T-REX: Tied Recurrence Extraction

Mozes Jacobs [1] **T. Andy Keller** [1] **Thomas Fel** [2 1] **Bingbin Liu** [1] **Richard Hakim** [1] **Yilun Du** [1] **Demba Ba** [1]

Abstract

Recurrent and looped transformers offer a promising inductive bias for reasoning by reusing the same computation across depth, but exact weight tying can make these models difficult to optimize. We introduce *Tied Recurrence Extraction* (T-REX), a simple training framework that bridges untied and tied transformers. T-REX trains layers with independent parameters and periodically projects them toward their shared parameter-wise mean, allowing optimization to benefit from untied flexibility while encouraging convergence to a recurrent computation. At inference time, the learned parameters are projected to their mean, yielding an exactly tied recurrent model. We show that parameter groups converge to their across-layer mean under learning-rate decay and illustrate how its temporary untying can help escape local minima that trap tied models. Empirically, T-REX improves out-of-distribution generalization on algorithmic reasoning tasks including soft distance propagation, maze solving, and in-context graph traversal, where it learns more stable iterative rules than both untied and tied-weight transformer baselines.

1. Introduction

Many reasoning problems have an inherently iterative structure: search, planning, path finding, arithmetic, and dynamic programming are often solved by repeatedly applying the same update rule until a solution is reached. This observation has motivated interest in recurrent and looped transformer architectures, which reuse the same block across multiple computation steps (Dehghani et al., 2018; Saunshi et al., 2025; Zhu et al., 2025). By sharing parameters across depth, these models impose a *weight symmetry over depth* – each computation step is governed by the same transformation. This symmetry provides an inductive bias toward algorithmic computation (Saunshi et al., 2025) and offers a natural way to extend test-time computation (Prairie et al., 2026; Geiping et al., 2025) beyond the training horizon.

However, training recurrent transformers remains difficult. Exact weight tying enforces this depth-wise weight symmetry throughout optimization. While appealing at convergence, this constraint can make training brittle: different depths may need to specialize early before consolidating into a shared algorithm. Untied transformers avoid this optimization difficulty, but lack the recurrence bias needed for robust reasoning (Saunshi et al., 2025). This creates a tension between optimization and generalization: untied models are easier to train, while tied models better match the structure of iterative reasoning.

We propose *Tied Recurrence Extraction* (T-REX), a simple training framework that bridges this gap. T-REX trains layers with independent parameters, but periodically projects them toward their shared parameter-wise mean. This lets the model explore the richer untied parameter space during optimization while softly encouraging depth-wise weight symmetry. At inference time, the learned parameters are projected to their mean, yielding an exactly tied recurrent model. This is related in spirit to relaxed constraint enforcement and parameter averaging used to simplify optimization of equivariant neural networks (Izmailov et al., 2018; Manolache et al., 2026), but applies the idea across depth to extract recurrence. Empirically, we find that this soft recurrence prior improves out-of-distribution generalization on algorithmic tasks, with T-REX outperforming both standard transformers and exactly tied baselines in long-horizon or larger-scale regimes. Concretely, our contributions are:

- **A soft tying framework for recurrent transformers.** We introduce T-REX, which trains layers independently while periodically projecting them toward a shared recurrent prototype, softly encouraging weight symmetry over depth and yielding an exactly tied model at inference.
- **Theory and intuition.** We prove convergence to the across-layer mean under learning-rate decay and provide a simple example showing how T-REX can avoid local minima that trap exactly tied models.
- **Improved out-of-distribution generalization.** We show that T-REX outperforms both untied transformers and exactly weight-tied transformers on algorithmic tasks that require generalization beyond the training distribution.

[1]Kempner Institute, Harvard University [2]Goodfire AI. Correspondence to: Mozes Jacobs <mozesjacobs@g.harvard.edu>.

Workshop on Weight-Space Symmetries, held in conjunction with the 43rd International Conference on Machine Learning, Seoul, South Korea. 2026. Copyright 2026 by the author(s).

2. Related Work

Recurrent and looped models have long been studied as architectures for iterative computation, with recurrence providing a natural inductive bias for repeatedly applying the same update rule (Greff et al., 2016; Zhu et al., 2025). This perspective has recently been used for state tracking (Liu et al., 2023), reasoning (Saunshi et al., 2025), and language modeling with recurrent Transformers (Oncescu et al., 2026). More broadly, work on neural algorithmic reasoning studies whether neural networks can learn procedures that extrapolate beyond the training distribution (Schwarzschild et al., 2021; Bansal et al., 2022). Recent evidence also suggests that even untied feedforward vision transformers can develop recurrent structure across depth (Jacobs et al., 2025).

Looped transformers make this recurrence explicit by reusing the same block across depth, which is appealing for reasoning and adaptive or test-time computation (Saunshi et al., 2025; Dehghani et al., 2018; Geiping et al., 2025). Recent work studies how to stabilize inference-time scaling and characterize compute scaling in such models (Prairie et al., 2026). However, exact recurrence inherits some of the optimization difficulties of recurrent networks (Pascanu et al., 2012). Other methods ease deep-network optimization by gradually growing models, for example through layer stacking (Saunshi et al., 2024; Panigrahi et al., 2024). T-REX instead keeps the full-depth model fixed but relaxes the tying constraint during training.

Our approach is also related to parameter averaging and relaxed constraint enforcement. Stochastic weight averaging averages weights across training iterates (Izmailov et al., 2018), whereas T-REX averages across depth to encourage convergence to a shared recurrent prototype. Most related is work on equivariant models that trains with relaxed symmetry constraints and enforces hard constraints at inference time (Manolache et al., 2026; Pertigkiozoglou et al., 2024). T-REX applies this principle to symmetry over depth.

3. T-REX

Below, we introduce T-REX and its convergence properties.

3.1. Method

T-REX. Let $\{\theta_1, \ldots, \theta_L\}$ be L groups of parameters for which we wish to impose a recurrent bias. We focus on the simple case where each group is a single transformer layer but note that T-REX readily extends to θ_i denoting blocks of layers. Every N optimization steps, we compute the mean $\bar{\theta}$ and project each layer i toward this shared barycenter:

$$\theta_i \leftarrow (1-\alpha)\theta_i + \alpha\bar{\theta}, \qquad \bar{\theta} = \frac{1}{L}\sum_{i=1}^{L}\theta_i. \tag{1}$$

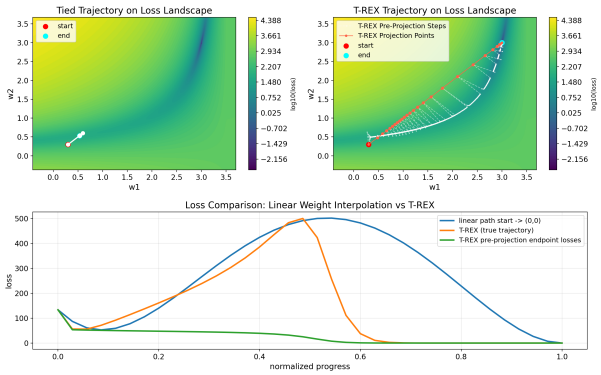

Figure 1. **T-REX escapes a spurious weight-tied local minimum in a ReLU toy problem.** The tied model is constrained to the diagonal and cannot move directly toward the better solution without temporarily increasing loss. T-REX performs untied gradient steps between projections, allowing it to use off-diagonal descent directions before returning to the recurrent subspace.

When $\alpha = 1$, this reduces to full projection onto a shared parameter vector (hard tying). When $0 < \alpha < 1$, the update is softer and can be viewed as an EMA-style pull toward a recurrent prototype. Thus, every layer remains independently trainable, but all are softly anchored to a common barycenter. The projection frequency N controls how strongly this prior is enforced over training. The default setting $N = 1$ with small α applies a gentle pull toward the mean after every gradient step and acts as a lightweight regularizer throughout optimization.

T-REX can be viewed as an algorithm for projected optimization onto the space of recurrent programs. Intuitively, T-REX encourages the model to reuse a similar transformation across depth without requiring exact weight tying. This may improve optimization and generalization by reducing layer-wise drift while preserving enough flexibility for mild layer-specific specialization. Untied parameters are allowed to evolve freely and explore the optimization landscape while still encouraging approximate recurrence overall. At inference, one can project the weights to their true mean, recovering a true weight-tied model. In Proposition 3.1, we show that under bounded update directions and a vanishing learning-rate schedule, the recurrent parameter groups converge to their across-layer mean, implying such a final projection will result in negligible performance change.

In practice, for momentum-based optimizers, we can also apply the projection to the optimizer state; details are provided in Appendix C.

Proximal gradient interpretation. T-REX can alternatively be viewed as a proximal step for a quadratic penalty measuring distance from the tied subspace. We provide a full derivation in Appendix A.

3.2. Convergence

We show that, under a decaying learning rate, T-REX drives all recurrent parameter groups to their across-layer mean.

Proposition 3.1 (Convergence to the mean)**.** *Consider T-REX with projection strength $\alpha \in (0, 1]$ applied after every update ($N = 1$). If the learning rates satisfy $\eta_k \to 0$, for step k as $k \to \infty$, and the optimizer update directions are uniformly bounded, then, with $\bar{\theta}^k := \frac{1}{L}\sum_{j=1}^{L} \theta_j^k$,*

$$\max_{i\in[L]} \|\theta_i^k - \bar{\theta}^k\|_2 \to 0.$$

Equivalently, $\|\theta_i^k - \theta_j^k\|_2 \to 0$ for all $i, j \in [L]$.

Intuitively, the projection step contracts inter-layer disagreement by a factor of $(1 - \alpha)$ at each step, while the gradient update perturbs disagreement by only $O(\eta_k)$. As the learning rate decays, the contraction dominates, so any final hard mean projection used at inference becomes asymptotically negligible. The proof is given in Appendix B.

3.3. T-REX Enables Flexible Optimization Trajectory

To illustrate how T-REX can escape optimization artifacts induced by weight tying, in Figure 1 we consider a two-weight ReLU regression problem,

$$\mathcal{L}(w_1, w_2) = \frac{1}{2}\sum_i (w_2 ReLU(w_1 x_i + b) - y_i)^2$$

The recurrent, weight-tied model is constrained to the diagonal $w_1 = w_2 = w$, whereas T-REX temporarily trains w_1 and w_2 independently before projecting them back to the tied subspace by averaging, $(w_1, w_2) \leftarrow \left(\frac{w_1+w_2}{2}, \frac{w_1+w_2}{2}\right)$.

We construct the data so that tied training from $w_0 = 0.3$ is attracted to a poor small-weight basin, while a better tied solution exists near $w_1 = w_2 = 3$. Moving between these solutions along the tied diagonal requires first increasing the loss, so ordinary tied gradient descent cannot follow this linear path. T-REX avoids this barrier by moving off the diagonal during its untied inner updates, using descent directions unavailable to the tied model, and then projecting this progress back onto the recurrent subspace.

Figure 1 shows the resulting trajectories. The tied model stalls near the spurious small-weight basin, while T-REX repeatedly leaves the tied manifold, projects back by averaging, and eventually moves toward the better tied solution near $w_1 = w_2 = 3$. The full experimental setup can be found in Appendix D.1

4. Experiments

We evaluate T-REX on three algorithmic reasoning settings: soft distance propagation, maze solving, and in-context graph traversal. These tasks are designed to test whether a model learns an iterative computation that continues to work beyond the training horizon, rather than merely fitting the finite-depth behavior seen during training. In each setting we compare T-REX against a standard (untied) transformer (referred to as Baseline) and an exactly tied (looped) baseline. Although T-REX is trained with untied weights and periodic projections controlled by N and α, at inference time we project its weights to the hard mean and run it as a tied, looped model. Full training details are in Section D.

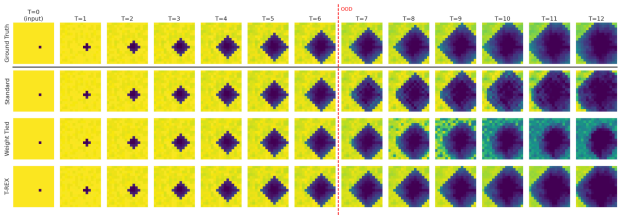

Figure 2. **T-REX learns the soft distance propagation rule more faithfully and generalizes better out of distribution than both a standard transformer and a weight-tied transformer.** All models solve the task nearly perfectly in distribution ($t = 0$ to 6), but performance degrades at longer, out-of-distribution rollout horizons. The standard transformer appears competitive in this visualization because the figure shows its best-performing seed out of three; however, this is not representative of its typical behavior, as two of the three standard-transformer seeds fail out of distribution. The weight-tied transformer also drifts from the ground-truth propagation pattern, whereas T-REX remains substantially closer to the optimal solution and has the lowest variance across seeds. See Figure 7 for the full loss curves over all three seeds.

Soft distance propagation. We first study a soft distance propagation task, where the model must iteratively expand a distance field from a source location. The ground-truth algorithm applies a softmin propagation rule at each step (see Appendix D.2 for the full recurrence). Importantly, the propagation is defined over an underlying cost field that can vary across squares, so distances are not uniform over the grid but instead reflect accumulated local traversal costs.

As shown in Figure 2, all models closely match the ground truth over the in-distribution rollout steps seen during training. However, when evaluated on longer rollouts, the standard transformer and the weight-tied transformer increasingly drift from the desired propagation pattern. Their predictions become less sharply aligned with the expanding distance field and accumulate visible errors over time.

T-REX produces substantially more faithful long-horizon rollouts and seems to learn a transformation that behaves more like a stable iterative update. This leads to better out-of-distribution generalization in time: the model remains close to the correct propagation dynamics well beyond the rollout length used during training. These results suggest that T-REX encourages the model to discover the underlying algorithmic rule, rather than memorizing a fixed-depth computation.

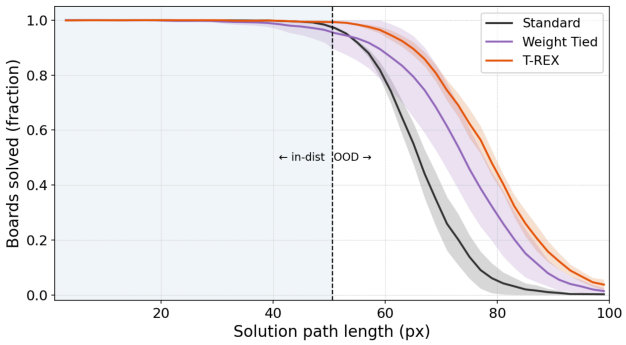

Figure 3. **T-REX generalizes substantially better than both a standard transformer and a looped transformer on maze solving at longer path lengths, especially when given additional thinking steps.** During training, models only see path lengths up to 50 pixels; evaluation includes much longer out-of-distribution paths from 50 to 100 pixels. T-REX maintains much stronger performance as path length increases and benefits more from additional thinking steps, whereas both baselines degrade rapidly on longer-horizon instances.

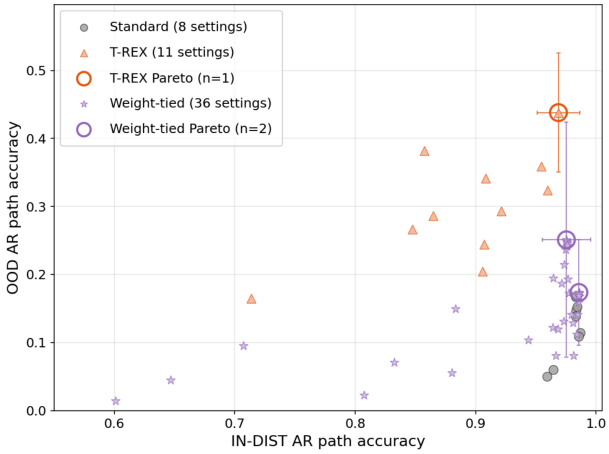

Figure 4. **T-REX generalizes much better than a standard transformer on in-context graph tasks trained autoregressively as a language model, especially out of distribution to larger graphs.** While both models perform well on the training graph sizes (cluster distance 1–2), T-REX maintains substantially stronger accuracy as graph size increases beyond the training distribution (cluster distance ≥ 3).

Maze solving. We next evaluate T-REX on maze solving, where the model must segment the shortest solution path through a maze image, as in (Schwarzschild et al., 2021; Bansal et al., 2022). Mazes are represented as 21×21 pixel grayscale images. The model is trained on mazes with path lengths up to 50 pixels, and is evaluated on substantially longer paths of up to 100 pixels. This directly tests for length generalization of the learned algorithm: a model that has learned a reusable search-like procedure should degrade gracefully as the required number of reasoning steps increases. Refer to Figure 5 for examples of in-distribution and out-of-distribution data.

Figure 3 shows that T-REX generalizes much better than both a standard transformer baseline and a weight-tied, looped transformer. While all models achieve high in-distribution performance, the standard and weight-tied baselines degrade rapidly as path length increases beyond the training distribution, whereas T-REX maintains stronger performance on longer paths. These results highlight the benefit of T-REX's soft recurrent prior: it encourages a shared update rule without the rigidity of exact weight tying, allowing better optimization for long-horizon maze solving while still yielding an exactly tied model at inference.

In-context graph traversal. Finally, we evaluate T-REX on in-context graph traversal trained autoregressively as a language model, as in (Zhang et al., 2025). The model is given a two-level Erdős–Rényi graph encoded as a sorted edge list in context, along with a source-target query, and must autoregressively generate the correct shortest path. We train on source-target pairs whose shortest paths require 1–2 hops and evaluate generalization to harder instances requiring 3–4 hops, while keeping the graph size fixed. Refer to Figure 6 for examples of in-distribution and out-of-distribution data.

As shown in Figure 4, T-REX achieves substantially better out-of-distribution accuracy than both the standard transformer baseline and the weight-tied models. While all models perform well on in-distribution path lengths, the standard transformer suffers a large drop as the required number of hops increases beyond the training regime, and the weight-tied models also lag behind T-REX on OOD path accuracy. T-REX maintains significantly higher accuracy on longer-hop instances, suggesting it learns a more robust iterative traversal procedure.

5. Conclusion

We introduced T-REX, a simple training procedure that encourages recurrent computation by promoting weight symmetry across depth without enforcing exact weight tying throughout optimization. By training layers independently and periodically projecting them toward their mean, T-REX combines the flexibility of untied models with the inductive bias of recurrent ones; at inference time, its weights are projected to their hard mean and run as a weight-tied, looped network. Across multiple algorithmic reasoning tasks, T-REX improves out-of-distribution generalization over both standard transformers and tied-weight baselines, suggesting that softly encouraging recurrence helps models learn update rules that can be iterated beyond the training horizon, especially in settings requiring repeated application of simple rules such as propagation, search, or graph traversal. Overall, T-REX provides a practical middle ground between untied and exactly tied transformers, making recurrent optimization easier to learn for algorithmic reasoning.

Impact Statement

This work studies optimization methods for transformer models on algorithmic reasoning tasks. We do not foresee direct negative societal impacts beyond those common to machine learning research.

Acknowledgements

This work has been made possible in part by a gift from the Chan Zuckerberg Initiative Foundation to establish the Kempner Institute for the Study of Natural and Artificial Intelligence at Harvard University. T.A.K., B.L., and R.H. are supported by Kempner Research Fellowships. M.J. is supported by the Kempner Graduate Fellowship.

References

Bansal, A., Schwarzschild, A., Borgnia, E., Emam, Z., Huang, F., Goldblum, M., and Goldstein, T. End-to-end algorithm synthesis with recurrent networks: Extrapolation without overthinking. *Advances in Neural Information Processing Systems*, 35:20232–20242, 2022.

Dehghani, M., Gouws, S., Vinyals, O., Uszkoreit, J., and Kaiser, Ł. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018.

Geiping, J., McLeish, S., Jain, N., Kirchenbauer, J., Singh, S., Bartoldson, B. R., Kailkhura, B., Bhatele, A., and Goldstein, T. Scaling up test-time compute with latent reasoning: A recurrent depth approach. *arXiv preprint arXiv: 2502.05171*, 2025.

Greff, K., Srivastava, R. K., and Schmidhuber, J. Highway and residual networks learn unrolled iterative estimation. *arXiv preprint arXiv:1612.07771*, 2016.

Huang, G., Sun, Y., Liu, Z., Sedra, D., and Weinberger, K. Q. Deep networks with stochastic depth. In *European conference on computer vision*, pp. 646–661. Springer, 2016.

Izmailov, P., Podoprikhin, D., Garipov, T., Vetrov, D., and Wilson, A. G. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*, 2018.

Jacobs, M., Fel, T., Hakim, R., Brondetta, A., Ba, D., and Keller, T. A. Block-recurrent dynamics in vision transformers. *arXiv preprint arXiv:2512.19941*, 2025.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Liu, B., Ash, J., Goel, S., Krishnamurthy, A., and Zhang, C. Exposing attention glitches with flip-flop language modeling. *Advances in Neural Information Processing Systems*, 36:25549–25583, 2023.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Manolache, A., Chamon, L., and Niepert, M. Learning (approximately) equivariant networks via constrained optimization. *Advances in Neural Information Processing Systems*, 38:144352–144381, 2026.

Oncescu, C.-A., Morwani, D., Jelassi, S., Meterez, A., Kwun, M., and Kakade, S. The recurrent transformer: Greater effective depth and efficient decoding. *arXiv preprint arXiv:2604.21215*, 2026.

Panigrahi, A., Saunshi, N., Lyu, K., Miryoosefi, S., Reddi, S., Kale, S., and Kumar, S. Efficient stagewise pretraining via progressive subnetworks. *arXiv preprint arXiv: 2402.05913*, 2024.

Pascanu, R., Mikolov, T., and Bengio, Y. On the difficulty of training recurrent neural networks. *International Conference on Machine Learning*, 2012.

Pertigkiozoglou, S., Chatzipantazis, E., Trivedi, S., and Daniilidis, K. Improving equivariant model training via constraint relaxation. *Advances in Neural Information Processing Systems*, 37:83497–83520, 2024.

Prairie, H., Novack, Z., Berg-Kirkpatrick, T., and Fu, D. Y. Parcae: Scaling laws for stable looped language models. *arXiv preprint arXiv:2604.12946*, 2026.

Saunshi, N., Karp, S., Krishnan, S., Miryoosefi, S., Reddi, S. J., and Kumar, S. On the inductive bias of stacking towards improving reasoning. *arXiv preprint arXiv: 2409.19044*, 2024.

Saunshi, N., Dikkala, N., Li, Z., Kumar, S., and Reddi, S. J. Reasoning with latent thoughts: On the power of looped transformers. *International Conference on Learning Representations*, 2025. doi: 10.48550/arXiv. 2502.17416.

Schwarzschild, A., Borgnia, E., Gupta, A., Huang, F., Vishkin, U., Goldblum, M., and Goldstein, T. Can you learn an algorithm? generalizing from easy to hard problems with recurrent networks. *Advances in Neural Information Processing Systems*, 34:6695–6706, 2021.

Zhang, H., Liu, B., Kim, J., and Risteski, A. Improving pathfinding with anchoring tokens. In *ICML 2025 Workshop on Methods and Opportunities at Small Scale*, 2025.

Zhu, R.-J., Wang, Z., Hua, K., Zhang, T., Li, Z., Que, H., Wei, B., Wen, Z., Yin, F., Xing, H., et al. Scaling latent reasoning via looped language models. *arXiv preprint arXiv:2510.25741*, 2025.

A. Proximal Gradient Interpretation

We show that the T-REX projection can be interpreted as a proximal step for a quadratic penalty measuring distance from the tied subspace. Let

$$\tilde{\theta}_1, \ldots, \tilde{\theta}_L$$

denote the layer parameters after a gradient step, and define their mean

$$\bar{\tilde{\theta}} = \frac{1}{L} \sum_{\ell=1}^{L} \tilde{\theta}_\ell.$$

Consider the penalty

$$g(\theta_1, \ldots, \theta_L) = \frac{\lambda}{2} \sum_{\ell=1}^{L} \|\theta_\ell - \bar{\theta}\|_2^2, \qquad \bar{\theta} = \frac{1}{L} \sum_{\ell=1}^{L} \theta_\ell.$$

This penalty is zero exactly on the tied subspace

$$\theta_1 = \theta_2 = \cdots = \theta_L,$$

and increases with the layerwise spread around the across-layer mean.

The proximal update associated with this penalty is the joint optimization problem

$$(\theta_1^{k+1}, \ldots, \theta_L^{k+1}) = \arg\min_{\theta_1, \ldots, \theta_L} \left\{ \frac{1}{2} \sum_{\ell=1}^{L} \|\theta_\ell - \tilde{\theta}_\ell\|_2^2 + \frac{\lambda}{2} \sum_{\ell=1}^{L} \|\theta_\ell - \bar{\theta}\|_2^2 \right\}.$$

We now derive its closed form.

First, we show that the proximal update preserves the across-layer mean. Differentiating the objective with respect to θ_ℓ gives

$$\theta_\ell - \tilde{\theta}_\ell + \lambda(\theta_\ell - \bar{\theta}) = 0,$$

where the derivative of the spread term is $\theta_\ell - \bar{\theta}$. Summing this first-order condition over ℓ yields

$$\sum_{\ell=1}^{L} (\theta_\ell - \tilde{\theta}_\ell) + \lambda \sum_{\ell=1}^{L} (\theta_\ell - \bar{\theta}) = 0.$$

The second term vanishes because

$$\sum_{\ell=1}^{L} (\theta_\ell - \bar{\theta}) = 0.$$

Therefore,

$$\sum_{\ell=1}^{L} \theta_\ell = \sum_{\ell=1}^{L} \tilde{\theta}_\ell,$$

and hence

$$\bar{\theta} = \bar{\tilde{\theta}}.$$

Substituting $\bar{\theta} = \bar{\tilde{\theta}}$ back into the first-order condition gives

$$\theta_\ell - \tilde{\theta}_\ell + \lambda(\theta_\ell - \bar{\tilde{\theta}}) = 0.$$

Rearranging,

$$(1 + \lambda)\theta_\ell = \tilde{\theta}_\ell + \lambda \bar{\tilde{\theta}},$$

so

$$\theta_\ell = \frac{1}{1 + \lambda} \tilde{\theta}_\ell + \frac{\lambda}{1 + \lambda} \bar{\tilde{\theta}}.$$

Defining

$$\alpha = \frac{\lambda}{1+\lambda},$$

we obtain

$$\theta_\ell = (1-\alpha)\tilde{\theta}_\ell + \alpha\bar{\tilde{\theta}},$$

which is exactly the T-REX projection.

Equivalently, writing the deviations from the mean as

$$\delta_\ell = \theta_\ell - \bar{\theta}, \qquad \tilde{\delta}_\ell = \tilde{\theta}_\ell - \bar{\tilde{\theta}},$$

the update becomes

$$\delta_\ell = \frac{1}{1+\lambda}\tilde{\delta}_\ell = (1-\alpha)\tilde{\delta}_\ell.$$

Thus, the proximal step preserves the across-layer mean while contracting all deviations from the tied subspace by a factor of $1-\alpha$. This matches the intended behavior of T-REX: it leaves the shared recurrent prototype unchanged while pulling each layer toward that prototype. Given this equivalence, we expect such methods to yield similar benefits to our proposed approach, and believe experimentation with proximal gradient methods to be an exciting direction for future research.

B. Convergence of T-REX

We first restate Proposition 3.1. Let $[L] = \{1, \dots, L\}$, and let $\theta_i^k \in \mathbb{R}^d$ denote the parameters of recurrent group i after step k. At each step, T-REX first applies an update direction u_i^k with learning rate $\eta_k \geq 0$,

$$\tilde{\theta}_i^{k+1} = \theta_i^k - \eta_k u_i^k,$$

and then projects the updated groups toward their mean,

$$\theta_i^{k+1} = (1-\alpha)\tilde{\theta}_i^{k+1} + \alpha\bar{\tilde{\theta}}^{\,k+1}, \qquad \bar{\tilde{\theta}}^{\,k+1} := \frac{1}{L}\sum_{\ell=1}^{L}\tilde{\theta}_\ell^{k+1}.$$

Proposition 3.1 (restated). Assume $0 < \alpha \leq 1$, $\eta_k \to 0$, and there exists $0 < G < \infty$ such that $\|u_i^k\|_2 \leq G$ for all $i \in [1, \dots, L]$ and all k. Define

$$\bar{\theta}^k := \frac{1}{L}\sum_{i=1}^{L}\theta_i^k.$$

Then

$$\max_{i\in[L]} \|\theta_i^k - \bar{\theta}^k\|_2 \to 0.$$

Equivalently, $\|\theta_i^k - \theta_j^k\|_2 \to 0$ for all $i, j \in [L]$.

Proof. Let

$$D^k := \max_{i,j\in[L]} \|\theta_i^k - \theta_j^k\|_2$$

be the largest pairwise distance between recurrent parameter groups. If $\alpha = 1$, then $\theta_i^{k+1} = \bar{\tilde{\theta}}^{\,k+1}$ for every i, so the claim holds for all $k \geq 1$. Now suppose $0 < \alpha < 1$ and set $c := 1-\alpha \in (0,1)$.

For any $i, j \in [L]$,

$$\begin{aligned}\|\theta_i^{k+1} - \theta_j^{k+1}\|_2 &= c\|\tilde{\theta}_i^{k+1} - \tilde{\theta}_j^{k+1}\|_2 \\ &= c\|\theta_i^k - \theta_j^k - \eta_k(u_i^k - u_j^k)\|_2 \\ &\leq cD^k + 2cG\eta_k.\end{aligned}$$

Taking the maximum over i, j gives

$$D^{k+1} \leq cD^k + 2cG\eta_k.$$

Next, pick an arbitrary $\varepsilon > 0$. Since $c > 0$, $G > 0$, and $1 - c > 0$, the number $(1-c)\varepsilon/(2cG)$ is positive. Because $\eta_k \to 0$, there exists K such that

$$\eta_k \leq \frac{(1-c)\varepsilon}{2cG} \qquad \text{for all } k \geq K,$$

or equivalently $2cG\eta_k \leq (1-c)\varepsilon$ for all $k \geq K$. Substituting this into the inequality above we get,

$$D^{k+1} \leq cD^k + (1-c)\varepsilon \qquad \text{where } k > K.$$

Iterating the previous inequality n times, for any $n \geq 1$,

$$D^{K+n} \leq c^n D^K + (1-c)\varepsilon \sum_{r=0}^{n-1} c^r$$

Since $0 < c < 1$, the finite geometric sum satisfies

$$(1-c)\sum_{r=0}^{n-1} c^r = 1 - c^n \leq 1.$$

Therefore,

$$D^{K+n} \leq c^n D^K + \varepsilon.$$

Taking $n \to \infty$ gives $\limsup_{k\to\infty} D^k \leq \varepsilon$. Since ε was arbitrary and $D^k \geq 0$, we conclude that $D^k \to 0$.

Finally, for each i,

$$\theta_i^k - \bar{\theta}^k = \theta_i^k - \frac{1}{L}\sum_{j=1}^{L}\theta_j^k = \frac{1}{L}\sum_{j=1}^{L}(\theta_i^k - \theta_j^k).$$

Hence, by the triangle inequality and homogeneity of the Euclidean norm,

$$\|\theta_i^k - \bar{\theta}^k\|_2 = \left\| \frac{1}{L}\sum_{j=1}^{L}(\theta_i^k - \theta_j^k) \right\|_2 \leq \frac{1}{L}\sum_{j=1}^{L}\|\theta_i^k - \theta_j^k\|_2 \leq D^k.$$

Thus $\max_i \|\theta_i^k - \bar{\theta}^k\|_2 \to 0$. Also, $\|\theta_i^k - \theta_j^k\|_2 \leq D^k \to 0$ for every pair i, j, completing the proof. □

C. Optimizer State Projection

Optimizer state projection. When using optimizers with momentum, T-REX can also project the optimizer state together with the weights. This prevents the optimizer's accumulated state from immediately counteracting the weight projection in the following update. For SGD with momentum, we project the layer velocities by

$$v_i \leftarrow (1-\alpha)v_i + \alpha\bar{v}, \qquad \bar{v} = \frac{1}{L}\sum_{i=1}^{L} v_i.$$

For Adam (Kingma & Ba, 2014), each layer has a first moment m_i and second moment s_i. We consider three variants:

1. **Weights only**: project $\theta_i \leftarrow (1-\alpha)\theta_i + \alpha\bar{\theta}$ and leave m_i, s_i unchanged.
2. **Weights + first moment**: additionally project

$$m_i \leftarrow (1-\alpha)m_i + \alpha\bar{m}, \qquad \bar{m} = \frac{1}{L}\sum_{i=1}^{L} m_i.$$

3. **Weights + both moments**: additionally project

$$s_i \leftarrow (1-\alpha)s_i + \alpha\bar{s}, \qquad \bar{s} = \frac{1}{L}\sum_{i=1}^{L} s_i.$$

Projecting the first moment aligns the accumulated gradient directions across layers and helps maintain the effect of the weight projection. Projecting the second moment also aligns the per-coordinate adaptive learning-rate scales; this may help once layers implement similar functions, but can be harmful while layers are still specializing.

In our maze-solving experiments (Appendix D.3), we compare these three variants and find that projecting the first moment provides the most benefit, while also projecting the second moment offers no additional gain and can sometimes hurt early in training.

D. Experimental Details

D.1. Details of the ReLU Toy Problem

The toy problem in Section 3.3 uses the two-parameter loss

$$\mathcal{L}(w_1, w_2) = \frac{1}{2}\sum_i \left(w_2\sigma(w_1 x_i + b) - y_i\right)^2, \qquad \sigma(z) = \max\{0, z\}, \qquad b = 10.$$

The tied objective is obtained by restricting to $w_1 = w_2 = w$:

$$\mathcal{L}_{\text{tied}}(w) = \mathcal{L}(w, w) = \frac{1}{2}\sum_i \left(w\sigma(w x_i + b) - y_i\right)^2.$$

We use two distinct examples, one of which is repeated many times. Let

$$A = 0.5, \qquad B = 3, \qquad n_{\text{barrier}} = 50.$$

The repeated barrier example is

$$x_{\text{barrier}} = -\frac{b}{A+B}, \qquad y_{\text{barrier}} = A\sigma(A x_{\text{barrier}} + b).$$

This choice ensures

$$A\sigma(A x_{\text{barrier}} + b) = B\sigma(B x_{\text{barrier}} + b),$$

so both $w = A$ and $w = B$ fit the repeated barrier example equally well. We then add one tilt example,

$$x_{\text{tilt}} = -\frac{b}{6}, \qquad y_{\text{tilt}} = B\sigma(B x_{\text{tilt}} + b),$$

which favors the larger tied solution $w = B = 3$. The full dataset is

$$\underbrace{(x_{\text{barrier}}, y_{\text{barrier}}), \ldots, (x_{\text{barrier}}, y_{\text{barrier}})}_{50 \text{ copies}} \quad \cup \quad \{(x_{\text{tilt}}, y_{\text{tilt}})\}.$$

This construction creates a barrier along the tied diagonal. Although both the small-weight region near $w = A$ and the larger solution near $w = B$ fit the repeated barrier example, the single tilt example makes $w = B$ preferable. However, moving from the small-weight basin toward B along the linear tied path $w_1 = w_2 = w$ requires temporarily increasing the loss. As a result, tied gradient descent initialized at $w_0 = 0.3$ is trapped by the local geometry and does not cross the barrier.

T-REX begins each outer iteration from a tied point (w, w), performs K unconstrained gradient steps on $\mathcal{L}(w_1, w_2)$, and then projects back to the tied subspace via

$$w \leftarrow \frac{w_1 + w_2}{2}.$$

Because the inner updates are not restricted to the diagonal, T-REX can move through off-diagonal regions where the loss decreases, thereby bypassing the high-loss segment of the tied linear path. In the experiments, we use learning rate $\eta = 4 \times 10^{-4}$, $K = 50$ inner untied steps, and 35 projection steps. The tied baseline is run for the same total number of gradient updates, 35×50 steps.

D.2. Soft Distance Propagation

Task. The soft distance propagation task operates on $H \times W$ grids ($H = W = 16$ at training time). Each sample has a seed map $s \in \{0,1\}^{H\times W}$ (1–3 randomly placed seed pixels) and a cost map $c \in [0,1]^{H\times W}$ of smoothed uniform noise. The distance field $h^{(t)}$ is initialized as

$$h_{i,j}^{(0)} = \begin{cases} 0 & \text{if } s_{i,j} = 1 \\ C_{\max} & \text{otherwise,} \end{cases}$$

with $C_{\max} = 10$, and propagated by the softmin recurrence

$$h_{i,j}^{(t+1)} = -\tau \log \sum_{z\in\mathcal{N}(i,j)} \exp(-z/\tau), \quad \mathcal{N}(i,j) = \{h_{i,j}^{(t)}\} \cup \{h_{i',j'}^{(t)} + c_{i,j} : (i',j') \text{ neighbor}\},$$

with temperature $\tau = 0.2$ and out-of-bounds neighbors treated as $+\infty$. This is a smooth approximation of Dijkstra-style shortest-path distance propagation.

Architecture. All models use a 3-layer transformer with $d = 128$, 4 attention heads, $4\times$ MLP expansion, 2D RoPE, no CLS token (1 token per pixel), and no final LayerNorm. The patch embedding is a $\mathrm{Conv2d}(2, d, \text{kernel} = 1)$ mapping each pixel's (seed, cost) pair to a d-dimensional token. The decoder head is a $\mathrm{Linear}(d, 1)$ projecting each token to a scalar distance value.

Models. We compare three main conditions: `standard` (3 independent layers) `weight_tied` (1 block repeated 3 times), and `T-REX` (3 independent layers with periodic projection). For all conditions, the training forward pass unrolls two consecutive passes through the 3-layer stack: the model first predicts $\hat{h}^{(3)}$ from the input, then feeds its hidden state (without re-embedding) through the 3 layers again to predict $\hat{h}^{(6)}$. Loss is $\mathrm{MSE}(\hat{h}^{(3)}, h^{(3)}) + \mathrm{MSE}(\hat{h}^{(6)}, h^{(6)})$.

Training. Optimizer: AdamW (Loshchilov & Hutter, 2017), $\mathrm{lr} = 10^{-3}$, cosine decay to 10^{-6}, 2000 warmup steps. Batch size: 512. Up to 200 epochs. Dataset: 50k train / 5k val / 5k test, generated on-the-fly. T-REX projection frequency: $N \in \{16, 64, 256\}$ (sweep).

Evaluation. We evaluate rollout performance across temporal horizons $T \in \{1, \ldots, 12\}$. For each value of T, we generate 1000 samples for validation and 1000 samples for OOD/test evaluation. This allows us to measure both in-distribution performance and temporal generalization as the number of steps increases.

Set	Grid	T	What it tests
Val	16×16	$1, \ldots, 6$	Model selection / in-distribution validation
Test ID	16×16	$1, \ldots, 6$	In-distribution generalization
Test OOD	16×16	$1, \ldots, 12$	Temporal generalization beyond training horizon

Unless otherwise stated, each reported evaluation point at a given horizon T is computed over 1000 independently generated samples. `T-REX` and `weight_tied` project to a single mean block and loop it T_{eval} times; standard cycles its 3 layers for T_{eval} steps.

D.3. Maze Solving

Task. Mazes are 21×21 pixel RGB images (white = open, black = wall, green = start, red = end). The label is a binary pixel mask marking the shortest solution path (1 = path, 0 = background). Loss is weighted cross-entropy with class weight $w_{\text{path}} = N_{\text{bg}}/N_{\text{path}}$ computed globally over the training set (path pixels are $\leq 11\%$ of the image).

Architecture. All models use $d = 128$, 4 heads, $4\times$ MLP, dynamic 2D RoPE, and an 8-block structure: 1 prefix block (applied once), 6 reasoning blocks (applied iteratively), and 1 suffix block (applied once). Each 21×21 image tokenizes to 441 tokens via $\mathrm{Conv2d}(3, d, \text{kernel} = 1)$. The segmentation head is $\mathrm{Linear}(d, 2)$ applied to every token after RMSNorm.

Models. We compare `standard` (6 independent reasoning blocks), `weight_tied` (1 shared reasoning block looped $6\times$), and `T-REX` (6 independent reasoning blocks with periodic projection; prefix and suffix are never projected). We additionally ablate momentum variants, finding that projecting the first moment produces the best results (re: Figure 3).

Training. Optimizer: AdamW, lr $= 10^{-3}$, weight_decay$= 0.05$, cosine decay to 10^{-6}. Batch size: 256. 200 epochs. Stochastic iterations: at each training step the number of reasoning block applications is sampled uniformly from $\{6, 7, 8, 9, 10\}$, ensuring all reasoning blocks receive gradient every step and exposing iterable models to deeper compute budgets. Loss is computed only on the final output (no intermediate supervision).

Data. All mazes are 21×21 pixels (11×11 cells, 441 tokens). We generate 130k samples at three difficulty levels defined by solution path length:

Split	Path length (px)	Count	Role
Train	1–50	100k	Training
Val	1–50	10k	Validation
OOD	51–100	10k	OOD

Data. All mazes are 21×21 pixels (11×11 cells, 441 tokens). Maze generation code is adapted from Schwarzschild et al. (2021) and Bansal et al. (2022)[1]. We generate 130k samples at three difficulty levels defined by solution path length:

Evaluation. At evaluation, T-REX and `weight_tied` loop their single mean block for N reasoning applications; `standard` cycles its 6 reasoning blocks for N steps. We report results at $N = 6$ (standard) and $N = 12$ (extra thinking steps). T-REX weights are projected to their mean before evaluation. Each evaluation set contains 10000 mazes; metrics are averaged over 3 training seeds per condition.

D.4. In-Context Graph Traversal

Task. We construct a two-level Erdős–Rényi graph $G = (V, E)$ with $N_1 = 12$ clusters of $N_2 = 8$ nodes each (96 total nodes, IDs 0–95). Intra-cluster edges are drawn with $p_2 = 0.6$; inter-cluster edges are drawn at the cluster level with $p_1 = 0.2$ (one random node-level edge per connected cluster pair). Source-target pairs are always inter-cluster.

The full input sequence is:

$$[\text{edge}_0^u][\text{edge}_0^v] \cdots [\text{SEP}][S][T][\text{SEP}][S][v_1] \cdots [T][\text{EOS}][\text{PAD} \cdots]$$

where the edge list and query form a bidirectional prefix, and the path is predicted causally. Vocabulary size is 99 (96 node IDs + SEP, EOS, PAD); maximum sequence length 512. Loss is cross-entropy on path + EOS positions only. Difficulty is measured by cluster-level BFS distance $d = \text{dist}_{\text{cluster}}(C(S), C(T))$.

Architecture. $d = 128$, 4 heads, 4× MLP, 1D RoPE. All models use prefix-LM attention: prefix positions attend to all prefix positions (bidirectional), and path positions attend to all prefix positions plus prior path positions (causal). The model has $K = 10$ reasoning blocks flanked by a prefix block and suffix block (12 blocks total).

Models. We compare `standard` (10 independent reasoning blocks, no regularization) against T-REX with soft EMA pull ($N = 1, \alpha \in \{10^{-3}, 10^{-2}, 10^{-1}\}$, annealed from 0 over the first 20 epochs). Stochastic depth (Huang et al., 2016)(drop_path $\in \{0, 0.1, 0.4, 0.6\}$) and AdamW $\beta_2 \in \{0.95, 0.999\}$ are swept for all conditions. We train 5 seeds per experimental setting (160 total SLURM runs).

Training. Optimizer: AdamW, lr $= 10^{-3}$, weight_decay$= 0.05$. 100 epochs, batch size 256, cosine LR decay to 10^{-6} with 10-epoch warmup. Fixed $K = 10$ iterations per forward pass (no stochastic iterations).

Data. Each sample has a freshly generated graph topology and source-target pair.

Split	Cluster dist	Count	Role
Train	1–2	100k	Training
Val	1–2	1k	Validation
OOD	3–4	1k	OOD evaluation

[1] https://github.com/aks2203/easy-to-hard and https://github.com/aks2203/easy-to-hard-data

Evaluation. We report autoregressive (AR) path accuracy: any valid shortest $S \to T$ path is accepted (not just the specific ground-truth path). T-REX weights are projected to their mean before AR decoding. The primary evaluation plot shows val accuracy vs. OOD-Inter accuracy across all (condition, hyperparameter) combinations, with the Pareto frontier highlighted.

E. Data Examples

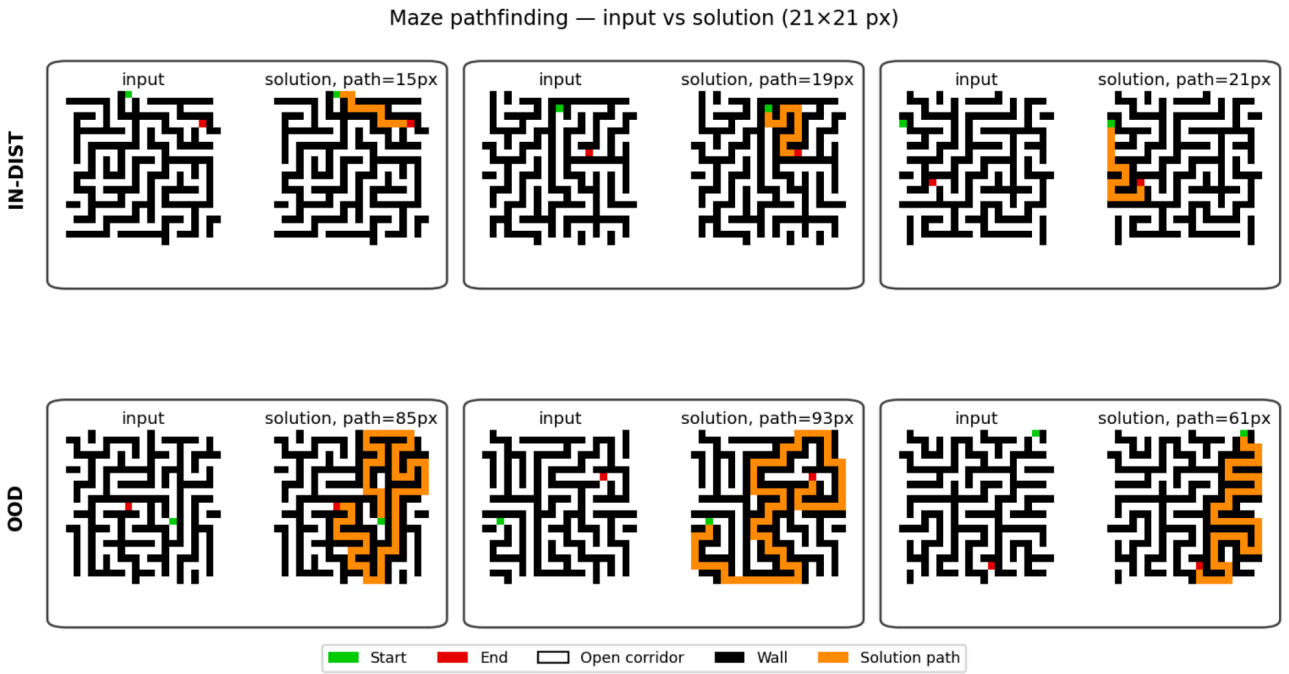

Figure 5. **Maze data examples.** Representative in-distribution and out-of-distribution mazes, together with their corresponding solution paths.

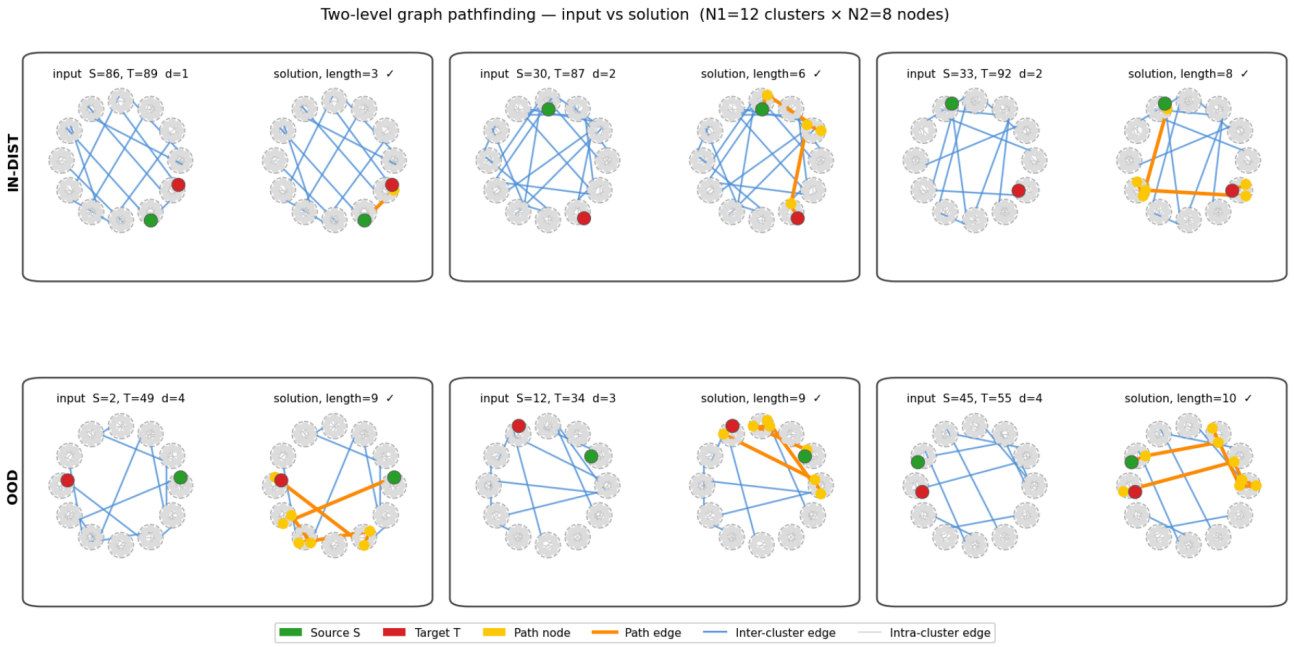

Figure 6. **Graph data examples.** Representative in-distribution and out-of-distribution graphs, together with their corresponding solutions.

F. Extended Experimental Results

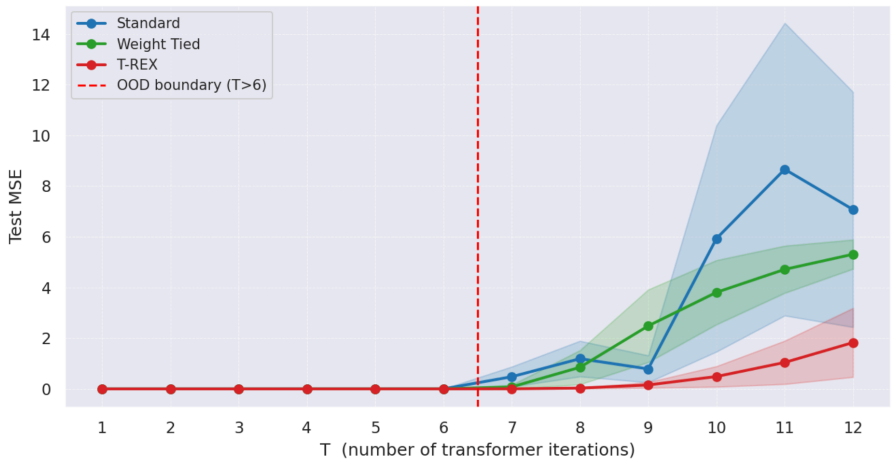

Figure 7. **Soft distance propagation loss across timesteps.** All models achieve low in-distribution loss, while T-REX attains the lowest out-of-distribution loss. T-REX also exhibits the lowest variance across timesteps, whereas the weight-tied and standard models show substantially higher variance.

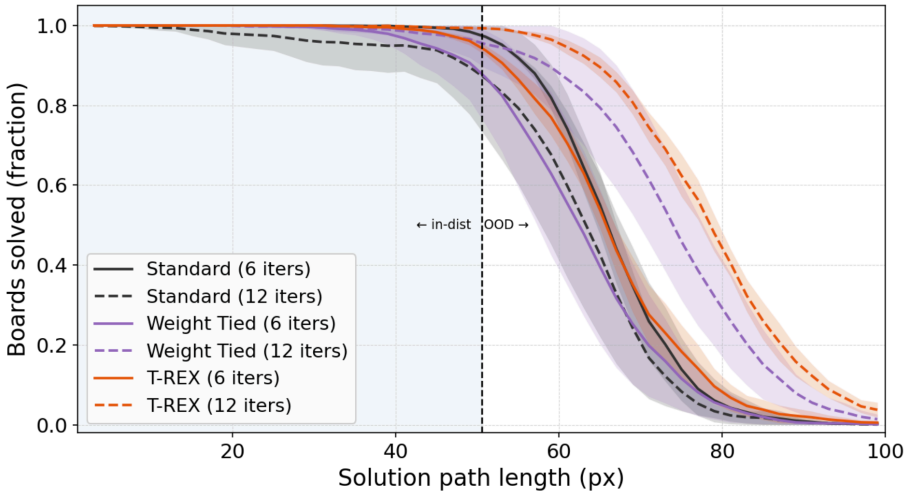

Figure 8. **Extended maze-solving results across computation budgets.** We evaluate T-REX, a standard transformer, and a looped transformer on maze instances with increasingly long solution paths. Results are shown for both 6 and 12 inference iterations, illustrating how each model uses additional computation at test time. T-REX maintains substantially stronger performance on longer-horizon mazes and benefits more reliably from additional thinking steps, whereas the standard and looped transformer baselines degrade more rapidly out of distribution.

Table 1. Standard condition (8 settings). AR-path accuracy (mean ± std over 5 seeds) for every hyperparameter configuration shown in Figure 4. IN-DIST = validation split; OOD = same graph density, longer hop count. Settings sorted by descending OOD accuracy.

Setting	IN-DIST	OOD	n
$p_{\text{drop}} = 0.1$, $\beta_2 = 0.999$	0.983 ± 0.003	0.167 ± 0.052	5
$p_{\text{drop}} = 0.6$, $\beta_2 = 0.999$	0.984 ± 0.002	0.153 ± 0.026	5
$p_{\text{drop}} = 0.1$, $\beta_2 = 0.95$	0.984 ± 0.002	0.148 ± 0.107	5
$p_{\text{drop}} = 0.4$, $\beta_2 = 0.95$	0.983 ± 0.004	0.139 ± 0.071	5
$p_{\text{drop}} = 0.4$, $\beta_2 = 0.999$	0.987 ± 0.001	0.114 ± 0.049	5
$p_{\text{drop}} = 0.6$, $\beta_2 = 0.95$	0.986 ± 0.002	0.109 ± 0.032	5
$p_{\text{drop}} = 0.0$, $\beta_2 = 0.999$	0.965 ± 0.010	0.060 ± 0.042	5
$p_{\text{drop}} = 0.0$, $\beta_2 = 0.95$	0.959 ± 0.008	0.050 ± 0.027	5

Table 2. T-REX condition with hard mean projection at eval (12 settings). AR-path accuracy (mean ± std over 5 seeds) for every hyperparameter configuration shown in Figure 4. IN-DIST = validation split; OOD = same graph density, longer hop count. Settings sorted by descending OOD accuracy.

Setting	IN-DIST	OOD	n
$\alpha = 0.1$, $p_{\text{drop}} = 0.1$	0.969 ± 0.018	0.438 ± 0.087	5
$\alpha = 0.1$, $p_{\text{drop}} = 0.4$	0.857 ± 0.236	0.382 ± 0.182	5
$\alpha = 0.01$, $p_{\text{drop}} = 0.1$	0.955 ± 0.020	0.359 ± 0.196	5
$\alpha = 0.1$, $p_{\text{drop}} = 0.0$	0.908 ± 0.094	0.342 ± 0.069	5
$\alpha = 0.001$, $p_{\text{drop}} = 0.4$	0.960 ± 0.013	0.324 ± 0.073	5
$\alpha = 0.01$, $p_{\text{drop}} = 0.6$	0.921 ± 0.060	0.293 ± 0.095	5
$\alpha = 0.01$, $p_{\text{drop}} = 0.0$	0.865 ± 0.183	0.286 ± 0.182	5
$\alpha = 0.01$, $p_{\text{drop}} = 0.4$	0.848 ± 0.256	0.267 ± 0.181	5
$\alpha = 0.001$, $p_{\text{drop}} = 0.6$	0.907 ± 0.074	0.245 ± 0.091	5
$\alpha = 0.1$, $p_{\text{drop}} = 0.6$	0.906 ± 0.127	0.205 ± 0.072	5
$\alpha = 0.001$, $p_{\text{drop}} = 0.1$	0.714 ± 0.287	0.165 ± 0.155	5
$\alpha = 0.001$, $p_{\text{drop}} = 0.0$	0.033 ± 0.066	0.000 ± 0.001	5

Table 3. Weight-tied with fixed reasoning depth at training time ($T_{\text{train}} = 10$); 20 settings. AR-path accuracy (mean $\pm$ std over 5 seeds) for every hyperparameter configuration shown in Figure 4. IN-DIST = validation split; OOD = same graph density, longer hop count. Settings sorted by descending OOD accuracy.

Setting	IN-DIST	OOD	n
$\beta_2 = 0.999, T = 12$	0.964 ± 0.024	0.194 ± 0.062	5
$\beta_2 = 0.999, T = 11$	0.977 ± 0.011	0.193 ± 0.055	5
$\beta_2 = 0.999, T = 10$	0.977 ± 0.010	0.173 ± 0.049	5
$\beta_2 = 0.999, T = 13$	0.883 ± 0.116	0.149 ± 0.068	5
$\beta_2 = 0.95, T = 11$	0.979 ± 0.009	0.141 ± 0.027	5
$\beta_2 = 0.95, T = 12$	0.973 ± 0.014	0.131 ± 0.028	5
$\beta_2 = 0.95, T = 10$	0.981 ± 0.008	0.129 ± 0.015	5
$\beta_2 = 0.999, T = 9$	0.968 ± 0.009	0.120 ± 0.051	5
$\beta_2 = 0.95, T = 13$	0.944 ± 0.033	0.104 ± 0.028	5
$\beta_2 = 0.999, T = 14$	0.707 ± 0.231	0.095 ± 0.077	5
$\beta_2 = 0.95, T = 9$	0.967 ± 0.020	0.081 ± 0.018	5
$\beta_2 = 0.95, T = 14$	0.833 ± 0.111	0.071 ± 0.033	5
$\beta_2 = 0.999, T = 15$	0.500 ± 0.258	0.059 ± 0.061	5
$\beta_2 = 0.999, T = 8$	0.880 ± 0.085	0.056 ± 0.037	5
$\beta_2 = 0.95, T = 15$	0.647 ± 0.206	0.045 ± 0.026	5
$\beta_2 = 0.95, T = 8$	0.808 ± 0.091	0.022 ± 0.013	5
$\beta_2 = 0.999, T = 7$	0.601 ± 0.201	0.014 ± 0.018	5
$\beta_2 = 0.999, T = 6$	0.198 ± 0.201	0.002 ± 0.004	5
$\beta_2 = 0.95, T = 7$	0.413 ± 0.167	0.002 ± 0.001	5
$\beta_2 = 0.95, T = 6$	0.111 ± 0.111	0.000 ± 0.000	5

Table 4. Weight-tied with stochastic reasoning depth $T_{\text{train}} \sim \text{Poisson}(10)$ at training time; 20 settings. AR-path accuracy (mean $\pm$ std over 5 seeds) for every hyperparameter configuration shown in Figure 4. IN-DIST = validation split; OOD = same graph density, longer hop count. Settings sorted by descending OOD accuracy.

Setting	IN-DIST	OOD	n
$\beta_2 = 0.95, T = 12$	0.975 ± 0.020	0.251 ± 0.173	5
$\beta_2 = 0.95, T = 13$	0.976 ± 0.020	0.249 ± 0.171	5
$\beta_2 = 0.95, T = 11$	0.976 ± 0.019	0.248 ± 0.168	5
$\beta_2 = 0.95, T = 14$	0.976 ± 0.019	0.246 ± 0.172	5
$\beta_2 = 0.95, T = 15$	0.976 ± 0.019	0.243 ± 0.174	5
$\beta_2 = 0.95, T = 10$	0.976 ± 0.019	0.242 ± 0.159	5
$\beta_2 = 0.95, T = 9$	0.975 ± 0.020	0.236 ± 0.149	5
$\beta_2 = 0.95, T = 8$	0.974 ± 0.019	0.215 ± 0.128	5
$\beta_2 = 0.95, T = 7$	0.971 ± 0.019	0.187 ± 0.096	5
$\beta_2 = 0.999, T = 12$	0.986 ± 0.002	0.174 ± 0.078	5
$\beta_2 = 0.999, T = 13$	0.986 ± 0.003	0.173 ± 0.081	5
$\beta_2 = 0.999, T = 11$	0.986 ± 0.002	0.173 ± 0.076	5
$\beta_2 = 0.999, T = 14$	0.986 ± 0.003	0.172 ± 0.079	5
$\beta_2 = 0.999, T = 15$	0.986 ± 0.003	0.168 ± 0.080	5
$\beta_2 = 0.999, T = 10$	0.986 ± 0.002	0.168 ± 0.074	5
$\beta_2 = 0.999, T = 9$	0.986 ± 0.002	0.159 ± 0.073	5
$\beta_2 = 0.999, T = 8$	0.985 ± 0.003	0.141 ± 0.067	5
$\beta_2 = 0.95, T = 6$	0.964 ± 0.021	0.122 ± 0.033	5
$\beta_2 = 0.999, T = 7$	0.984 ± 0.002	0.112 ± 0.056	5
$\beta_2 = 0.999, T = 6$	0.981 ± 0.002	0.081 ± 0.041	5