TEACH ONCE, EXECUTE REPEATEDLY: TEST-TIME TRAINING IN LOOPED TRANSFORMERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Test-time training (TTT) changes a model for the current task; looped Transformers reuse the same block for more steps. We study their composition: an update learned from demonstrations is placed inside a tied loop and executed at every recurrent visit. This placement changes its finite-horizon effect. In a linear recurrence, a rank-r update reused for K visits can induce an operator difference of rank Kr, and we show this is tight: exactly reproducing it with a single-use linear update requires rank Kr. For nonlinear networks, we use complete visit gating to recover interaction orders one through K. Test-Time Compiled Loops (TTCL) learns a low-rank adapter from demonstrations and shares it across the loop. On NumSeqBench, TTCL reaches 93.3% exact-match accuracy on held-out arithmetic rules, compared with 0% when the same update is used once. In a semantic-state audit, a rank-one update reaches executed rank 16 in a 17-dimensional state and remains exact for 32 visits. On Ouro-1.4B and Ouro-2.6B LoopLMs, shared execution improves answer NLL over the same adapter used once while approaching a depthwise adapter with four times as many task parameters. Visit-gating interventions at $K = 4$ expose high-order hidden interactions and sensitivity to visit order. Random norm-matched adapters produce larger interactions without improving NLL, indicating that the reuse gain depends on the learned update.

1 INTRODUCTION

Test-time training (TTT) learns from the current episode after deployment. Looped Transformers reuse a weight-tied block, giving the model additional recurrent steps. These two forms of test-time computation have largely been studied separately. We ask what changes when a loop repeatedly executes the update that TTT just learned.

When an adapter is placed inside a tied loop, it is applied at each of the K visits and composed with a different hidden state. We call the resulting procedure *compilation*: the test-time optimizer produces a task-conditioned transition, which the tied block applies repeatedly. This can yield effects that a one-shot update of the same rank cannot reproduce.

Under the linear insertion model analyzed below, consider a recurrent operator A and a rank-r test-time update Δ_τ. A single-use network receives a perturbation of rank at most r. A looped network instead computes $(A + \Delta_\tau)^K$. Its task-conditioned effect spans

$$\mathcal{K}_K(A, U) = \mathrm{span}\{U, AU, \ldots, A^{K-1}U\},$$

whose dimension reaches Kr in the construction. We call the separation between the size of the learned update and the complexity of its executed effect *rank–depth capacity*. This is the rank of a finite-horizon operator difference, not Kr adapter degrees of freedom. We give a tight construction: a rank-r recurrent program realizes a target for which every rank-q single-use approximation has Frobenius error at least $\sqrt{Kr - q}$. Adapter rank is the stored task-update dimension; recurrent depth controls how often the update is executed. The learned parameter count remains fixed.

Since the linear analysis does not directly apply to nonlinear networks, we attach binary gates to recurrent visits and evaluate all 2^K subsets. A Boolean-lattice Möbius transform then decomposes the final representation into first-order effects and nonadditive visit interactions of orders 2 through K. A second intervention permutes the same per-visit adapter doses while holding both their sum

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.

and squared sum fixed. These tests measure non-additive interactions across visits and sensitivity to visit order.

On two billion-scale LoopLMs, correct compiled updates produce substantial high-order hidden interactions, and matched-dose permutations change both hidden states and answer likelihood. A norm-matched random adapter produces even more raw high-order activity while providing no reuse gain.

We realize this view with *Test-Time Compiled Loops* (TTCL). TTCL learns a low-rank adapter from demonstrations while freezing the backbone, then reuses that adapter at every recurrent visit. Its task-specific parameter storage remains constant as runtime increases. Across held-out sequence rules, two LoopLM checkpoints, and answer-generation tasks, shared execution separates from the identical update used once and random or corrupted support, and approaches depthwise adapters with one quarter of the task parameters.

Our contributions are:

1. a compilation view of TTT that separates what a test-time update *stores* from what a recurrent architecture can *execute*;
2. tight rank–depth and approximation separations, together with an exact nonlinear intervention that recovers every order of recurrent interaction;
3. TTCL, a constant-task-memory method that compiles demonstrations into a low-rank program reused across recurrent depth; and
4. controlled evidence on two billion-scale LoopLMs and public downstream tasks, including equal-dose order tests and norm-matched random adapters that produce larger interactions without an NLL gain.

Scope. Our object of study is the finite computation produced when a transition learned at test time is executed repeatedly. The linear theorem states its capacity in a specified insertion model; the nonlinear experiments measure the resulting execution interactions directly.

2 Related Work

Looped Transformers have been studied as programmable computers, in-context iterative learners, and latent test-time compute models (Giannou et al., 2023; Yang et al., 2024; Gatmiry et al., 2024; Geiping et al., 2025). This line is also connected to recurrently reusing a Transformer block in the Universal Transformer (Dehghani et al., 2019) and to mechanistic accounts of in-context learning as an implicit optimization process (Von Oswald et al., 2023). Weight-tied depth also appears in deep equilibrium models (Bai et al., 2019), while adaptive computation and early-exit methods choose how much recurrent or layerwise compute an input receives (Graves, 2016; Xin et al., 2020). TTT ranges from episodic parameter adaptation to layers whose hidden states are themselves updated models, connecting to fast-weight systems (Schmidhuber, 1992; Sun et al., 2020; Schlag et al., 2021; Sun et al., 2024). Fully test-time adaptation by entropy minimization is a complementary distribution-shift approach that updates a restricted set of normalization parameters rather than a task adapter (Wang et al., 2021). Low-rank adapters (Hu et al., 2022) have also been used for TTT (Kojima et al., 2025). Meta-learning and learned-optimizer work learns update rules, metrics, or subspaces (Andrychowicz et al., 2016; Lee & Choi, 2018; Park & Oliva, 2019; Flennerhag et al., 2020).

Prior low-rank TTT studies how a small episode-specific update changes a model. TTCL studies the finite computation induced when that update is reused by a weight-tied architecture. This setting yields the rank–depth capacity result, constant task-parameter storage, and the shared-versus-single-use comparisons studied here.

3 Adaptation–Execution Duality

Let $A \in \mathbb{R}^{d\times d}$ be a base recurrent operator and let $B = A + \Delta$, where $\mathrm{rank}(\Delta) = r$. Recurrent reuse produces B^K; the unadapted computation is A^K.

Table 1: Structural distinction from the closest adaptation families. Here p is the parameter count of one task adapter.

Method	Tied recurrent base	Update shared over visits	Task parameters
Episodic LoRA-TTT	Not required	Not applicable	p
TTT layer / fast weight	Yes	Hidden learner state	Architecture-dependent
Depthwise LoRA control	Yes	No	Kp
TTCL (ours)	Yes	Yes	p

Definition 1 (Realized rank and rank-class capacity)**.** *The realized rank of an executed update is*

$$R_K(A, \Delta) = \operatorname{rank}\big((A+\Delta)^K - A^K\big).$$

For a fixed interpreter A, the capacity of the rank-r update class is

$$C_K(A, r) = \sup_{\operatorname{rank}(\Delta)\le r} R_K(A, \Delta).$$

The first quantity describes one compiled update; the second is a capacity of an update class. Neither is a parameter count.

In Theorem 1, *single-use* denotes one rank-q linear insertion in an otherwise linear depth-K computation.

Why looping changes the comparison. A rank-r update introduces r task-specific directions. Used once, those directions appear once in the computation. Used inside a tied loop, they are transported and recombined at every visit: the next visit sees their image under A, then the image of that image, and so on. The executed effect can therefore occupy K rank-r slices while the stored update remains unchanged.

Theorem 1 (Rank–depth separation)**.** *Let $r, K \ge 1$ be integers.*

1. *For every $A \in \mathbb{R}^{d\times d}$ and every Δ with $\operatorname{rank}(\Delta) \le r$,*
$$R_K(A, \Delta) \le \min\{d, Kr\}, \qquad C_K(A, r) \le \min\{d, Kr\}.$$
2. *If $d \ge Kr$, there exist A and a rank-r update Δ for which $R_K(A, \Delta) = Kr$.*
3. *For the same construction, let $T = (A+\Delta)^K - A^K$. Every matrix M with $\operatorname{rank}(M) \le q$ and $0 \le q \le Kr$ satisfies*
$$\|T - M\|_F \ge \sqrt{Kr - q}.$$
Hence an exact single-use realization requires rank at least Kr, while recurrent execution uses a rank-r update.

Proof. The noncommutative telescoping identity

$$(A+\Delta)^K - A^K = \sum_{j=0}^{K-1} A^{K-1-j}\Delta(A+\Delta)^j$$

is a sum of K matrices of rank at most r, proving part 1. For parts 2–3, use r disjoint K-dimensional blocks. On each block let A be the nilpotent shift $e_i \mapsto e_{i+1}$, $e_K \mapsto 0$, and let the rank-one block of Δ map $e_K \mapsto e_1$. Then $A+\Delta$ is cyclic, so $(A+\Delta)^K = I$ and $A^K = 0$ on all Kr coordinates. Thus T has exactly Kr singular values equal to one; unused coordinates are padded by zero. Eckart–Young gives the stated best rank-q error. Finally, inserting a rank-q update once into any linear depth-K computation produces $L\Gamma R$ for a rank-q matrix Γ and therefore has rank at most q. □

The theorem distinguishes stored description from executed effect. Reuse gives a rank-r update K opportunities to act, so its finite-horizon effect can reach Kr independent directions. In the construction above, a one-shot linear replacement needs rank at least Kr to realize the same map.

A factorized $d \times d$ rank-r update uses $O(dr)$ factor entries (typically $2dr$), not $O(r)$ parameters. Its *rank* is r, while its executed operator can have rank Kr. The result is an exact expressivity separation for the stated linear insertion model. Its empirical consequences depend on the architecture and the learned update.

The rank theorem has an observable algebraic counterpart. For any gated execution map $H : \{0,1\}^K \to \mathcal{V}$ into a vector space, define

$$\mathcal{I}_S[H] = \sum_{T \subseteq S} (-1)^{|S|-|T|} H(\mathbf{1}_T), \qquad S \subseteq [K],$$

where $\mathbf{1}_T$ activates exactly the visits in T.

Proposition 1 (Execution-interaction decomposition)**.** *Every gated execution satisfies the exact reconstruction*

$$H(\mathbf{1}) - H(\mathbf{0}) = \sum_{\emptyset \neq S \subseteq [K]} \mathcal{I}_S[H].$$

For the linear recurrent map $H(c) = (A + c_K\Delta)\cdots(A + c_1\Delta)$, $\mathcal{I}_S[H]$ is precisely the ordered noncommutative word that places Δ at visits in S and A elsewhere. It is nonzero exactly when that ordered interaction survives.

Proof. The first identity is Boolean-lattice Möbius inversion. The linear H is multi-affine in its gates, so inversion extracts the coefficient of $\prod_{j \in S} c_j$, namely the stated ordered word. □

For a nonlinear network, $\mathcal{I}_S[H]$ remains an exact finite-difference interaction rather than a matrix word. Terms with $|S| \geq 2$ show that the response cannot be reconstructed by adding isolated one-visit effects. We use the complete 2^K intervention as an execution diagnostic for recurrent TTT; the learned-update and schedule controls connect those interactions to task utility.

Writing $\Delta = UV^\top$ also gives

$$\mathrm{col}((A + \Delta)^K - A^K) \subseteq \mathcal{K}_K(A, U).$$

Kr is an upper envelope and may be loose after Krylov collapse. Numerical controllability measures available execution directions; task performance also depends on which direction TTT learns.

The theorem gives the linear capacity envelope, while the LoopLM experiments test the nonlinear execution algebra directly. Appendix B.1 gives additional local sensitivity diagnostics for shared parameters.

4 Test-Time Compiled Loops

TTCL instantiates this view. We call the adapter factors produced by episode-level optimization a *compiled adapter*: they parameterize the task-conditioned transition reused at every visit. Conventional TTT optimizes an episode-specific parameter and uses the resulting model once. TTCL treats that parameter as a recurrent transition, compiled once and invoked at every visit to the shared block.

Let F_θ be a weight-tied Transformer block. We insert the same task-specific rank-r adapter into selected projections at every recurrence:

$$W_\tau = W_0 + U_\tau V_\tau^\top, \qquad h_{k+1} = F_{\theta, W_\tau}(h_k, x).$$

At the start of every episode, (U_τ, V_τ) is reset and optimized from the demonstrations. The LoopLM experiments use 20 Adam steps; controlled experiments use the settings stated with their respective protocols. The base parameters remain frozen. Unlike depth-specific adapters, a compiled adapter is identical at every visit to the physical block. Its storage cost is independent of execution depth even though its effect is repeatedly transported through a changing hidden trajectory.

The present experiments use direct gradient-based TTT on demonstration loss. An amortized compiler that predicts adapter factors would be a natural extension of the same architecture.

The training loss exposes every prefix depth up to the training horizon. We do not add a Jacobian or spectral-norm penalty. Performance at longer horizons is reported separately.

5 EXPERIMENTS

5.1 CLAIMS AND TESTS

Table 2: Claims and their direct empirical or analytical tests.

Claim	Direct test
Rank–depth capacity	Linear tight construction and single-insertion lower bound
Repeated execution	Identical shared/single schedules, visit removal, and equal-dose order tests
Useful compilation	Held-out-rule accuracy and learned/random/shuffled controls

5.2 TASKS AND BASELINES

We evaluate linear system identification and formula tasks on LoopLMs. We compare no adaptation, extra loops only, single-use low-rank adaptation, shared TTCL, and a full-rank residual adaptation upper bound. The rule experiments fix the architecture, adaptation steps, and adapter parameterization, and report task-specific parameter counts. A separate CUDA microbenchmark records adaptation and execution wall-clock time. The nonlinear non-looped LoRA-TTT control appears in Appendix A.2. Both the linear system-identification control and the nonlinear teacher control include independent per-depth adapters, whose parameter cost grows as K.

The main split holds out complete rules. We extrapolate sequence length and loop depth separately, and select hyperparameters before opening the confirmatory test split.

5.3 CONTROLLED MECHANISM CHECKS

Before the LoopLM study, we verify the measurements where ground truth is available (Appendix A). At $K = 16$, a shared rank-2 update fitted from one-step linear demonstrations reaches 3.44×10^{-5} relative MSE and executed rank 6, versus 9.50×10^{-2} and rank 2 when the identical update is used once. Removing any recurrent contribution raises error. In a frozen nonlinear attention–MLP teacher, reuse expands the independent-control diagnostic from rank 3 to 17.5 at $K = 8$ and lowers relative MSE from 9.67×10^{-2} to 2.28×10^{-7}; every leave-one-visit intervention hurts. On Ouro-1.4B, the corresponding rank-2 probe expands to ranks 2, 4, and 8. On Ouro-2.6B, it reaches ranks 2, 4, and 7 at threshold 10^{-5}, and 8 at 10^{-6}. The TTT interventions below test whether learned directions are useful.

5.4 PUBLIC DOWNSTREAM CHECKS

We first validate the mechanism outside language models. On NumSeqBench, a learned rank-2 recurrent compiler trained over 80 formula groups reaches 93.3% exact-match accuracy on held-out arithmetic rules, compared with 0% when the same compiled update is applied only once. This is the query-level mean over 45 evaluations spanning 11 held-out formulas (15 queries per seed); shared exact-match accuracy is 93.3% in each of three seeds. The LoopLM study below uses a stricter, formula-aware protocol. A complementary semantic-state audit fixes the compiled update at rank one while scaling the state dimension from 5 to 9 and 17. Shared execution is exact through $K = 32$; at $d = 17$ its effect reaches rank 16, which an optimal one-shot approximation must match (Appendix B.1 and Proposition 3).

5.5 RANK NORMALIZATION AND INDEPENDENT OURO CONFIRMATION

The rank–depth sweep shows that an unnormalized multiplier conflates initialization gain with rank. Our shared adapter is factorized as $\Delta = UV^\top$ and initialized with $V = 0$. At the first compiler step, the right-factor gradient contains $U^\top g$; hence the induced matrix update contains $UU^\top g$. With independent columns of fixed variance, $\mathbb{E}[UU^\top]$ grows linearly with r under an isotropic, zero-mean initialization. A fixed multiplier can therefore increase the initial effective control gain as rank grows, even before the extra subspace directions are used. We compare the raw multiplier with

Table 3: Independent Ouro evaluation at $K = 4$ (hybrid objective; answer-token NLL on the held-out next-term query slice; three seeds for both checkpoints and nine held-out formula clusters).

Checkpoint	No adapt	Single-use	Shared	Depthwise
Ouro-1.4B	1.820	1.670	0.825	0.797
Ouro-2.6B	2.167	1.837	1.112	1.074

$\alpha/\sqrt{r}$ and the more conservative LoRA-style α/r scaling. Under the isotropic mean calculation above, α/r normalizes the expected first-step matrix gain. We include $\alpha/\sqrt{r}$ as a less aggressive interpolation between no compensation and exact mean-gain normalization; validation selects it for the confirmatory runs. Neither rule adds trainable parameters.

We fix square-root scaling, rank 4, 20 Adam updates with compilation horizon $K = 4$, and the three TTT objectives before evaluating an independent held-out formula split. This split contains nine arithmetic and geometric formulas excluded from exploratory evaluation. Adaptation uses either solved support rows ("sup"), pseudo next-value targets constructed only from visible query prefixes ("self"), or their hybrid. Because "self" uses the current query's visible prefix, it is a transductive query-conditioned TTT objective; supervised support remains disjoint. Formula-generated targets lie strictly outside the visible prefix; we additionally remove every query whose target position or numeric label occurs in a support row. Table 3 reports the complete query grid at $K = 4$ using the fixed hybrid objective. The paired shared–single gaps reported below are therefore hybrid-objective results. The causal schedule interventions use supervised support; self-prefix and hybrid objectives are retained as prespecified robustness ablations. Tables report row means over seeds and queries. For paired NLL intervals, we average paired differences within each formula across seeds and queries, then resample the nine formula clusters; the confirmatory generation intervals average the three support seeds within each of 256 fixed query identities and resample those query identities.

For rank 4, the formula-cluster paired shared–single gap is -0.829 (95% CI $[-1.005, -0.644]$) on 1.4B and -0.839 (CI $[-0.998, -0.674]$) on 2.6B. Shared uses 16,384 active task parameters, whereas depthwise LoRA uses 65,536; their NLL gap is only 0.028 on 1.4B and 0.038 on 2.6B (the latter interval includes zero). The separate rank–depth sweep shows saturation: increasing rank beyond 4 changes NLL little while increasing stored task parameters. Shared reuse therefore approaches depthwise adaptation with one quarter of the active task parameters in this setting.

The full $K_c \times K_e$ grid shows horizon transfer and saturation. At $K_e = 4$, 2.6B shared NLL for $K_c = 1$, 2, 4, and 8 is 1.158, 0.915, 0.877, and 0.887, respectively; the corresponding 1.4B values are 1.200, 0.890, 0.831, and 0.827. Compiling beyond four loops therefore provides no systematic gain. A schedule intervention tests which applications matter. Using supervised support, NLL for all-loop, first-only, last-only, and disabled execution is 0.830, 1.677, 1.138, and 1.866 on 1.4B; the corresponding 2.6B values are 0.877, 1.734, 1.521, and 2.102. The all-loop value is not directly comparable to the hybrid-objective value in Table 3. Dropping application 3 or 4 yields a formula-cluster bootstrap interval excluding zero at both scales, showing that later recurrent applications contribute to the effect.

Support controls separate learned adaptation from arbitrary perturbation. On 2.6B, shared NLL for one/two/four solved rows is $0.956/0.929/0.908$, versus approximately 2.07 for a norm-matched random adapter; deranging support labels worsens the two/four-shot results to $1.079/1.078$. Arbitrary updates and corrupted supervision do not reproduce the shared-execution gain. A cross-dataset intervention further shows that valid but wrong-task MATH-500 support is worse than correct support on both GSM8K and SVAMP, with prespecified query-bootstrap intervals excluding zero. Exchanging format-matched three-choice logical and object-tracking support gives task-balanced gaps of -0.103 on 1.4B and -0.259 on 2.6B, with both 95% CIs excluding zero (Appendix B.2).

5.6 The execution algebra inside LoopLMs

To characterize the computation behind the performance separation, we intervene on all 2^4 binary subsets of the four recurrent visits and apply the Möbius transform from Proposition 1 to the final query representation. This is an exact finite decomposition of the nonlinear network response. The protocol covers nine held-out formula clusters, three compiler seeds, and both checkpoints, and

correct, shuffled-label, and norm-matched random updates. For hidden interactions we report

$$\mathrm{HO}(H) = \frac{\sum_{|S|\geq 2} \|\mathcal{I}_S[H]\|_2^2}{\sum_{|S|\geq 1} \|\mathcal{I}_S[H]\|_2^2},$$

computed per query and seed before aggregation. The random control samples new rank-r factors and rescales them so $\|UV^\top\|_F$ exactly matches the compiled adapter matrix before the common injection scale. Hence norm matching refers to adapter-matrix Frobenius norm, not an assumed match of every trajectory-dependent residual.

Table 4: Execution algebra at $K = 4$ (three seeds, nine formula clusters). High-order is the fraction of hidden interaction energy at orders 2–4; reuse gain is single-use minus shared NLL. Order range permutes the matched dose $(2, 1, 1, 0)$ while preserving both total and squared dose.

	High-order fraction				
Checkpoint	Correct	Shuffled	Random	Reuse gain	Order range
Ouro-1.4B	0.240	0.181	0.663	0.838	0.169
Ouro-2.6B	0.492	0.421	0.680	0.850	0.205

Table 4 shows that correct TTT has substantial high-order response and its first-order reconstruction error is 0.503 on 1.4B and 0.734 on 2.6B, so a sum of four one-shot effects does not recover recurrent execution. Correct updates also contain more high-order energy than shuffled updates: paired formula-cluster differences are 0.057 (95% CI $[0.024, 0.093]$) and 0.075 (CI $[0.017, 0.116]$). Yet the random update has the *largest* high-order fraction and zero measurable reuse gain (-0.005 and 0.003 NLL before sign reversal). High-order interaction magnitude alone therefore does not predict a reuse gain.

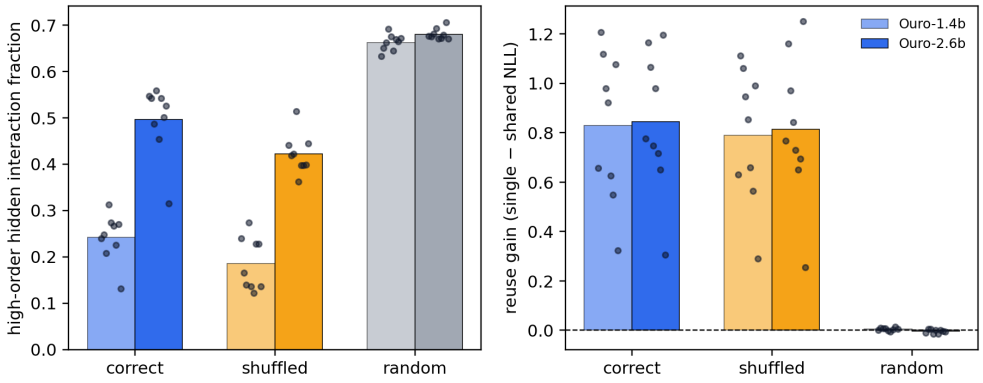

Figure 1: Execution capacity and task utility separate. Bars are three-seed means and points are formula clusters. A norm-matched random adapter produces the most high-order hidden interaction but no reuse gain; compiled updates combine nonlinear execution with useful task direction.

We also test whether recurrent execution is order-sensitive. Across all 12 distinct permutations of $(2, 1, 1, 0)$, the mean NLL range is 0.169 (95% cluster-bootstrap CI $[0.141, 0.201]$) on 1.4B and 0.205 (CI $[0.165, 0.236]$) on 2.6B. Repeating an identical schedule gives zero hidden distance. The execution depends on the placement of an adapted transition, even when the total update dose is fixed.

5.7 Family extrapolation and downstream generation

We then hold the adaptation procedure fixed and change only the held-out sequence family. We use rank 2, supervised support, $K = 4$, distances $1, 2, 4, 8, 16$, and three seeds at both scales. Shared reuse improves over single-use for every family at both checkpoints. The mean family-level shared–single NLL gap is -0.77 on 1.4B and -0.90 on 2.6B; per-family values and formula-cluster intervals are reported in Appendix B.4.

We evaluate answer-only generation on 256 fixed held-out questions and three support seeds. Both conditions use the same tokenizer, $K = 4$, greedy decoding, 16-token limit, and EOS/newline

stopping. Across GSM8K/GSM-Hard/SVAMP, shared minus single-use permissive exact-match gains are 3.13 points on 1.4B (query-bootstrap 95% CI $[1.61, 4.69]$) and 2.99 points on 2.6B ($[1.69, 4.34]$); shared minus no-adapt gains are 11.85 and 25.82 points, respectively. Per-task exact-match results are reported in Table 10. Shared minus single-use gains on 1.4B are 0.13 points for GSM8K (95% CI $[-2.34, 2.60]$), 1.04 for GSM-Hard (CI $[-0.26, 2.47]$), and 8.20 for SVAMP (CI $[4.82, 11.85]$). On 2.6B, the corresponding gains are 2.86 (CI $[0.39, 5.47]$), 1.43 (CI $[-0.13, 3.26]$), and 4.69 (CI $[2.08, 7.42]$). These intervals and Table 10 use the same 256-query split and permissive exact-match metric. Unparsable attempted generations count as errors. A separate 64-query audit (Table 11) reports strict exact-match accuracy and parse rate. Because this audit uses a different split and a strict whole-response metric, it is reported separately rather than pooled with the 256-query results. On MATH-500, where we retain teacher-forced final-answer evaluation, NLL is 1.308, 1.613, and 1.705 for shared, single-use, and no adaptation on 1.4B; the corresponding values on 2.6B are 1.228, 1.799, and 2.247.

6 Limitations

The linear result is a worst-case expressivity result for a shared linear base, not a typical-case theorem for attention. Krylov collapse can make the bound loose. Confirmatory inference uses nine formula clusters; cells with one formula are descriptive. Generation is answer-only at $K = 4$, so we report parse rate and strict as well as permissive exact match, while MATH-500 uses teacher-forced final-answer evaluation. Bootstrap intervals average three compiler seeds and are not multiplicity-adjusted over exploratory sweeps.

7 Conclusion

TTCL places a test-time update inside a tied loop and executes it repeatedly. For the linear insertion model, this produces a rank–depth separation: a rank-r update with constant task-specific storage can induce a rank-Kr finite-horizon operator effect. Visit-gating and matched-dose interventions show nonlinear, order-sensitive responses in LoopLMs, and random-adapter controls show that high-order interaction magnitude alone does not predict improvement. Shared execution improves controlled answer NLL across both evaluated checkpoints and approaches depthwise adaptation with four times the task parameters. The generation results vary by model and task, motivating further work on objectives and stability for longer-horizon adaptation.

Reproducibility Statement

Code, fixed seeds, configurations, and row-level outputs accompany the experiments. The figures are regenerated from the reported raw outputs.

AI Use Statement

Generative AI tools assisted with experiment design, code implementation, the development and checking of mathematical derivations and claims, interpretation of experimental results, and editing the wording and organization of the manuscript. The authors reviewed and verified all AI-assisted text, derivations, code, experiments, and claims, and take full responsibility for the final content.

References

Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W. Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, 2016.

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Deep equilibrium models. In *Advances in Neural Information Processing Systems*, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/01386bd6d8e091c2ab4c7c7de644d37b-Abstract.html.

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.

Sebastian Flennerhag, Andrei A. Rusu, Razvan Pascanu, Francesco Visin, Hujun Yin, and Raia Hadsell. Meta-learning with warped gradient descent. In *International Conference on Learning Representations*, 2020.

Khashayar Gatmiry, Nikunj Saunshi, Sashank J. Reddi, Stefanie Jegelka, and Sanjiv Kumar. Can looped transformers learn to implement multi-step gradient descent for in-context learning? In *International Conference on Machine Learning*, 2024.

Jonas Geiping, Sean McLeish, Neel Jain, et al. Scaling up test-time compute with latent reasoning: A recurrent depth approach. *arXiv preprint arXiv:2502.05171*, 2025.

Angeliki Giannou, Shashank Rajput, Jy-yong Sohn, Kangwook Lee, and Dimitris Papailiopoulos. Looped transformers as programmable computers. In *International Conference on Machine Learning*, 2023.

Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016. URL https://arxiv.org/abs/1603.08983.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Yuto Kojima, Jiarui Xu, Xueyan Zou, and Xiaolong Wang. LoRA-TTT: Low-rank test-time training for vision-language models. *arXiv preprint arXiv:2502.02069*, 2025. URL https://arxiv.org/abs/2502.02069.

Yoonho Lee and Seungjin Choi. Gradient-based meta-learning with learned layerwise metric and subspace. In *International Conference on Machine Learning*, 2018.

Eunbyung Park and Junier B. Oliva. Meta-curvature. In *Advances in Neural Information Processing Systems*, 2019.

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers. In *International Conference on Machine Learning*, 2021.

Jürgen Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992.

Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei A. Efros, and Moritz Hardt. Test-time training with self-supervision for generalization under distribution shifts. In *International Conference on Machine Learning*, 2020.

Yu Sun, Xinhao Li, Karan Dalal, Jiarui Xu, Arjun Vikram, Genghan Zhang, Yann Dubois, Xinlei Chen, Xiaolong Wang, Sanmi Koyejo, Tatsunori Hashimoto, and Carlos Guestrin. Learning to (learn at test time): Rnns with expressive hidden states. *arXiv preprint arXiv:2407.04620*, 2024.

Johannes Von Oswald, Eyvind Niklasson, Ettore Randazzo, Joao Sacramento, Alexander Mordvintsev, Andrey Zhmoginov, and Max Vladymyrov. Transformers learn in-context by gradient descent. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 35151–35174. PMLR, 2023. URL https://proceedings.mlr.press/v202/von-oswald23a.html.

Dequan Wang, Evan Shelhamer, Shaoteng Liu, Bruno Olshausen, and Trevor Darrell. Tent: Fully test-time adaptation by entropy minimization. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=uXl3bZLkr3c.

Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. DeeBERT: Dynamic early exiting for accelerating BERT inference. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 2246–2251, 2020. doi: 10.18653/v1/2020.acl-main.204. URL https://aclanthology.org/2020.acl-main.204/.

Liu Yang, Kangwook Lee, Robert Nowak, and Dimitris Papailiopoulos. Looped transformers are better at learning learning algorithms. In *International Conference on Learning Representations*, 2024.

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. URL https://arxiv.org/abs/2510.25741.

A Controlled Mechanism Checks

A.1 Linear instrumentation check

Before training a nonlinear model, we test whether the proposed measurements recover the known mechanism in a controlled system-identification problem. We sample held-out transitions $B_\tau = A + \Delta_\tau$ with $\mathrm{rank}(\Delta_\tau) = 2$. A compiler observes only one-step pairs $(x, B_\tau x)$ and fits either a rank-2 update or an unrestricted matrix. We then evaluate the transition for horizons that never appear in the compilation loss. Table 5 reports three-seed means at $K = 16$.

Table 5: Controlled linear instrumentation check at horizon $K = 16$. Shared and single-use rank-2 methods use the same compiled matrix and the same 64 task-specific parameters; only recurrent reuse differs.

Method	Relative MSE ↓	Effective rank	Task parameters
No adaptation	1.083×10^{-1}	0.0	0
Single-use rank-2	9.501×10^{-2}	2.0	64
Shared rank-2	3.437×10^{-5}	6.0	64
Shared full-rank	1.709×10^{-5}	13.4	256

This result validates the instrumentation in a task generator that satisfies the assumed low-rank recurrent model. Held-out-rule experiments test whether the mechanism also appears in learned systems.

As a simplicity control, we independently compile a rank-2 adapter for each physical depth from the same one-step demonstrations. In the three-seed linear run, depthwise relative MSE is below 10^{-6} through $K = 8$ (mean 6.3×10^{-7} at $K = 8$, versus 3.9×10^{-6} for shared), but its task-specific parameter count grows from 64 to $64K$. We keep it as a linear control and separately test the same parameter-efficiency tradeoff in the nonlinear teacher block. Across three seeds, nonlinear depthwise relative MSE at $K = 8$ is 2.71×10^{-6} versus 2.21×10^{-5} for shared reuse, while using $64K$ rather than 64 task parameters.

We additionally perform an operator-level causal intervention. For each compiled transition $\widehat{B} = A + \widehat{\Delta}$, we use the exact telescoping decomposition $\widehat{B}^K - A^K = \sum_j \widehat{B}^{K-1-j} \widehat{\Delta} A^j$ and remove one summand at a time from the final map. Across three seeds, removing any one summand increased relative MSE in 100% of cases; the mean increase was 7.19×10^{-4}, 6.83×10^{-4}, 5.90×10^{-4}, and 4.23×10^{-4} at $K = 2, 4, 8, 16$, respectively. This indicates that the propagated directions carry task signal in the controlled linear setting. The following nonlinear teacher and held-out-rule interventions test the same idea without assuming an exact linear decomposition.

We next repeat the check with a frozen nonlinear pre-norm block containing multi-head softmax attention, an MLP, normalization, and residual connections. Held-out tasks are rank-1 residual injections at the output of the block, and the adapter is again compiled only from one-step demonstrations. With three tokens, the single-use local controllability rank is three. At $K = 8$, recurrent reuse increases its three-seed mean to 17.5; relative MSE is 2.283×10^{-7} compared with 9.667×10^{-2} for using the identical compiled adapter once and 1.272×10^{-1} without adaptation. This remains a teacher-generated mechanism check rather than evidence on naturally arising rules, but it verifies the finite-horizon independent-control diagnostic $\mathcal{U}_K$ defined above.

We also apply a direct nonlinear intervention: on each shared rollout, we skip the compiled adapter at exactly one loop and compare with the intact rollout. Across three seeds and all task instances,

every skip increased relative MSE at $K = 2, 4, 8$ (100% positive interventions), with mean increases 4.39×10^{-3}, 4.00×10^{-3}, and 2.98×10^{-3}, respectively. Thus propagated adapter use is causally necessary in this nonlinear teacher setting. Appendix A.2 reports the same leave-one-loop test on a task-trained held-out affine-rule core.

A.2 Untied and held-out-rule controls

As an architecture stress test, we remove weight tying with a four-block nonlinear Transformer stack while holding the rank-4 task adapter, demonstrations, a 400-step optimizer in this synthetic setting, and one-step transition demonstrations fixed. Table 6 reports three-seed held-out affine-rule accuracy. The untied adapter reproduces part of the shared-use gain at $K = 2$, but its benefit collapses by $K = 4$, where no-adaptation is strongest; the tied model retains a positive shared advantage at both horizons.

Table 6: Held-out affine-rule accuracy with a tied recurrent core versus a higher-base-parameter untied four-block stack (mean ± SD over three seeds). Both adaptive methods use 128 task parameters; the backbones are not function-matched.

Architecture	Horizon	No adapt	Single use	Shared use
Tied loop	2	$0.240 \pm .144$	$0.107 \pm .046$	$\mathbf{0.640 \pm .040}$
Untied stack	2	$0.160 \pm .040$	$0.093 \pm .023$	$0.533 \pm .061$
Tied loop	4	$0.200 \pm .080$	$0.147 \pm .023$	$\mathbf{0.293 \pm .092}$
Untied stack	4	$\mathbf{0.347 \pm .115}$	$0.133 \pm .023$	$0.173 \pm .023$

On the tied held-out-rule checkpoints, removing one shared adapter application raises cross-entropy in 100% of horizon-2 interventions and 91.7% at horizon 4. At horizons 2 and 4, mean full accuracy is 0.600 and 0.320; the corresponding intervened accuracy is 0.120 and 0.143. These measurements indicate task-conditioned multi-visit composition.

A.3 Mechanistic measurements

For nonlinear models we estimate the tangent effect of adapter parameters on the final hidden state through a finite-horizon controllability matrix along the unadapted trajectory and report its numerical rank and conditioning. The adapter is injected at the recurrent block output so that its instantaneous image is explicitly low rank; placing it inside input-dependent attention can destroy the one-step rank premise and is treated as a separate ablation. In controlled tasks with known transition rules, we separately measure one-step rule storage and repeated composition. The linear, teacher-block, and held-out-rule leave-one-loop tests above provide causal checks at three levels of realism. A support-conditioned compiled transition should change intermediate computation; output-only calibration would not establish the stronger task-program claim.

A.4 Scale check on released Ouro LoopLMs

We repeat the mechanism measurement on the released Ouro-1.4B and Ouro-2.6B LoopLM checkpoints (Zhu et al., 2025). Both use a 2,048-dimensional hidden state and four recurrent steps, with 24 and 48 shared Transformer layers, respectively. A fixed rank-2 adapter injected after each recurrent block gives a measured hidden-state operator response of numerical rank 2 at every horizon. Independent per-step coefficients give ranks 2, 4, and 8 at $K = 1$, 2, and 4 on Ouro-1.4B. Ouro-2.6B gives ranks 2, 4, and 7 at a relative SVD threshold of 10^{-5}, and rank 8 at 10^{-6}. This scale check supports the algebraic observable beyond the 10M checkpoint; the constant shared rank depends on the residual insertion point and measurement convention.

B Additional Parameter–Performance Results

B.1 Shared-parameter sensitivity

The following local diagnostic distinguishes independent per-visit controls from a genuinely shared TTT parameter. Along a nominal trajectory $z_{k+1} = f(z_k)$ with $J_k = Df(z_k)$, independent injections

Table 7: Distinct rank objects; only the first is the executed operator in Theorem 1.

Quantity	Object	Can grow with K?
$R_K(A, \Delta)$	executed operator difference	yes, up to Kr
$\mathrm{rank}(\mathcal{U}_K)$	independent-control Jacobian	yes, up to Kr
$\mathrm{rank}(D_\phi z_K)$	shared-parameter Jacobian	no, at most p
residual intervention response	finite hidden-state intervention	intervention-dependent

$z_{k+1} = f(z_k) + Uu_k$ have final-state Jacobian

$$\mathcal{U}_K = [J_{K-1} \cdots J_1 U, \ldots, J_{K-1} U, U], \qquad \mathrm{rank}(\mathcal{U}_K) \le \min\{d, Kr\}.$$

This is the diagnostic upper envelope measured by independent-control probes; it is not the parameter Jacobian of a shared adapter.

Proposition 2 (Shared-parameter boundary)**.** *Let $\phi \in \mathbb{R}^p$ be reused at every visit and suppose $f_\phi(z) = f(z) + G(z)\phi + o(\|\phi\|)$. With $G_j = G(z_j)$ and $\Phi_{K,j+1} = J_{K-1} \cdots J_{j+1}$ (with an empty product equal to I),*

$$D_\phi z_K|_{\phi=0} = \sum_{j=0}^{K-1} \Phi_{K,j+1} G_j, \qquad \mathrm{rank}(D_\phi z_K) \le \min\{d, p\}.$$

Recurrent reuse can change the realized function through composition, but it does not manufacture Kp independent parameter degrees of freedom.

Proof. Differentiate the inhomogeneous tangent recurrence and unroll it. The rank bound follows because the resulting Jacobian has only p columns. □

Proposition 3 (Finite-order recurrences are rank-one programs)**.** *For any affine recurrence $v_{t+1} = \sum_{j=0}^{p-1} a_j v_{t-j} + b$, there is a task-independent $(p+1)$-dimensional delay-shift operator A and a task-specific update Δ of rank at most one such that $z_{t+1} = (A + \Delta) z_t$ for $z_t = [v_t, \ldots, v_{t-p+1}, 1]^\top$.*

Proof. Let A shift the first p coordinates down by one and preserve the final homogeneous coordinate. Setting $\Delta = e_1[a_0, \ldots, a_{p-1}, b]$ gives the stated recurrence in its first coordinate and leaves the shift unchanged. The outer-product form has rank at most one, and repeated application therefore gives every finite horizon. □

B.2 NumSeqBench compiler protocol

The NumSeqBench result uses released arithmetic rows and a two-dimensional homogeneous state $[v, 1]$. A shared affine recurrent core is meta-trained for 1,000 Adam steps (learning rate 2×10^{-3}) jointly with one rank-2 adapter per training formula. We select 80 formula groups before evaluation, use the first 60 only for this training, and freeze the recurrent core. For each held-out formula, a fresh rank-2 adapter is initialized with a random left factor and zero right factor, then optimized for 200 steps at learning rate 3×10^{-2} using transitions extracted from two support rows. Queries use disjoint rows and horizons 1–8; 11 held-out formulas yield 15 admissible query evaluations per seed. The identical compiled matrix is then evaluated either once or at every recurrent visit. No language model, amortized adapter predictor, query label, or test-formula core training is used. Thus "learned compiler" here means a meta-trained shared interpreter followed by genuine episode-level gradient TTT of fresh adapter factors. Rank 2 is full rank in this two-dimensional state, so this experiment isolates exact held-out-rule composition rather than providing the low-rank evidence; the rank separation and the $r \ll d$ released-model tests serve that role.

We add a delay-coordinate audit with a five-dimensional state $[v_t, v_{t-1}, v_{t-2}, v_{t-3}, 1]$; a fixed formula-independent companion core shifts the four semantic history coordinates, while demonstrations optimize only $\Delta = e_1 w^\top$, so $\mathrm{rank}(\Delta) = 1 < 5$. Starting at the end of each visible prefix, shared execution is exact on all 12 held-out arithmetic queries requiring two to 15 future steps, whereas applying the identical update once is exact on 0/12; mean relative error is 2.0×10^{-6} versus

0.436. Horizon-1 queries are excluded from this contrast because shared and single-use coincide by construction. This audit uses an explicit delay-state interpreter and provides a controlled test of low-rank repeated execution.

Figure 2 generalizes the audit to dense stable autoregressions of orders 4, 8, and 16. Every lag coefficient is nonzero and task-specific; support contains $8d$ one-step transitions and evaluation uses 128 fresh states for each of 64 held-out rules per dimension and three seeds. The interpreter only shifts lag history, while least-squares TTT estimates the rank-one first row. Optimal rank-q single-use controls are computed by truncated SVD of the exact finite-horizon operator difference, providing an oracle upper bound.

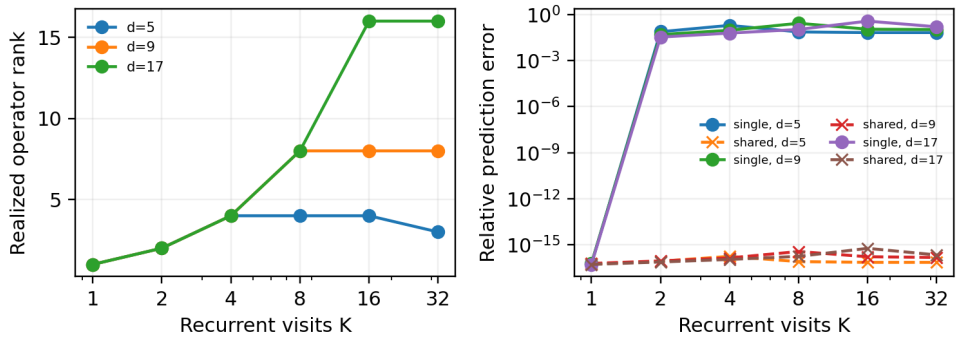

Figure 2: A fixed rank-one compiled transition scales through semantic state dimension and recurrent runtime. Its executed operator rank grows with visits (left), while shared prediction remains at numerical precision and the identical update used once does not compose the recurrence (right).

B.3 Support-specificity audit

We additionally replace solved supports by examples from another formula or the other arithmetic/geometric family. Shared minus cross-family NLL is 0.005 (95% CI $[-0.067, 0.088]$) on 1.4B and 0.008 (CI $[-0.062, 0.100]$) on 2.6B. These intervals do not provide evidence for formula-specific adaptation within this sequence domain. They are consistent with transferable components in the update. Evidence for held-out-rule generalization comes from NumSeqBench, while the random and shuffled-label controls test whether arbitrary or corrupted updates suffice.

We also intervene across natural-language datasets on Ouro-2.6B, holding the 64 queries, canonical answer format, eight support examples, and three support seeds fixed. Replacing correct support by valid but wrong-task MATH-500 support worsens shared-TTCL answer NLL by 0.0391 on GSM8K (query-bootstrap 95% CI $[0.0231, 0.0561]$) and 0.0556 on SVAMP (CI $[0.0411, 0.0707]$). A closer GSM8K–SVAMP exchange is directional: SVAMP-specific support wins by 0.0281 ($[0.0097, 0.0454]$), whereas GSM8K is tied (0.00135, $[-0.0126, 0.0157]$). These results show clear degradation for the MATH-500 replacement, but mixed evidence for the closer GSM8K–SVAMP exchange.

Finally, we remove output-format and label-frequency confounds using two three-choice BBH capabilities: logical deduction over ordered objects and tracking objects through swaps. For each direction we fix 128 queries, match the eight support labels and format, average three independently compiled support sets, and bootstrap query identities. For this cross-capability experiment, validation selects scale 0.3 before the test indices are evaluated. The formula experiments use the separate $0.1/\sqrt{r}$ scale in Table 9. Task-balanced correct-minus-cross-task NLL is -0.103 (CI $[-0.133, -0.071]$) on 1.4B and -0.259 (CI $[-0.334, -0.182]$) on 2.6B. Cycling every support label gives gaps -0.138 ($[-0.202, -0.072]$) and -0.292 ($[-0.360, -0.222]$). The compiled transition is sensitive to capability and supervision. Shared execution also beats the identical single-use update in all four model–task cells.

B.4 FAMILY EXTRAPOLATION DETAILS

Table 8 gives the per-family results summarized in Section 5.7. We hold rank 2, supervised support, $K = 4$, distances $1, 2, 4, 8, 16$, and three seeds fixed, and change only the held-out sequence family. Entries are formula-cluster bootstrap gaps (shared minus single-use NLL). The triangular family has one available formula per checkpoint, so no cluster interval is defined.

Table 8: Held-out sequence-family extrapolation. Negative shared–single NLL favors shared reuse.

Family	Ouro-1.4B	Ouro-2.6B
Fibonacci	-0.902 $[-0.994, -0.788]$	-1.125 $[-1.269, -1.019]$
Power	-0.406 $[-0.572, -0.239]$	-0.575 $[-0.838, -0.312]$
Quadratic	-1.002 $[-1.228, -0.828]$	-1.073 $[-1.232, -0.901]$
Triangular	-0.752	-0.824

B.5 CONFIRMATORY EVALUATION PROTOCOL

Table 9 gives the implementation choices for the confirmatory Ouro experiments. Each episode resets adapter factors and Adam state; only the factors receive gradients. Held-out query answers are never used for adaptation.

Table 9: Confirmatory protocol for the LoopLM evaluation.

Component	Setting
Backbone / insertion	Frozen Ouro-1.4B or 2.6B; token-wise residual insertion at the recurrent final-normalization boundary; the same insertion is reused at every visit.
Adapter	$\Delta = UV^\top$, rank 4; random zero-mean U, $V = 0$; injection multiplier $0.1/\sqrt{r}$.
Test-time optimizer	Adam, learning rate 10^{-3}, 20 steps, adaptation horizon 4; factors and optimizer state reset for every episode.
Objectives	"sup": two solved support rows; "self": two targets formed from visible query prefixes; "hybrid": supervised loss plus 0.5 times the self-supervised loss.
Confirmation split	Nine held-out arithmetic/geometric formulas disjoint from exploratory evaluation; three support/adaptation seeds; held-out query targets and matching support labels removed.
Execution controls	No adaptation; identical adapter first-use only; identical shared adapter at every visit; four independent depthwise adapters; shuffled and norm-matched random support controls.
Inference	Formula-cluster bootstrap after averaging queries and seeds within formula; confirmatory generation uses 256 fixed queries, greedy decoding, 16 new tokens, and query-identity bootstrap after averaging three support seeds.

C GENERATION AND PARSING AUDIT

Table 10: Answer-only generation on the 256-query evaluation split. Entries are permissive exact-match accuracy (%) averaged over three support seeds.

Model	Task	No adaptation	Single-use	TTCL
1.4B	GSM8K	6.3	9.2	9.4
	GSM-Hard	2.0	3.8	4.8
	SVAMP	27.7	49.1	57.3
2.6B	GSM8K	2.0	13.8	16.7
	GSM-Hard	0.4	4.7	6.1
	SVAMP	9.8	62.1	66.8

Table 11: Answer-only generation results on a separate 64-query split at $K = 4$ (three support seeds). Prompt receives the same eight solved examples used by the TTT compiler. Parsed accuracy conditions on a numeric parse; strict exact requires the whole response to be numeric.

Model	Task	Method	Exact	Parse	Parsed acc.	Strict exact
1.4B	GSM8K	Prompt	0.115	1.000	0.115	0.115
		No adapt	0.078	0.484	0.161	0.016
		Single	0.130	0.693	0.188	0.068
		TTCL	0.099	1.000	0.099	0.042
	GSM-Hard	Prompt	0.089	1.000	0.089	0.089
		No adapt	0.031	0.516	0.061	0.016
		Single	0.042	0.667	0.063	0.026
		TTCL	0.057	1.000	0.057	0.042
	SVAMP	Prompt	0.542	1.000	0.542	0.526
		No adapt	0.219	0.391	0.560	0.203
		Single	0.401	0.818	0.490	0.354
		TTCL	0.552	1.000	0.552	0.479
2.6B	GSM8K	Prompt	0.026	0.177	0.147	0.026
		No adapt	0.000	0.219	0.000	0.000
		Single	0.099	0.708	0.140	0.078
		TTCL	0.167	1.000	0.167	0.120
	GSM-Hard	Prompt	0.026	0.120	0.217	0.026
		No adapt	0.016	0.250	0.063	0.016
		Single	0.063	0.734	0.085	0.047
		TTCL	0.083	1.000	0.083	0.057
	SVAMP	Prompt	0.375	0.521	0.720	0.375
		No adapt	0.047	0.281	0.167	0.031
		Single	0.651	0.958	0.679	0.609
		TTCL	0.672	1.000	0.672	0.641