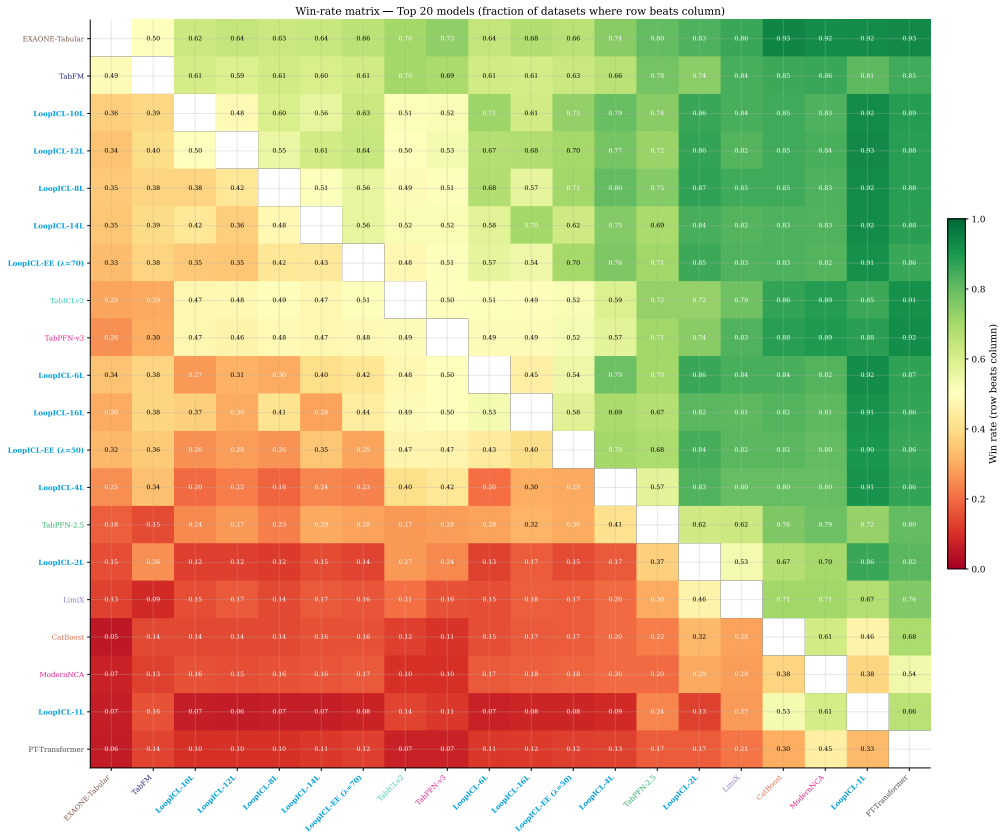

Figure 20: Pairwise win rates on *TALENT* classification (top-20 models). Each cell shows the fraction of datasets where the row model outperforms the column model.