

TAB-LOOP: RECURRENT IN-CONTEXT LEARNING FOR TABULAR FOUNDATION MODELS

Tejas Vaidhya^{1,2}, Gaurav Sahu^{1,3}, & Irina Rish^{1,2}

¹Mila – Quebec AI Institute

²Nolano AI

³Unified Sciences

{iamtejasvaidhya, sahu.gaurav719, irina.rish}@gmail.com

ABSTRACT

Tabular foundation models adapt to new datasets through in-context learning (ICL), and, like most transformers, they tie computational depth to the number of independently parameterized layers. We ask whether the ICL stage of such a model needs a distinct set of parameters for every layer, or only depth. We introduce Tab-Loop, which replaces the 12 untied ICL blocks of TabICLv2 with 4 tied blocks applied k times and changes nothing else: the encoders, decoder, synthetic prior and training recipe are shared with an untied baseline trained for the same compute. At matched pretraining compute (18.2 EFLOP) and matched inference compute (12 block applications), Tab-Loop matches the baseline with 62% fewer parameters (10.4M vs. 27.6M): held-out log loss is 0.4583 vs. 0.4588, mean accuracy on 200 TALENT datasets is 84.10% vs. 83.92%, and Elo on 38 TabArena datasets is 1409 vs. 1396. A 4-block untied control with Tab-Loop’s parameter count, which sees 32% more training tables for the same compute, trails both (83.46%; Elo 1343). Because weights are shared, depth becomes a choice made at inference. Accuracy rises with the number of loops through the trained range and holds beyond it, and continued training with deeper loops extends the useful range, reaching the baseline’s stage-2 scores at $4\text{--}5\times$ its inference compute. At this scale, the ICL stage of a tabular foundation model needs depth more than it needs distinct parameters.

1 INTRODUCTION

Foundation models have changed how machine learning is practised, replacing task-specific training with a single pretrained model that adapts to new tasks at inference. After language and vision, this shift has reached tabular data, the modality behind applications such as credit scoring, fraud detection, and demand forecasting. Tabular data long resisted deep learning, and gradient-boosted trees remained the method to beat (Grinsztajn et al., 2022; McElfresh et al., 2023). Tabular foundation models such as TabPFN (Hollmann et al., 2023; 2025), TabICL (Qu et al., 2025; 2026), Mitra (Zhang et al., 2025b), TabDPT (Ma et al., 2025), LimiX (Zhang et al., 2025a), TabPFN-3 (Grinsztajn et al., 2026) and EXAONE Tabular (Eo et al., 2026) have changed this picture. Pretrained on synthetic or real tables, they predict labels for a new table in a single forward pass by conditioning on its labelled rows, without per-dataset training or tuning, and on small and medium-sized tables they now match or outperform tuned tree ensembles (Hollmann et al., 2025; Erickson et al., 2025).

However, because adaptation happens at inference, every cost of these models is paid at every prediction, and this cost grows with model size and context length (Kong et al., 2026). Moreover, these models inherit a design choice from standard transformers: every layer has its own parameters, so depth and parameter count are the same resource. Recent work suggests that they need not be (see Section 2). Tabular foundation models refine their representations iteratively across layers, and much of their depth is redundant (Balef et al., 2026). Looped transformers, which reapply shared blocks, match deeper untied models on algorithmic and in-context learning tasks with far fewer parameters (Yang et al., 2024; Saunshi et al., 2025).

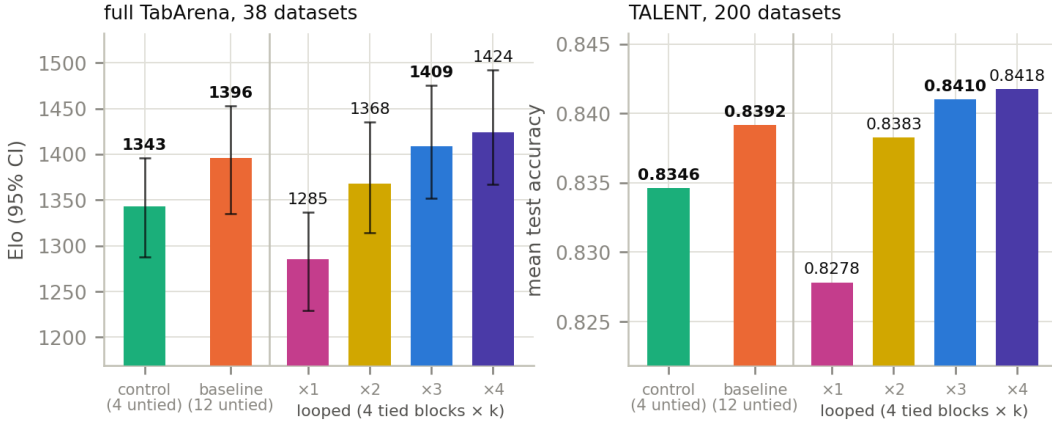


Figure 1: **Depth without parameters.** End-of-training results for three models trained with the same recipe for the same compute (18.2 EFLOP): the untied 12-block baseline (27.6M parameters), a 4-block untied control (10.4M), and Tab-Loop (4 tied blocks, 10.4M) read at k loops, i.e. $4k$ block applications. **Left:** Elo on full TabArena (38 classification datasets, 594 splits) with 95% bootstrap intervals. **Right:** mean test accuracy on TALENT (200 classification datasets). At $k = 3$, which matches the baseline’s 12 block applications, Tab-Loop matches the baseline with 62% fewer parameters. The control, with the same parameters but one third of the depth, trails both.

This raises a crucial unanswered question: *does the ICL stage of a tabular foundation model need a distinct set of parameters for every layer, or only depth?* Answering it requires a controlled comparison, in which the looped and untied models share everything except the ICL stage and, *critically*, are compared at equal compute, both in training and at inference.

To address this, we introduce Tab-Loop, which builds on TabICLv2 (Qu et al., 2026) and replaces its ICL stage with a recurrent one. Rather than proposing a new architecture family, we use Tab-Loop as a controlled probe of where the capacity of a tabular foundation model lives. Our contributions are as follows.

A controlled test of recurrence in tabular foundation model. Tab-Loop, loops 4 tied blocks in place of the 12 untied ICL blocks of TabICLv2, with parameter-free input re-injection and training over sampled loop counts, and leaves the encoders, decoder, prior and curriculum unchanged. We compare it with the untied baseline at matched, measured training compute, and with a 4-block untied control that has Tab-Loop’s parameters and spends the same compute on 32% more training tables. Unlike earlier proof-of-concept studies, this design separates weight sharing from depth, and depth from additional data (Sections 3 and 4).

The ICL stage relies on depth more than on distinct parameters. At equal training and inference compute, Tab-Loop performs comparably to the baseline with 62% fewer parameters, reaching 84.10% against 83.92% mean accuracy on 200 TALENT datasets and an Elo of 1409 against 1396 on TabArena. The control trails both despite its additional data. Moreover, read at a single loop, Tab-Loop is weaker than the control, which shows that its computation is distributed across loops rather than stacked on a complete single pass (Section 5.1).

Inference-time depth scaling. Without retraining, TabArena Elo rises from 1288 at one loop to 1424 at four and 1427 at five, and TALENT accuracy peaks at 84.18% at four loops, the only depth with a significant gain over the baseline. Continued training with deeper loops moves the useful range outward. On TabArena-Lite, Elo climbs from 1298 at three loops to 1512 at twelve and to about 1548 at 24 to 32 loops, while TALENT accuracy reaches 84.42% at sixteen loops, matching the stage-2 baseline at 4 to 5 times its inference compute (Sections 5.2 and 5.3).

2 WHY RECURRENT DEPTH FOR TABULAR IN-CONTEXT LEARNING?

Recurrent depth lets a tabular foundation model apply its in-context learner as many times as a table requires, while storing the weights of that learner only once. Tabular foundation models have instead

followed the recipe of language modelling, which is to scale: TabPFN-3 is roughly five times larger than TabPFN-2.5 (Grinsztajn et al., 2026), and published models now span more than two orders of magnitude (Table 1). Because these models adapt at inference, this growth is paid at every prediction. Compared with adding untied layers, recurrent depth has several advantages.

- **Tabular in-context learning is iterative and draws on the context rather than on stored knowledge.** Gradient boosting refines a prediction by fitting each new learner to the residuals of the last (Friedman, 2001), and transformers learn in context in a similar way, with successive layers implementing steps of an iterative learner (von Oswald et al., 2023; Müller et al., 2024). Parameters, in contrast, are where a model stores knowledge (Morris et al., 2026), whereas a tabular foundation model answers from the labelled rows in front of it. Looped transformers already match deeper untied models on in-context learning tasks with a fraction of their parameters (Yang et al., 2024; Saunshi et al., 2025).
- **Existing tabular foundation models are redundant in depth.** Their layers refine representations progressively, and much of their depth can be removed (Balef et al., 2026; Koshil et al., 2026). However, removing layers reduces parameters and depth together, so redundancy alone cannot say which of the two the model needs. Sharing weights reduces parameters while preserving depth, and is therefore the intervention that separates them.
- **Recurrent depth decouples memory from computation.** A tied model stores its blocks once, so Tab-Loop applies 12 blocks with 62% fewer parameters than the untied baseline. Tying alone does not reduce computation, since a shared block costs as much as an untied one, but it makes computation the only cost that grows with depth.
- **Depth becomes a choice at inference.** An untied stack cannot be shortened after training, because its decoder only ever reads the final layer. A recurrent model with one decoder trained across depths can be read after any number of loops, and recurrent networks are known to solve harder problems by iterating longer (Schwarzschild et al., 2021; Geiping et al., 2025). Computation can then follow the difficulty of the dataset rather than the size of the model.

Recurrence also has known failure modes: a shared block must perform every stage of a computation that an untied stack spreads across specialized layers (Blayney et al., 2026; Bae et al., 2025a), its state can drift with depth (Geiping et al., 2025; Movahedi et al., 2026), and information about the input can wash out unless it is re-injected (Yang et al., 2024). Tab-Loop addresses these by looping a group of four blocks rather than a single block, by re-adding the encoded input before every block application, and by confining recurrence to the ICL stage (Section 3). If our hypothesis holds, a tied model should match an untied one at equal compute, outperform a parameter-matched model that

Table 1: **Parameter counts of recent tabular foundation models** (classification variants). Bars are on a logarithmic scale from 1M parameters. Counts are as reported for TabDPT (Ma et al., 2025), for TabPFN-3 and TabPFN-2.5 in the TabPFN-3 report (Grinsztajn et al., 2026), and for EXAONE Tabular and TabFM in the EXAONE Tabular report (Eo et al., 2026); the TabICLv2 count is ours, since our untied baseline uses its architecture, and the LimiX count is the size in the name of its released model. The bottom rows are the three models trained in this work, coloured as in every figure: **orange** for the untied baseline, **blue** for Tab-Loop and **green** for the control.

Model	Parameters	(log scale)
TabFM (Kong et al., 2026)	1.64B	
TabDPT (Ma et al., 2025)	78M	
TabPFN-3 (Grinsztajn et al., 2026)	53M	
TabICLv2 (Qu et al., 2026)	27.6M	
EXAONE Tabular (Eo et al., 2026)	20.81M	
LimiX-16M (Zhang et al., 2025a)	16M	
TabPFN-2.5 (Prior Labs, 2025)	11M	
<i>Trained in this work (same recipe, 18.2 EFLOP each)</i>		
Untied baseline (12 untied blocks)	27.6M	
Tab-Loop (4 tied blocks, k loops)	10.4M	
Untied control (4 untied blocks)	10.4M	

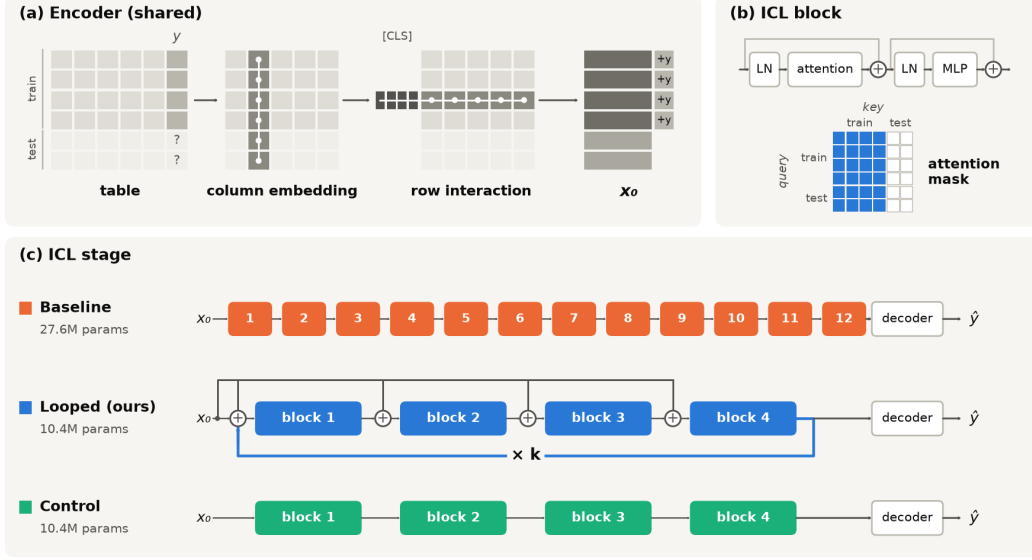


Figure 2: **Models compared.** (a) The column and row encoders are shared by all models and run once per forward pass. (b) Every ICL block is the same pre-norm transformer block; every row attends only to context rows. (c) The baseline applies 12 distinct blocks once (27.6M parameters). Tab-Loop applies 4 tied blocks k times and re-adds the stage input X_0 before every block application after the first (10.4M). The control applies 4 untied blocks once (10.4M).

spends the same compute on more data, and improve with additional loops; Sections 5.1–5.3 test each prediction. More broadly, a model that is heavy in computation and light in parameters sets a prior towards solving a table by iterating on the evidence in its context rather than by recalling what is stored in its weights.

3 TAB-LOOP

Tab-Loop is TabICLv2 (Qu et al., 2026) with one change: the ICL stage reuses 4 blocks k times instead of applying 12 distinct blocks once. Figure 2 shows the three models we compare.

3.1 LOOPED ARCHITECTURE

Setting and input representation. We consider tabular classification with feature vectors $x \in \mathbb{R}^m$ and labels $y \in \{1, \dots, C\}$. Given a labelled context set $D_{\text{train}} = \{(x_i^{\text{train}}, y_i^{\text{train}})\}_{i=1}^{n_{\text{tr}}}$, the model predicts class probabilities for n_{q} unlabelled query rows; together they form a table with $T = n_{\text{tr}} + n_{\text{q}}$ rows. Following TabICLv2, a column encoder builds cell representations within each column, and a row encoder aggregates them across features into one vector per row. Adding learned label embeddings to the context rows yields the ICL input $X_0 \in \mathbb{R}^{T \times d}$. These stages run once per forward pass, and X_0 is reused throughout the recurrent computation. In the ICL stage, every row attends only to context rows, under the same attention mask as the baseline.

Recurrence with shared blocks. The baseline processes X_0 through L independently parameterized transformer blocks:

$$H_{\text{base}} = (B_L \circ \dots \circ B_1)(X_0). \quad (1)$$

Tab-Loop instead uses n blocks $C_1, \dots, C_n$, applied repeatedly in a fixed order. One loop is one application of each of the n blocks. For k loops, the hidden states evolve as

$$\begin{aligned} S_0 &= X_0, \\ S_{i+1} &= C_{1+(i \bmod n)}(S_i + \mathbf{1}_{\{i > 0\}} X_0), \quad i = 0, \dots, nk - 1, \end{aligned} \quad (2)$$

where i indexes single block applications. Re-adding X_0 before every application after the first, including within a loop, keeps the encoded features and context labels available to every update. The injection adds no parameters. Each block keeps the baseline’s attention, feed-forward, normalization and residual structure (Appendix A). Appendix A.3 compares the injection with no injection and with a learned linear carry, which becomes expansive and diverges beyond the trained depths.

Prediction. After k loops, the decoder g_θ reads the query rows of S_{nk} . For query row t ,

$$p_{\theta,t}^{(k)} = \text{softmax}(g_\theta([S_{nk}]_t)). \quad (3)$$

The same decoder serves every k , so predictions can be read at any depth without depth-specific parameters.

Parameters and compute. k loops cost nk block applications with only n blocks of parameters. With $n = 4$ and $L = 12$, ICL parameters fall from 25.7M to 8.6M and total parameters from 27.6M to 10.4M. At $k = 3$, both models apply 12 blocks, so they cost the same per table. Tying therefore saves memory, not FLOPs. Saving FLOPs requires running fewer loops (Section 5.2).

3.2 TRAINING AND INFERENCE

Curriculum. Following TabICLv2 (Qu et al., 2026), pretraining proceeds in three stages that progressively increase the size of the training tables. Stage 1 trains for 500K steps on tables of 1,024 rows. Stage 2 continues for 40K steps on tables of 400 to 10,240 rows, and stage 3 continues for a further 10K steps on tables of 400 to 60,000 rows with a lower learning rate (Table 6). The later stages therefore adapt a model trained on small tables to large contexts. In this version, the untied baseline completes all three stages, while Tab-Loop completes stages 1 and 2 (see Appendix B).

For a training table $\mathcal{D}$ with query rows $\mathcal{Q}_{\mathcal{D}}$, the supervised loss after k loops is

$$\ell_{\text{CE}}(\theta; \mathcal{D}, k) = -\frac{1}{|\mathcal{Q}_{\mathcal{D}}|} \sum_{t \in \mathcal{Q}_{\mathcal{D}}} \log p_{\theta,t}^{(k)}[y_t]. \quad (4)$$

Stage 1. For each micro-batch we sample $k \sim \mathcal{U}\{2, 3, 4\}$ and backpropagate through all loops, with supervision only at the final depth. The expected depth matches the baseline,

$$n \mathbb{E}_k[k] = 3n = L, \quad (5)$$

so Tab-Loop and the baseline cost the same per training step. We use $\{2, 3, 4\}$ rather than a wider range with the same mean to keep per-step cost and activation memory close to the baseline’s; $k = 1$ is therefore never seen in training. The baseline uses the same loss, prior and optimization schedule.

Stage 2. To train deeper loops, we sample $k = \min\{2 + z, 32\}$ with $z \sim \text{Poisson}(10)$ for each micro-batch and run one extra loop for a latent-consistency term. For query row i with ICL state $s_k^{(i)}$ after k loops,

$$\ell_{\text{LC}}(\theta; \mathcal{D}, k) = \frac{1}{|\mathcal{Q}_{\mathcal{D}}|} \sum_{i \in \mathcal{Q}_{\mathcal{D}}} \frac{\|s_{k+1}^{(i)} - \text{sg}(s_k^{(i)})\|_2^2}{\|\text{sg}(s_k^{(i)})\|_2^2 + \varepsilon}, \quad (6)$$

where sg is stop-gradient and $\varepsilon = 10^{-6}$. The stage-2 objective is $\ell_{\text{CE}} + \lambda \ell_{\text{LC}}$ with $\lambda = 1$. We backpropagate through the last $\min\{8, k + 1\}$ loops; earlier loops run without gradient tracking. The baseline follows the same stage without the loop-specific terms. In this version, Tab-Loop is trained through stage 2 only (Section 5.3).

Inference. A budget of k loops costs nk ICL block applications, in addition to the shared encoders and decoder. Because the same decoder serves every depth, k can be changed at inference without retraining. In this work, k is set globally for each evaluation.

Table 2: **Models compared in stage 1.** Per-step cost is measured (forward and backward). Control steps are 24% cheaper, so the control trains for 660K steps on 32% more tables to reach the same total compute.

Model	ICL blocks	Params	Applications	TFLOP/step	Steps	Total EFLOP
Untied baseline	12 untied	27.6M	12	36.4	500K	18.2
Tab-Loop	4 tied	10.4M	4k (mean 12)	36.4	500K	18.2
Untied control	4 untied	10.4M	4	27.7	660K	18.2

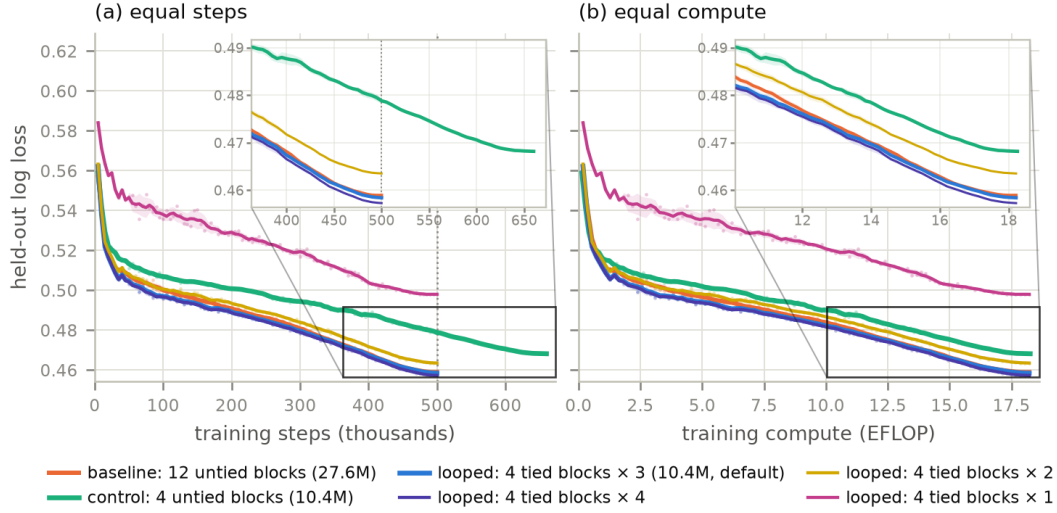


Figure 3: **Held-out log loss during stage 1** on 1,024 synthetic tables, against (a) training steps and (b) training compute. Tab-Loop is read at $k \in \{1, 2, 3, 4\}$ from the same checkpoints. Each control step is about 24% cheaper, so its 660K steps match the 500K steps of the other runs in compute. At equal compute, Tab-Loop at $k = 3$ tracks the baseline throughout training, and every reading with $k \geq 2$ stays below the control.

4 EXPERIMENTAL DESIGN

Three models share the prior, encoders, decoder, optimizer and schedule, and each receives 18.2 EFLOP of stage-1 training (Table 2). They differ only in the ICL stage. Tab-Loop against the baseline holds depth and compute fixed and varies tying, so it measures parameter efficiency. Tab-Loop against the control holds parameters and training compute fixed and varies depth, so it measures whether compute is better spent on depth or on more training data.

Benchmarks. We evaluate on three suites, with all models applied zero-shot using TabICLv2’s default preprocessing (Appendix B.4). (i) *Held-out prior*: 1,024 synthetic tables drawn with a seed disjoint from training, cached once and reused for every checkpoint, so comparisons are paired. We report query log loss. (ii) *TabArena* (Erickson et al., 2025): 38 classification datasets and all 594 splits, with Elo against the public leaderboard and 95% bootstrap intervals. Intermediate checkpoints and stage-2 models are evaluated with TabArena-Lite (one split per dataset), whose Elo is not comparable with full TabArena. (iii) *TALENT* (Liu et al., 2024): 200 classification datasets with official splits. We report mean test accuracy.

5 RESULTS

5.1 AT EQUAL COMPUTE, TIED DEPTH MATCHES UNTIED DEPTH

Tying is free. At $k = 3$, which matches the baseline’s 12 block applications, Tab-Loop tracks the baseline’s held-out loss throughout training (Figure 3). Over the final 100K steps it has the lower loss

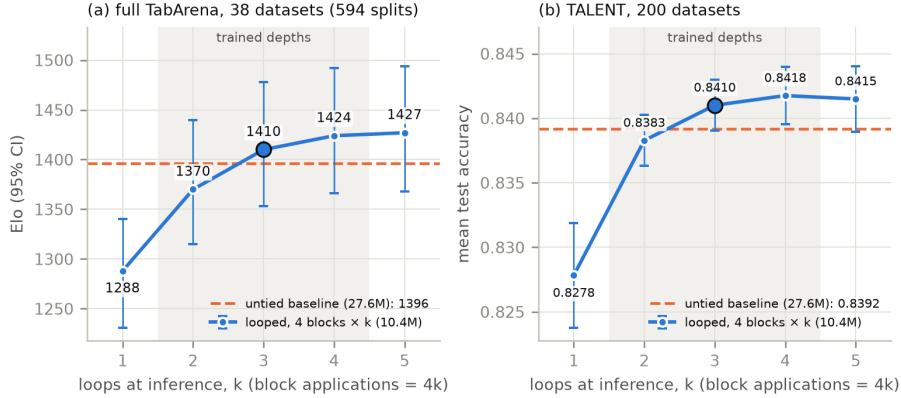


Figure 4: **Test-time depth for the stage-1 checkpoint.** Tab-Loop (10.4M parameters) read at k loops on (a) full TabArena (38 datasets, 594 splits) and (b) TALENT (200 datasets), against the untied 12-block baseline (27.6M, dashed). Shading marks the depths sampled in training. Performance rises through the trained range and is flat one loop beyond it.

at 94 of 100 checkpoints, and it ends at 0.4583 against 0.4588. The equivalence carries to real data (Figure 1): mean TALENT accuracy is 84.10% against 83.92%, and full TabArena Elo is 1409 against 1396, with overlapping 95% intervals. With one seed, we read this as equivalence, not superiority. The two models also trade places at intermediate budgets: at 7.9 EFLOP the baseline leads on both real-data suites, and at 3.6 EFLOP Tab-Loop leads on TabArena-Lite (Appendix Figure 7).

Depth beats data. The control has Tab-Loop’s parameter count, applied once. Its steps are 24% cheaper, so at equal compute it trains on 32% more tables. It still trails Tab-Loop at every evaluation, including the end of training. On held-out tables it ends at 0.4681. On TALENT its accuracy is 83.46%, against 84.10% for Tab-Loop (better on 130 of 200 datasets) and 83.92% for the baseline. Its full TabArena Elo is 1343, against 1409 and 1396. At this scale, compute spent re-applying shared weights is worth more than compute spent on more data.

The computation is spread across loops. Tab-Loop’s weights are not a good 4-block model refined by extra passes. Read at one loop, a depth never used in training, they are worse than the control throughout training (Figure 3, pink against green) and at the end: held-out loss 0.4979 against 0.4681, TALENT accuracy 82.78% against 83.46%, and TabArena Elo 1285 against 1343. Taken together, the two comparisons indicate that the ICL stage needs depth, not a distinct set of parameters for every layer.

5.2 DEPTH IS A DIAL WITHIN THE TRAINED RANGE

Because the blocks are shared, the number of loops is free at inference. Figure 4 varies k for the final stage-1 checkpoint. Performance rises with depth on both benchmarks. One loop, effectively a 4-block model, falls well below the baseline (Elo 1288; accuracy 82.78%), and two loops come close (Elo 1370; 83.83%). Four loops, at a third more inference compute than the baseline, give the best results in the trained range (Elo 1424; 84.18%); this is the only depth whose TALENT gain over the baseline is significant. Beyond the trained range, Elo is 1427 at $k = 5$ and TALENT accuracy holds at 84.15% for $k = 5$ and $k = 6$. Extra loops thus help up to about one loop beyond training and are neutral after that, without collapsing. The useful range of the dial is set by the depths seen in training.

5.3 TRAINING WITH DEEPER LOOPS EXTENDS THE DIAL

If the dial’s range is set by training, training deeper should move it. We continue Tab-Loop and the baseline into stage 2, where tables grow to 400–10,240 rows, and train Tab-Loop with $k = \min(2 + \text{Poisson}(10), 32)$, which centres training near 12 loops (Figure 5).

The range moves as predicted. On TabArena-Lite, Tab-Loop rises from Elo 1298 at $k = 3$ to 1512 at $k = 12$, against 1509 for the stage-2 baseline, and plateaus near 1548 for k between 24 and 32,

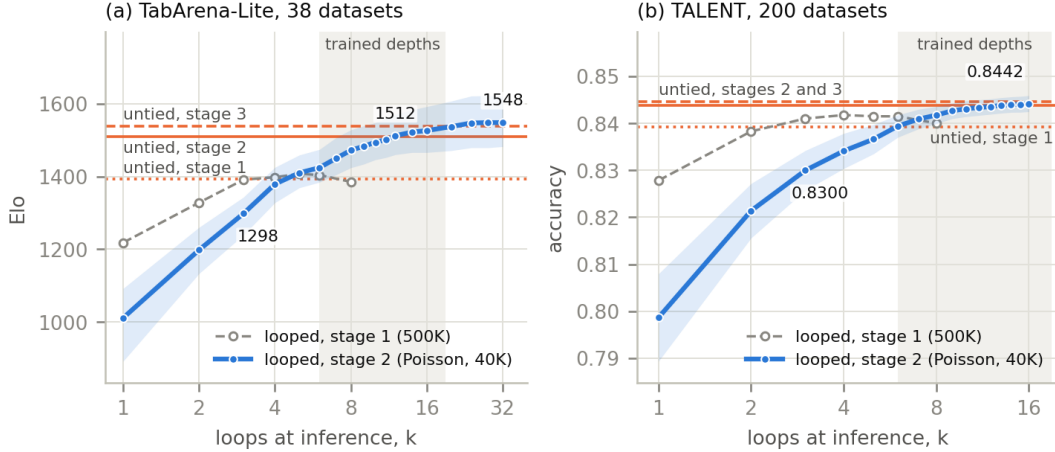


Figure 5: **Test-time depth after stage 2.** Tab-Loop after 40K stage-2 steps with $k = \min(2 + \text{Poisson}(10), 32)$ (blue), compared with its stage-1 checkpoint (grey, trained on $k \in \{2, 3, 4\}$) and with the untied baseline after stages 1, 2 and 3 (horizontal lines). **(a)** TabArena-Lite (38 datasets). **(b)** TALENT (200 datasets). Shading marks the central 95% of stage-2 training depths. The x-axis is logarithmic; k loops cost $4k$ block applications.

against 1539 for the baseline after stage 3. On TALENT, it rises from 83.00% at $k = 3$ to 84.42% at $k = 16$, between the baseline’s stage-2 (84.39%) and stage-3 (84.46%) scores. Three caveats bound this result. First, it is test-time compute scaling, not parameter efficiency at equal cost: $k = 12$ is 48 block applications, $4\times$ the baseline, and $k = 16$ is 64, about $5.3\times$. Second, training deep moves the operating point rather than widening it: at $k = 3$, the stage-2 model is weaker than its own stage-1 checkpoint on both suites. Third, training compute is not matched in this stage, since a Tab-Loop step unrolls about 13 loops on average (52 block applications in the forward pass, with backpropagation through at most 32) against 12 block applications for a baseline step, and Tab-Loop has not been trained through stage 3. We therefore read this result as evidence that the dial’s range is learnable, not as a compute-matched comparison.

6 RELATED WORK

Tabular foundation models. TabPFN (Hollmann et al., 2023) showed that a transformer pretrained on synthetic tables can classify real tables without parameter updates. TabPFN v2 (Hollmann et al., 2025), TabICL (Qu et al., 2025) and TabICLv2 (Qu et al., 2026) improved the prior, the architecture and the training curriculum. Mitra (Zhang et al., 2025b) mixes synthetic priors, TabDPT (Ma et al., 2025) trains on real data, and TabPFN-2.5 (Prior Labs, 2025), TabPFN-3 (Grinsztajn et al., 2026), LimiX (Zhang et al., 2025a), EXAONE Tabular (Eo et al., 2026) and TabSwift (Liu & Ye, 2026) extend the family in scale, scope and efficiency. Attention along both rows and columns, which these models share, traces back to non-parametric transformers (Kossen et al., 2021). These models use untied layers. We change only that choice and hold the rest of TabICLv2 fixed. We evaluate on TabArena (Erickson et al., 2025) and TALENT (Liu et al., 2024).

Weight sharing across depth. Universal Transformers (Dehghani et al., 2019) and ALBERT (Lan et al., 2020) share parameters across layers. Looped transformers can emulate programs and iterative learning algorithms (Giannou et al., 2023), and they match standard transformers on in-context learning tasks with fewer parameters, with input injection as a key component (Yang et al., 2024). Saunshi et al. (2025) show that reasoning benefits from depth without a proportional increase in distinct parameters, looped transformers generalize to longer inputs than they were trained on (Fan et al., 2025), and recurrent networks extrapolate from easy to hard problems by iterating longer (Schwarzschild et al., 2021). Looped language models have since been pretrained at scale (Zhu et al., 2025). Relaxed Recursive Transformers (Bae et al., 2025a) and Gated Recurrent Transformers (Hegazy et al., 2026) study the trade-off between sharing and specialization in language models.

Recurrent depth at test time. Geiping et al. (2025) scale test-time compute in language models through recurrent depth and describe convergent, orbiting and drifting latent trajectories. Fixed-point formulations stabilize deep loops (Movahedi et al., 2026), and mechanistic analyses show that a single looped block can carry out different stages of computation across iterations (Blayney et al., 2026). These dynamics motivate our question of how far beyond the trained depths a looped model remains useful.

Adaptive computation. Adaptive Computation Time (Graves, 2016) and PonderNet (Banino et al., 2021) learn when to halt, and Mixture-of-Recursions (Bae et al., 2025b) learns a recursion depth per token. For tabular in-context learning, early stopping of the forward pass has been studied for untied models (Küken et al., 2025). Tab-Loop makes depth a choice at inference; selecting it adaptively for each table is left to future work.

Recurrence for tabular ICL. Balef et al. (2026) show that the layers of tabular foundation models are largely redundant and give a small-scale proof of concept for layer reuse, and task-aware pruning removes many of their blocks (Koshil et al., 2026). Iterative computation also appears in state-space tabular models (Thielmann et al., 2024; Koch et al., 2025) and in tiny recursive networks (Jolicoeur-Martineau, 2025). Unlike these, Tab-Loop changes only the ICL stage of an established tabular foundation model and compares it against both an untied baseline and a parameter-matched control at equal compute, so that differences can be attributed to tying and depth.

7 CONCLUSION

At the scale of modern tabular foundation models, four tied blocks applied three times match twelve distinct blocks at equal training and inference compute while using 62% fewer parameters, and a parameter-matched model that spends the same compute on more data instead of more depth falls behind. The ICL stage of a tabular foundation model therefore needs depth more than it needs distinct parameters. Tying also turns depth into a choice made at inference: within the trained range, additional loops give modest and reliable gains, and training with deeper loops extends that range.

ACKNOWLEDGMENTS

We thank Mila – Quebec AI and Nolano AI for computational resources and support.

REFERENCES

- Sangmin Bae, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Seungyeon Kim, and Tal Schuster. Relaxed recursive transformers: Effective parameter sharing with layer-wise LoRA. In *International Conference on Learning Representations*, pp. 34282–34327, 2025a. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/54d6a55225cebbdc16fbb0e45c5bdf2b-Abstract-Conference.html.
- Sangmin Bae, Yujin Kim, Reza Bayat, Sungnyun Kim, Jiyouon Ha, Tal Schuster, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Aaron Courville, and Se-Young Yun. Mixture-of-recursions: Learning dynamic recursive depths for adaptive token-level computation, 2025b. URL <https://arxiv.org/abs/2507.10524>. arXiv:2507.10524.
- Amir Rezaei Balef, Mykhailo Koshil, and Katharina Eggersperger. Is one layer enough? understanding inference dynamics in tabular foundation models, 2026. URL <https://arxiv.org/abs/2605.06510>.
- Andrea Banino, Jan Balaguer, and Charles Blundell. PonderNet: Learning to ponder, 2021. URL <https://arxiv.org/abs/2107.05407>. arXiv:2107.05407.
- 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, 2026.
- Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2019. URL <https://openreview.net/forum?id=HyzdRiR9Y7>.

- Moonjung Eo, Min-Kook Suh, Hye-Seung Cho, Jiwon Kim, Seoyoon Kim, Sangjun Nam, and Soonyoung Lee. EXAONE Tabular 1.0: Technical report, 2026.
- Nick Erickson, Lennart Purucker, Andrej Tschalzev, David Holzmüller, Prateek 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. Curran Associates, Inc., 2025. doi: 10.52202/085713-0519. URL https://papers.neurips.cc/paper_files/paper/2025/hash/1697e3fb412da11dc9488249f9e7bbc9-Abstract-Datasets_and_Benchmarks_Track.html.
- Ying Fan, Yilun Du, Kannan Ramchandran, and Kangwook Lee. Looped transformers for length generalization. In *International Conference on Learning Representations*, 2025.
- Jerome H. 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 R. Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. In *Advances in Neural Information Processing Systems*, volume 38, pp. 41340–41391. Curran Associates, Inc., 2025. doi: 10.52202/085713-1380. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/3b01972cf31e6fa0fe29e4b8b5c2a0a1-Abstract-Conference.html.
- Angeliki Giannou, Shashank Rajput, Jy-Yong Sohn, Kangwook Lee, Jason D. Lee, and Dimitris Papailiopoulos. Looped transformers as programmable computers. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 11398–11442. PMLR, 2023. URL <https://proceedings.mlr.press/v202/giannou23a.html>.
- Alex Graves. Adaptive computation time for recurrent neural networks, 2016. URL <https://arxiv.org/abs/1603.08983>. arXiv:1603.08983.
- Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. Why do tree-based models still outperform deep learning on typical tabular data? In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, 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, 2026.
- Amr Hegazy, Amr Alanwar, and Mostafa Elhoushi. Gated recurrent transformers: Expressive depth through recurrent modulation, 2026. URL <https://arxiv.org/abs/2608.15062>. arXiv:2608.15062.
- Noah Hollmann, Samuel Müller, Katharina Eggenberger, and Frank Hutter. TabPFN: A transformer that solves small tabular classification problems in a second. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=cp5PvcI6w8_.
- Noah Hollmann, Samuel Müller, Lennart Purucker, Arjun Krishnakumar, Max Körfer, Shi Bin Hoo, Robin Tibor Schirrmeyer, and Frank Hutter. Accurate predictions on small data with a tabular foundation model. *Nature*, 637:319–326, 2025. doi: 10.1038/s41586-024-08328-6. URL <https://www.nature.com/articles/s41586-024-08328-6>.
- Alexia Jolicoeur-Martineau. Less is more: Recursive reasoning with tiny networks, 2025.
- Felix Koch, Marcel Wever, Fabian Raisch, and Benjamin Tischler. State-space models for tabular prior-data fitted networks, 2025.
- Weihao Kong, Erez Louidor Ilan, Shuxin Nie, Taman Narayan, Rajat Sen, Yichen Zhou, Deqing Fu, Samet Oymak, and Abhimanyu Das. Tabfm: A zero-shot foundation model for tabular data, 2026. URL <https://arxiv.org/abs/2609.37959>.
- Mykhailo Koshil, Matthias Feurer, and Katharina Eggenberger. TACTICL: Task-aware compression of tabular ICL models, 2026.

- Jannik Kossen, Neil Band, Clare Lyle, Aidan N. Gomez, Tom Rainforth, and Yarin Gal. Self-attention between datapoints: Going beyond individual input-output pairs in deep learning. In *Advances in Neural Information Processing Systems*, volume 34, pp. 28742–28756, 2021.
- Jaris Küken, Lennart Purucker, and Frank Hutter. Early stopping tabular in-context learning. In *1st International Workshop on Foundation Models for Structured Data (FMSD) at ICML*, 2025.
- Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A lite BERT for self-supervised learning of language representations. In *International Conference on Learning Representations*, 2020. URL <https://openreview.net/forum?id=H1eA7AetvS>.
- Si-Yang Liu and Han-Jia Ye. TabSwift: An efficient tabular foundation model with row-wise attention. In *Proceedings of the 43rd International Conference on Machine Learning*, Proceedings of Machine Learning Research, 2026.
- Si-Yang Liu, Hao-Run Cai, Qi-Le Zhou, and Han-Jia Ye. Talent: A tabular analytics and learning toolbox, 2024. URL <https://arxiv.org/abs/2407.04057>.
- Junwei Ma, Valentin Thomas, Rasa Hosseinzadeh, Alex Labach, Jesse Cresswell, Keyvan Golestan, Guangwei Yu, Anthony L Caterini, and Maks Volkovs. Tabdpt: Scaling tabular foundation models on real data. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 172692–172722. Curran Associates, Inc., 2025. doi: 10.52202/085713-5748. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/fc0e3f908a2116ba529ad0a1530a3675-Paper-Conference.pdf.
- Duncan McElfresh, Sujay Khandagale, Jonathan Valverde, Vishak Prasad C, Ganesh Ramakrishnan, Micah Goldblum, and Colin White. When do neural nets outperform boosted trees on tabular data? In *Advances in Neural Information Processing Systems*, pp. 76336–76369, 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 *Proceedings of the 43rd International Conference on Machine Learning*, 2026.
- 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, 2026.
- Samuel Müller, Noah Hollmann, Sebastian Pineda Arango, Josif Grabocka, and Frank Hutter. Transformers can do bayesian inference, 2024. URL <https://arxiv.org/abs/2112.10510>.
- Prior Labs. TabPFN-2.5: Advancing the state of the art in tabular foundation models, 2025.
- 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. URL <https://proceedings.mlr.press/v267/qu25d.html>.
- Jingang Qu, David Holzmüller, Gaël Varoquaux, and Marine Le Morvan. TabICLv2: A better, faster, scalable, and open tabular foundation model, 2026. URL <https://arxiv.org/abs/2602.11139>. arXiv:2602.11139.
- 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*, pp. 14855–14881, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/2676109d49d1eb26d6bc584a8f556305-Abstract-Conference.html.

- 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. In *Advances in Neural Information Processing Systems*, volume 34, pp. 6695–6706, 2021.
- Anton Frederik Thielmann, Manish Kumar, Christoph Weisser, Arik Reuter, Benjamin Säfken, and Soheila Samiee. Mambular: A sequential model for tabular deep learning, 2024.
- Johannes von Oswald, Eyvind Niklasson, Ettore Randazzo, João Sacramento, Alexander Mordvintsev, Andrey Zhmoginov, and Max Vladymyrov. Transformers learn in-context by gradient descent, 2023. URL <https://arxiv.org/abs/2212.07677>.
- Liu Yang, Kangwook Lee, Robert D. Nowak, and Dimitris Papailiopoulos. Looped transformers are better at learning learning algorithms. In *International Conference on Learning Representations*, pp. 42195–42214, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/b8402301e7f06bdc97a31bfaa653dc32-Abstract-Conference.html.
- Xingxuan Zhang, Gang Ren, Han Yu, Hao Yuan, Hui Wang, et al. LimiX: Unleashing structured-data modeling capability for generalist intelligence, 2025a.
- Xiyuan Zhang, Danielle C. Maddix, Junming Yin, Nick Erickson, Abdul Fatir Ansari, Boran Han, Shuai Zhang, Leman Akoglu, Christos Faloutsos, Michael W. Mahoney, Cuixiong Hu, Huzefa Rangwala, George Karypis, and Bernie Wang. Mitra: Mixed synthetic priors for enhancing tabular foundation models, 2025b. URL <https://arxiv.org/abs/2510.21204>.
- 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, 2025.

A ARCHITECTURE DETAILS

A.1 SHARED COMPONENTS

Tab-Loop inherits the column encoder, row encoder, label embeddings and prediction head of TabICLv2 (Qu et al., 2026) without modification (Table 3). All pre-ICL components run once per forward pass; their output $X_0 \in \mathbb{R}^{T \times d}$ is cached and reused as the injected input at every ICL block application (Eq. 2).

Table 3: Components shared between the untied baseline, the control and Tab-Loop. Parameter counts are identical in all three models.

Stage	Role	Parameters
Column encoder	Per-column cell embeddings (set transformer over rows)	1.8M total
Row encoder	Aggregates cell embeddings across features into a row vector	
Label embedding + head	Adds class embeddings to context rows; linear decoder g_θ	

A.2 ICL BLOCK SPECIFICATION

Each ICL block C_j (and each baseline block B_ℓ) is a standard pre-norm transformer block:

$$Z = S + \text{MHA}(\text{LN}_1(S); M), \quad (7)$$

$$C_j(S) = Z + \text{FFN}(\text{LN}_2(Z)), \quad (8)$$

where M is the context-only attention mask (every row attends to context rows only; query rows never attend to one another), MHA is multi-head self-attention and FFN is a two-layer GELU MLP. Table 4 lists the hyperparameters. The block is identical in all models; Tab-Loop differs from the baseline only in the number of distinct blocks (4 against 12), their cyclic reuse and the additive input injection.

Table 4: ICL block hyperparameters (identical for all models).

Model width d	512
Attention heads	8
Head dimension	64
FFN hidden width	2048
Activation	GELU
Normalization	Pre-LayerNorm
Dropout	0.0
Parameters per block	$\approx 2.14\text{M}$
Distinct ICL blocks (baseline / Tab-Loop)	12 / 4
ICL parameters (baseline / Tab-Loop)	25.7M / 8.6M
Total parameters (baseline / Tab-Loop)	27.6M / 10.4M

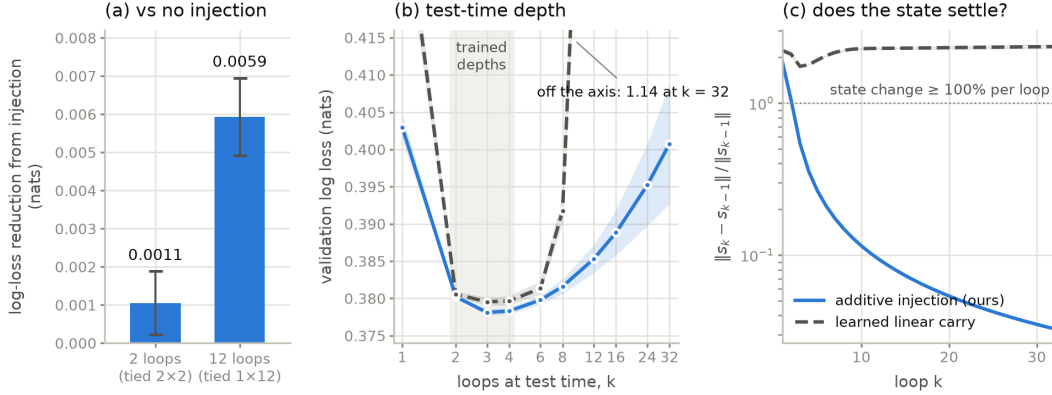


Figure 6: **Input injection** (small-scale study: width 128, tied ICL blocks, 10^{17} training FLOPs per run). **(a)** Reduction in validation log loss from re-adding X_0 before every block application, against no injection, at 2 loops (tied 2×2) and 12 loops (tied 1×12), with paired 95% intervals. **(b)** Validation log loss against test-time depth for additive injection and a learned linear carry ($S \leftarrow W[S; X_0]$, W initialized to $[I, I]$), both trained with $k \sim \mathcal{U}\{2, 3, 4\}$; mean and range over two seeds. **(c)** Relative change of the ICL state per loop, median over 512 validation tables. It shrinks steadily with additive injection but stays above 100% per loop with the learned carry, whose loss diverges beyond the trained depths.

A.3 INPUT INJECTION

Before every block application $i > 0$, the cached stage input X_0 is added to the current state, so the block input is $S_i + X_0$. The injection is applied uniformly, including between blocks within a loop, and has no learned parameters or scaling. We compared it with two alternatives in a small-scale study (Figure 6):

- *No injection*, where blocks receive only the current state S_i . Re-adding X_0 lowers validation log loss by 0.0011 nats at 2 loops and by 0.0059 nats at 12 loops, so its benefit grows with depth.
- *Learned linear carry*, $S_i \leftarrow W[S_i; X_0]$ with $W \in \mathbb{R}^{d \times 2d}$ initialized to $[I, I]$. It is slightly worse than additive injection at the trained depths. Left unconstrained, the carry becomes expansive: the largest singular value of its state block grows to 2.2, and the state changes by more than its own norm at every loop. Its loss diverges beyond the trained range, reaching 1.14 nats at $k = 32$, against about 0.40 with additive injection.

We therefore use the parameter-free injection and leave constrained carries to future work.

A.4 COMPUTE ACCOUNTING

Per-application FLOPs are identical across models, since the block is identical. For a table with T rows, one forward block application costs approximately $24Td^2 + 4T^2d$ FLOPs (attention and FFN), about 44 GFLOP at the training table sizes. Table 5 lists the inference compute of the configurations in the main text.

Table 5: ICL-stage inference compute. The encoders and prediction head are identical across models and excluded.

Configuration	Block applications	% of baseline
Untied baseline ($L = 12$)	12	100
Untied control ($L = 4$)	4	33
Tab-Loop, $k = 1$	4	33
Tab-Loop, $k = 2$	8	67
Tab-Loop, $k = 3$ (stage-1 default)	12	100
Tab-Loop, $k = 4$	16	133
Tab-Loop, $k = 5$	20	167
Tab-Loop, $k = 6$	24	200
Tab-Loop, $k = 12$ (stage-2 default)	48	400
Tab-Loop, $k = 16$	64	533

B TRAINING AND EVALUATION DETAILS

B.1 PRETRAINING DATA

All models are pretrained on synthetic tables drawn from the TabICLv2 prior (Qu et al., 2026), which mixes structural-causal-model, tree-based and nearest-neighbour generators with randomized feature counts, class counts and context sizes. No real-world data is used in pretraining. Each table is split into labelled context rows and unlabelled query rows, and the loss in Eq. 4 is computed on query rows only. A fixed held-out slice of 1,024 tables (16 batches of 64) is drawn from the same prior with a seed disjoint from training. It is cached once and reused for every checkpoint of every model, so per-checkpoint comparisons are paired.

B.2 OPTIMIZATION

Table 6 lists the training settings. They are shared by the untied baseline, the control and Tab-Loop; only the number of stage-1 steps differs for the control (660K instead of 500K), to equalize total pretraining compute at 18.2 EFLOP.

B.3 LOOP-COUNT SAMPLING AND GRADIENT TRUNCATION

Stage 1. The loop count is sampled once per micro-batch, $k \sim \mathcal{U}\{2, 3, 4\}$, and shared by all tables in the micro-batch so that it can be processed as a single unrolled graph. Gradients flow through every loop. The expected number of block applications is $4 \cdot \mathbb{E}[k] = 12$, equal to the baseline, so per-step cost matches.

Stage 2. The loop count is sampled as $k = \min\{2+z, 32\}$ with $z \sim \text{Poisson}(10)$, giving a mean of about 12 loops. One additional loop is run after the supervised depth to compute the latent-consistency term in Eq. 6, with weight $\lambda = 1$. Backpropagation is truncated to the final $\min\{8, k+1\}$ loops: the earlier loops run under `torch.no_grad`, their final state is detached, and the differentiable tail runs from that state.

B.4 EVALUATION PROTOCOLS

- **Held-out prior:** mean query log loss on the cached held-out slice. Tab-Loop is evaluated at $k \in \{1, \dots, 6\}$ at every checkpoint, and the baseline and control at their single fixed depth.

Table 6: Training settings, shared by the untied baseline and Tab-Loop unless marked. The control follows the stage-1 recipe only. In this version, Tab-Loop is trained through stage 2; the stage-3 column applies to the baseline.

	Stage 1	Stage 2	Stage 3
<i>Data (synthetic prior)</i>			
Rows per table	1,024	400–10,240, log-uniform	400–60,000, log-uniform
In-context (training) fraction	30–90%	80%	80%
Features / classes		2–100 / up to 10	
Batch		64 tables per step	
<i>Optimization</i>			
Optimizer	Muon (2D hidden weights) + AdamW (embeddings, heads, norms, biases)		
Peak learning rate, Muon / AdamW	$8 \times 10^{-4} / 1 \times 10^{-4}$	$1 \times 10^{-4} / 1 \times 10^{-4}$	$2 \times 10^{-5} / 2 \times 10^{-5}$
Muon momentum (Nesterov); AdamW (β_1, β_2)		0.95; (0.95, 0.95)	
Weight decay (decoupled)		0.1	
Warmup (linear)	10K steps	800 steps	200 steps
Decay		cosine to 0	
Gradient clipping (global norm)	10	10	1.0
Training steps	500K (control: 660K)	40K	10K
Checkpoint interval	5K	1K	250
Activation checkpointing	–	–	tables >20K rows
Precision		bf16 autocast, fp32 master weights	
<i>Tab-Loop only</i>			
Loops per training forward, k	$\mathcal{U}\{2, 3, 4\}$	$\min(2 + \text{Poisson}(10), 32)$	–
Latent-consistency weight λ	–	1	–
Backpropagation	through all k loops	through the last 8 of $k+1$ loops	–
Default inference depth	3	12	–
<i>Compute (forward + backward)</i>			
Cost per step, baseline / Tab-Loop / control	36.4 / 36.4 / 27.7 TFLOP	–	–
Total, baseline / Tab-Loop / control	18.2 / 18.2 / 18.2 EFLOP	–	–

- **TabArena:** we follow the TabArena protocol (Erickson et al., 2025) on its 38 classification datasets and all 594 outer-fold splits, and compute Elo against the public leaderboard with 95% bootstrap intervals. Intermediate stage-1 checkpoints (Appendix C) and stage-2 models (Section 5.3) are evaluated with TabArena-Lite, which uses one split per dataset; its Elo is not comparable with full TabArena. Models are applied zero-shot, with inputs preprocessed by the TabICLv2 default pipeline.
- **TALENT:** we evaluate on the 200 classification datasets of TALENT (Liu et al., 2024) with the official train/validation/test splits and report mean test accuracy.

C ADDITIONAL RESULTS

C.1 EQUAL-COMPUTE COMPARISON AT INTERMEDIATE BUDGETS

Figure 7 compares the three stage-1 models at two intermediate budgets of equal training compute. The control trails Tab-Loop at $k = 3$ at both budgets. Tab-Loop and the baseline trade places: at 3.6 EFLOP, Tab-Loop leads on TabArena-Lite (1215 against 1197) and on TALENT (82.65% against 82.58%); at 7.9 EFLOP, the baseline leads on both (1276 against 1248; 83.15% against 82.90%). By the end of training at 18.2 EFLOP, the two are equivalent (Section 5.1).

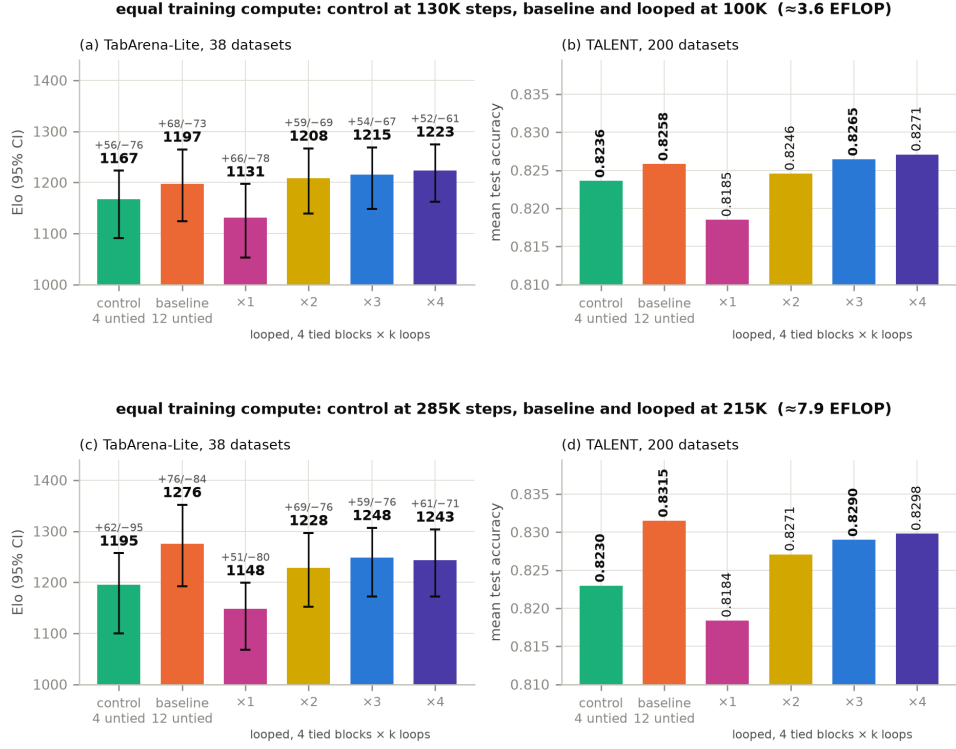


Figure 7: **Equal-compute comparison at intermediate budgets.** Top: control at 130K steps, baseline and Tab-Loop at 100K steps (≈ 3.6 EFLOP). Bottom: control at 285K steps, baseline and Tab-Loop at 215K steps (≈ 7.9 EFLOP). (a, c) TabArena-Lite (38 datasets), Elo with 95% bootstrap intervals. (b, d) TALENT (200 datasets), mean test accuracy. Tab-Loop is read at $k \in \{1, 2, 3, 4\}$; $k = 3$ matches the baseline’s 12 block applications.