Adaptive Recurrence as Algorithmic Time for Length Generalization in Addition

Imran Ibrahimli [1 *] **Stefan Wermter** [2] **Jae Hee Lee** [2]

Abstract

Length generalization asks whether a model can reuse a learned computation beyond the lengths seen in training. Arithmetic makes this test precise because a model trained on short additions should apply the same digit-wise rule for more steps, rather than fit patterns tied to the training range. Many strong extrapolation results, however, rely on explicit aids that tell the model how digits should be aligned. We ask whether a model can length-generalize without such aids. We train small looped transformers with adaptive halting on a controlled decimal-addition task, without positional encodings or arithmetic-specific side information. The models extrapolate beyond the training range across multiple training regimes. To understand why, we analyze their internal dynamics and find that recurrent depth becomes organized as algorithmic time: later answer positions are resolved at later recurrent phases. These results suggest that adaptive recurrence can turn repeated latent computation into a less hand-engineered route for applying a digit-wise rule beyond the training length.

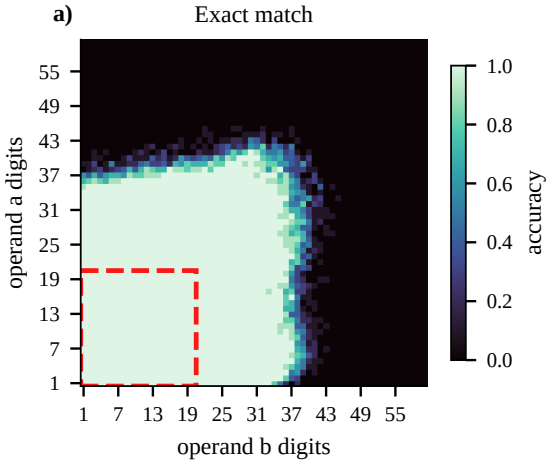

Figure 1. **Mid-regime operand-length generalization.** The red guide marks the training boundary. Panel (a) shows exact-match accuracy for each operand-length pair. The model is trained on up to 20-digit operands and extrapolates to substantially longer additions; digit-accuracy, halt-step, and answer-length diagnostics are in Appendix E.

1. Introduction

Length generalization is a concrete test of whether a neural model has learned a reusable computation rather than a surface pattern tied to the training range. It is closely related to compositional generalization and productivity: the ability to extend a learned rule to larger structures than those observed during training (Lake & Baroni, 2018; Hupkes et al., 2020). Arithmetic is a useful setting for this question because the target algorithm is known, correctness is unambiguous, and intermediate variables such as digit position and carry have clear meanings. A model trained on short additions should not merely fit the observed input lengths; it should apply the same digit-wise rule for more steps.

This remains difficult for transformers. Small transformers can learn arithmetic in distribution, yet fail sharply when evaluated on longer inputs (Lee et al., 2023; Jelassi et al., 2023; Zhou et al., 2024). Prior work also shows that positional choices themselves matter for extrapolation, including methods that bias, randomize, or remove positional encodings (Press et al., 2021; Ruoss et al., 2023; Kazemnejad et al., 2023). In arithmetic, many of the strongest results add further task structure, such as digit indices, same-significance position IDs, task hints, or custom encodings of arithmetic structure (McLeish et al., 2024; Cho et al., 2024; Awasthi & Gupta, 2023; Sabbaghi et al., 2024; Cho et al., 2025).

In this paper, we ask whether a model can length-generalize *without* explicit aids that tell the model how the digits should be aligned. Prior work shows that recurrence and depth-wise weight sharing can improve compositional and length generalization (Dehghani et al., 2019; Csordas et al., 2021; Ontañón et al., 2022; Fan et al., 2025; Soulos et al., 2024). Even for addition, however, existing looped-transformer

[*]Work done while at the University of Hamburg. [1]Independent researcher, Hamburg, Germany [2]University of Hamburg, Hamburg, Germany. Correspondence to: Imran Ibrahimli <imranibrahimli98@gmail.com>, Jae Hee Lee <jae.hee.lee@uni-hamburg.de>.

Proceedings of the 43^{rd} International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026.

results rely on structured training or decoding protocols, so they do not settle whether looped transformers address the length-generalization problem in an autoregressive addition setting without explicit arithmetic coordinates.

To answer the question above, we train small looped transformers with tokenwise halting on a controlled decimal-addition task (Section 4). Across three training regimes, the models retain greater than 99% mean exact-match accuracy up to $1.8\times$ the maximum trained operand length. A fixed-loop ablation solves the training-length distribution but fails under length extrapolation, showing that adaptive halting is not just an implementation detail in this setup.

We then ask why the halting model extrapolates (Section 5). We find that recurrent depth becomes organized as algorithmic time, with a computation frontier moving across digit positions as recurrent time increases. Halt depth grows with answer length, and independent observables align along the same diagonal frontier over recurrent iteration and answer digit position. Taken together, the behavioral and mechanistic evidence support the same contribution: adaptive recurrence can turn extra depth into *algorithmic time*, giving an inspectable route by which a model applies a digit-wise rule beyond the training length without explicit digit-alignment aids.

2. Related Work and Background

Length generalization is a concrete form of compositional generalization: a model must apply the same rule for more steps than it saw in training (Lake & Baroni, 2018; Hupkes et al., 2020). In arithmetic, small transformers often learn the training distribution but fail on longer inputs (Lee et al., 2023; Jelassi et al., 2023; Zhou et al., 2024). Many stronger arithmetic systems improve extrapolation by supplying task structure, such as digit-position embeddings, same-significance position IDs, task hints, or symmetry-based encodings (McLeish et al., 2024; Cho et al., 2024; Awasthi & Gupta, 2023; Sabbaghi et al., 2024; Cho et al., 2025). NoPE is relevant context because it removes positional embeddings rather than adding arithmetic coordinates; our setting asks whether recurrence and learned halting can make such a coordinate emerge internally.

Our architectural motivation comes from adaptive computation and recurrent-depth transformers (Graves, 2016; Dehghani et al., 2019; Banino et al., 2021; Fan et al., 2025; Soulos et al., 2024). PonderNet is especially useful for our setting because it was proposed as a probabilistic successor to ACT: it replaces ACT's heuristic computation-cost objective and thresholded accumulated halting with a distribution over stopping times, an expected prediction loss, and a KL prior over halt depths (Banino et al., 2021). Prior work shows that recurrence, weight sharing, and other design choices can affect systematic or length generalization (Csordas et al., 2021; Ontañón et al., 2022), but our focus is mechanistic: whether the learned recurrent dynamics form a readable computation schedule over digit positions. This connects to mechanistic analyses of arithmetic and recurrent models (Quirke & Barez, 2024; Lu et al., 2025; Blayney et al., 2026), while using a controlled setting where intermediate variables can be probed directly. Appendices B and C provide a more comprehensive related work discussion and baseline-context table.

Adaptive recurrent computation. Adaptive Computation Time introduced learned halting for recurrent models: instead of applying a fixed number of transition steps, a model can learn how long to keep refining a state before producing an output (Graves, 2016). The Universal Transformer brought this idea to transformer architectures by repeatedly applying a shared transition block across recurrent depth, with optional adaptive halting at each position (Dehghani et al., 2019). Modern looped transformers use the same weight-sharing principle for transformer blocks, turning depth into a recurrent computation axis (Fan et al., 2025; Soulos et al., 2024).

Let $x_i^t \in \mathbb{R}^d$ be the residual state at token position i after loop iteration t. A looped transformer applies the same block F_θ repeatedly,

$$x^{t+1} = F_\theta(x^t), \qquad t = 0, \ldots, T-1. \tag{1}$$

Because F_θ is shared, a useful update can in principle be reused on inputs that require more recurrent steps. For addition, this is the relevant inductive bias: a digit-wise update learned on short examples might be applied for more positions by running the loop longer.

PonderNet halting. PonderNet turns learned halting into a distribution over stopping times (Banino et al., 2021). It was introduced to address two practical issues with ACT: sensitivity to the trade-off between accuracy and compute, and a biased computation-cost gradient that only flows through the final ponder step. PonderNet instead makes the halt depth probabilistic and trains on the expected loss over possible stopping times. Index loop steps by $n = 1, \ldots, T$. At token i and step n, the halting head predicts a conditional halt probability

$$\lambda_i^n = \sigma(w_h^\top x_i^n + b_h). \tag{2}$$

This induces a distribution $p_i(n)$ over stopping steps, so the model learns both what prediction to make at each recurrent step and how much computation to spend before using that prediction. We use PonderNet as a generic differentiable halting mechanism; we do not assume that its geometric prior is an ideal model of computation time for addition. The finite-cap implementation used in our experiments is given in Appendix A.

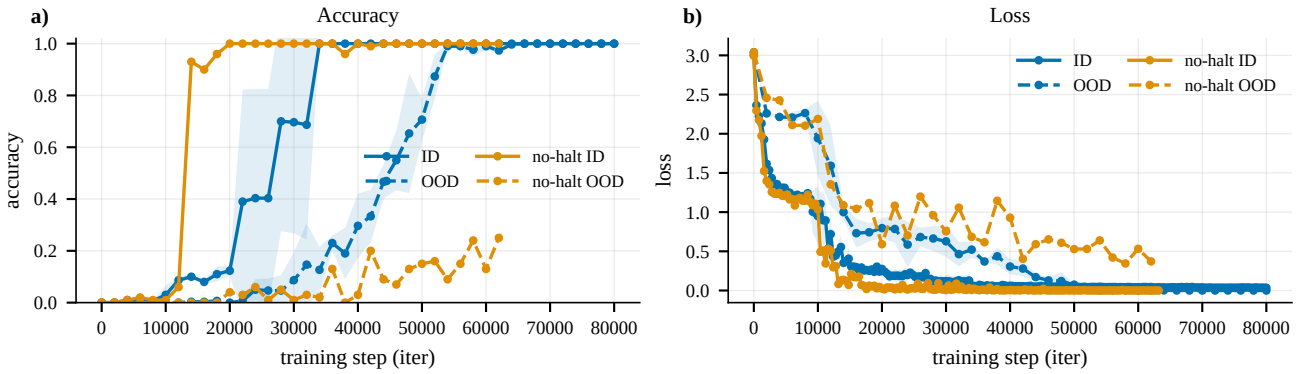

Figure 2. **Mid-regime training curves and no-halting ablation.** Panels show accuracy and loss for PonderNet mid-regime runs and the fixed-loop mid ablation. The fixed-loop ablation uses 40 recurrent iterations, matching the mid-regime checkpoint loop cap. The legend entries are ID and OOD for the PonderNet runs, and no-halt ID and no-halt OOD for the ablation; ID curves are solid and OOD curves are dashed. The PonderNet curves show the mean over three seeds with one-standard-deviation bands, while the ablation is a single run. The OOD stream samples the ten held-out operand lengths immediately above training, 21–30 digits for this regime. The PonderNet runs acquire high OOD accuracy after fitting the training lengths, whereas the no-halting ablation fits ID examples but generalizes much less reliably.

3. Experimental Setup

Task. Examples are character-level addition strings. We use `$` as both the start delimiter and the answer-end delimiter, and `;` as the example separator. Both operands and the answer are written least-significant digit first, so the ordinary equation $23 + 18 = 41$ is represented as `$32+81=14$;`. This format is intentionally task-aligned: left-to-right generation in the token sequence follows the ordinary carry direction from least to most significant digit, so the task is cleaner than standard most-significant-digit-first addition. The model is trained autoregressively, with loss applied only on the answer span after `=`, including the closing `$`; prompt tokens and the example separator are masked out. We use k for the answer digit position, with $k = 0$ denoting the least-significant answer digit. When discussing the internal dynamics, t denotes the recurrent loop iteration.

Model and objective. All PonderNet experiments use one-layer adaptive looped transformers with embedding dimension 128, 2 attention heads, dropout 0.1, and tokenwise halting. Let ℓ_i^n be the cross-entropy loss for supervised answer-span position i when read out after step n. The training objective over supervised answer-span positions $\mathcal{A}$ is

$$\mathcal{L} = \frac{1}{|\mathcal{A}|} \sum_{i \in \mathcal{A}} \left[\sum_{n=1}^{T} p_i(n) \ell_i^n + \beta \, \mathrm{KL}\big(p_i \,\|\, q_{\lambda_p}\big) \right], \qquad (3)$$

where p_i is the learned finite-cap halt distribution and q_{λ_p} is the corresponding geometric prior distribution, obtained by replacing the learned conditional halt probabilities with the prior parameter λ_p; see Appendix A. At evaluation time we decode with a thresholded halt rule on cumulative halt probability, using threshold 0.2. The training objective does not depend on this inference threshold. The model uses *no positional encodings*. Token embeddings are injected only before the first recurrent step, not re-injected at every loop iteration. We train short, mid, and long regimes with three seeds each. The fixed-loop baseline uses the same mid-regime data and 40-step loop cap but removes the halting head and PonderNet objective. Appendix H gives the full configuration summary.

4. Length Generalization Without Explicit Coordinates

Evaluation protocol. For an operand-length grid with maximum length L, we evaluate every pair $(\ell_a, \ell_b) \in \{1, \ldots, L\}^2$. Exact match requires every generated digit, over the ground-truth answer length, to equal the target digit; delimiter prediction is not included in these behavioral metrics. Digit accuracy pools correctness over answer positions. For a sample, the reported selected halt step is the maximum selected step over generated answer digits. For Table 2, we report the largest number of digits covered with exact-match accuracy above 99%. The table's Mean EM column reports mean exact-match accuracy on that boundary ring. The training curves show the ID and OOD evaluations, OOD covering the ten held-out operand lengths immediately above the training range.

Main behavioral result. Figure 1 shows the main exact-match evaluation for the mid regime, trained on operands of length 1–20. The model remains accurate well beyond the training square, then fails gradually. The extrapolation is not a consequence of an explicit coordinate system: the model receives no absolute or relative positional encodings, no digit-index hints, no same-significance position IDs, no

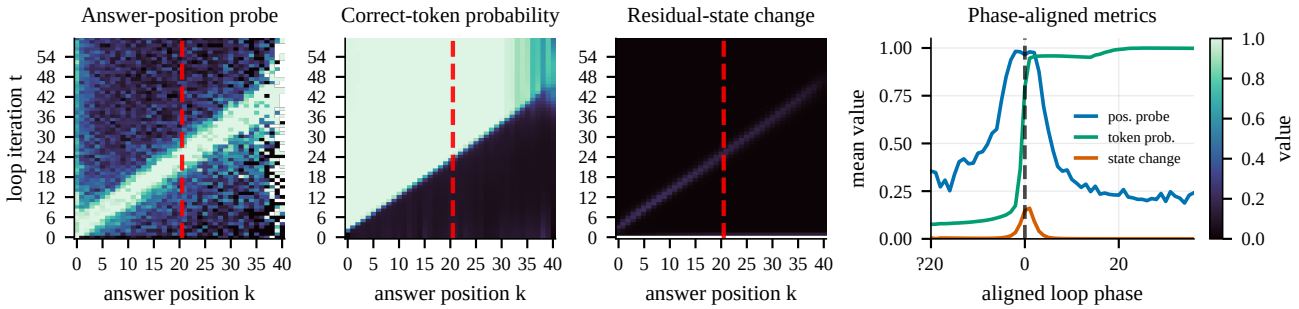

Figure 3. **Main mechanistic evidence for algorithmic time.** Mid-regime diagnostics at answer-token predictor states, red dashed lines mark the maximum trained length. From left to right: stepwise answer-position probe accuracy, correct-token probability under the model's own readout, residual-state change between consecutive loop iterations, and phase-aligned metrics after shifting each digit by its correct-token frontier time. The four views show the same moving computation frontier; halting rises after the answer digit becomes readable.

carry labels, and no scratchpad supervision. The plots in Figures 4 and 5 show that the selected halt step grows with answer length, rather than remaining fixed after training lengths. This is the first behavioral sign that the model is not merely applying a shallow in-distribution heuristic. It has learned to spend more recurrent computation on later answer positions, and that schedule continues beyond the trained operand lengths.

Figure 2 shows that this OOD behavior appears during training across seeds in the mid regime. The no-halting ablation reaches high ID accuracy, but its OOD curve stays far below the PonderNet runs. This suggests that, in this setup, the learned halting distribution is not just a reporting device for compute usage; it is part of the mechanism that lets recurrence scale to held-out lengths.

The across-regime summary and behavioral length-generalization table are in Appendix D.

5. Recurrent Depth Becomes Algorithmic Time

The behavioral result raises a mechanistic question: what is the recurrence doing? For answer digit k, a serial addition rule must use the aligned operand digits and carry-in c_k, produce the answer digit, and pass carry information to later positions. If recurrent depth is acting as algorithmic time, these subcomputations should not become available for all answer positions at once. They should sweep across k as loop iteration t increases. Because the answer is reversed, this sweep is ordinary right-to-left addition in numerical notation, but left-to-right in the generated token sequence.

We call this moving boundary the *computation frontier*. The main diagnostic is whether independent observables agree on the same diagonal frontier in (t, k) space. For this analysis, we collect residual states at the predictor position immediately before each answer digit and train stepwise linear probes for answer position and carry-in; additional probe details and the frontier-fitting definition are in Appendix G.

Figure 3 shows the resulting frontier. First, answer-position information is phase-specific: it is not linearly decodable at all recurrent steps, but becomes decodable near the active phase for that digit. Second, the same diagonal appears through the model's own language-model head: before the frontier, the correct answer digit is usually not readable, and after it the correct-token probability rises sharply. Third, residual-state change concentrates in the same narrow band, showing that the recurrent update is doing most of its work where the digit is becoming readable, rather than uniformly refining all positions. Fourth, after aligning digits by their correct-token frontier time, position decodability, correct-token probability, state change, and halting align on a common phase profile.

The model first moves the residual state for a digit into a phase where its position and answer become readable, then the halting probability rises after the prediction has become available. Thus halting behaves like a learned stopping rule attached to an ordered recurrent computation over answer positions. This is the sense in which recurrence becomes algorithmic time: loop iteration supplies an internal axis along which the digit-wise addition rule is computed.

6. Discussion and Limitations

The result should be read as a mechanistic and architectural claim, not as a new state-of-the-art arithmetic system. Reversed least-significant-digit-first notation is a task-aligned simplification, an adaptive looped transformer is overkill for addition, and we test one-layer PonderNet models on two-operand decimal addition only. Standard notation, multiplication, multi-operand arithmetic, and non-arithmetic algorithmic tasks remain open.

The evidence for the computation frontier is convergent but

observational: probes, readout, state change, phase alignment, and halting agree, but causal interventions are needed to show that the frontier is necessary, and more fixed-loop seeds would strengthen the negative-control evidence. The failure boundary should also not be read as a hard $2\times$ limit. Halt depth grows roughly linearly, but finite loop caps, the KL prior toward shorter computation, and frontier or readout drift can make extrapolation fail before available recurrent depth is exhausted.

Adaptive recurrence gives a transformer a reusable transition and a learned stopping rule. On least-significant-digit-first addition, this is enough to obtain moderate but clear length generalization without explicit positional encodings or arithmetic-specific side information. The internal dynamics explain why: recurrent depth becomes a learned time axis, and answer digits are computed along a diagonal frontier over loop iteration and digit position. Looped transformers can therefore sometimes internalize the coordinate structure that other length-generalization methods provide explicitly. This study focuses on reversed addition, leaving other arithmetic formats, broader algorithmic tasks, deeper causal circuit analysis, and alternative halting mechanisms as natural next steps.

References

Awasthi, P. and Gupta, A. Improving Length-Generalization in Transformers via Task Hinting. *arXiv preprint arXiv:2310.00726*, 2023.

Banino, A., Balaguer, J., and Blundell, C. Pondernet: Learning to ponder, 2021. URL https://arxiv.org/abs/2107.05407. 8th ICML Workshop on Automated Machine Learning.

Belrose, N., Ostrovsky, I., McKinney, L., Furman, Z., Smith, L., Halawi, D., Biderman, S., and Steinhardt, J. Eliciting latent predictions from transformers with the tuned lens, 2023. URL https://arxiv.org/abs/2303.08112.

Blayney, H., Arroyo, A., Obando-Ceron, J., Castro, P. S., Courville, A., Bronstein, M. M., and Dong, X. A mechanistic analysis of looped reasoning language models, 2026. URL https://arxiv.org/abs/2604.11791.

Cho, H., Cha, J., Awasthi, P., Bhojanapalli, S., Gupta, A., and Yun, C. Position Coupling: Improving Length Generalization of Arithmetic Transformers Using Task Structure. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Cho, H., Cha, J., Bhojanapalli, S., and Yun, C. Arithmetic Transformers Can Length-Generalize in Both Operand Length and Count. In *The Thirteenth International Conference on Learning Representations*, 2025.

Csordas, R., Irie, K., and Schmidhuber, J. The devil is in the detail: Simple tricks improve systematic generalization of transformers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 619–634. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.emnlp-main.49. URL https://aclanthology.org/2021.emnlp-main.49/.

Dehghani, M., Gouws, S., Vinyals, O., Uszkoreit, J., and Kaiser, L. Universal transformers. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HyzdRiR9Y7.

Fan, Y., Du, Y., Ramchandran, K., and Lee, K. Looped transformers for length generalization. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=2edigk8yoU.

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. In *Advances in Neural Information Processing Systems*, 2025. URL https://neurips.cc/virtual/2025/poster/117966.

Golowich, N., Jelassi, S., Brandfonbrener, D., Kakade, S. M., and Malach, E. The Role of Sparsity for Length Generalization in Transformers. *arXiv preprint arXiv:2502.16792*, 2025.

Graves, A. Adaptive computation time for recurrent neural networks, 2016. URL https://arxiv.org/abs/1603.08983.

Huang, X., Yang, A., Bhattamishra, S., Sarrof, Y., Krebs, A., Zhou, H., Nakkiran, P., and Hahn, M. A Formal Framework for Understanding Length Generalization in Transformers. In *The Thirteenth International Conference on Learning Representations*, 2024.

Hupkes, D., Dankers, V., Mul, M., and Bruni, E. Compositionality decomposed: How do neural networks generalise? *Journal of Artificial Intelligence Research*, 67:757–795, 2020. doi: 10.1613/jair.1.11674. URL https://doi.org/10.1613/jair.1.11674.

Jelassi, S., d'Ascoli, S., Domingo-Enrich, C., Wu, Y., Li, Y., and Charton, F. Length Generalization in Arithmetic Transformers. *arXiv preprint arXiv:2306.15400*, 2023.

Kazemnejad, A., Padhi, I., Natesan, K., Das, P., and Reddy, S. The Impact of Positional Encoding on Length Generalization in Transformers. In *Thirty-Seventh Conference on Neural Information Processing Systems*, 2023.

Lake, B. M. and Baroni, M. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *International Conference on Machine Learning*, 2018. URL http://proceedings.mlr.press/v80/lake18a.html.

Lee, N., Sreenivasan, K., Lee, J. D., Lee, K., and Papailiopoulos, D. Teaching arithmetic to small transformers, 2023. URL https://arxiv.org/abs/2307.03381.

Lu, W., Yang, Y., Lee, K., Li, Y., and Liu, E. Latent chain-of-thought? decoding the depth-recurrent transformer, 2025. URL https://arxiv.org/abs/2507.02199.

McLeish, S., Bansal, A., Stein, A., Jain, N., Kirchenbauer, J., Bartoldson, B. R., Kailkhura, B., Bhatele, A., Geiping, J., Schwarzschild, A., and Goldstein, T. Transformers Can Do Arithmetic with the Right Embeddings. *Advances in Neural Information Processing Systems*, 37:108012–108041, 2024.

Moosa, I. M., Lohit, S., Wang, Y., Chatterjee, M., and Yin, W. Understanding dynamic compute allocation in recurrent transformers, 2026. URL https://arxiv.org/abs/2602.08864.

Ontañón, S., Ainslie, J., Fisher, Z., and Cvicek, V. Making transformers solve compositional tasks. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3591–3607. Association for Computational Linguistics, 2022. doi: 10.18653/v1/2022.acl-long.251. URL https://aclanthology.org/2022.acl-long.251/.

Press, O., Smith, N., and Lewis, M. Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation. In *International Conference on Learning Representations*, 2021.

Quirke, P. and Barez, F. Understanding addition in transformers, 2024. URL https://arxiv.org/abs/2310.13121.

Ruoss, A., Delétang, G., Genewein, T., Grau-Moya, J., Csordás, R., Bennani, M., Legg, S., and Veness, J. Randomized Positional Encodings Boost Length Generalization of Transformers. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 1889–1903. Association for Computational Linguistics, 2023.

Sabbaghi, M., Pappas, G., Hassani, H., and Goel, S. Explicitly encoding structural symmetry is key to length generalization in arithmetic tasks, 2024. URL https://arxiv.org/abs/2406.01895.

Saunshi, N., Dikkala, N., Li, Z., Kumar, S., and Reddi, S. J. Reasoning with latent thoughts: On the power of looped transformers. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2502.17416.

Soulos, P., Terzic, A., Hersche, M., and Rahimi, A. Recurrent Transformers Trade-off Parallelism for Length Generalization on Regular Languages. In *The First Workshop on System-2 Reasoning at Scale, NeurIPS'24*, 2024.

Wang, J., Ji, T., Wu, Y., Yan, H., Gui, T., Zhang, Q., Huang, X., and Wang, X. Length Generalization of Causal Transformers without Position Encoding. In *Findings of the Association for Computational Linguistics: Acl 2024*, pp. 14024–14040. Association for Computational Linguistics, 2024.

Zhao, L., Feng, X., Feng, X., Zhong, W., Xu, D., Yang, Q., Liu, H., Qin, B., and Liu, T. Length Extrapolation of Transformers: A Survey from the Perspective of Positional Encoding. In *Findings of the Association for Computational Linguistics: Emnlp 2024*, pp. 9959–9977. Association for Computational Linguistics, 2024.

Zhou, H., Bradley, A., Littwin, E., Razin, N., Saremi, O., Susskind, J. M., Bengio, S., and Nakkiran, P. What Algorithms can Transformers Learn? A Study in Length Generalization. In *The Twelfth International Conference on Learning Representations*, 2023.

Zhou, Y., Alon, U., Chen, X., Wang, X., Agarwal, R., and Zhou, D. Transformers can achieve length generalization but not robustly, 2024. URL https://arxiv.org/abs/2402.09371.

A. Finite-Cap PonderNet Implementation

In the experiments, looped computation is capped at T recurrent steps. For token i and step n, the halting head produces the conditional halt probability λ_i^n. The finite-cap halt distribution used in the objective is

$$p_i(n) = \begin{cases} \lambda_i^n \prod_{s<n}(1-\lambda_i^s), & n < T, \\ \prod_{s<T}(1-\lambda_i^s), & n = T. \end{cases} \tag{4}$$

The final case assigns any remaining unhalted probability mass to the last allowed recurrent step. The prior q_{λ_p} in Equation (3) uses the same finite-cap form with every conditional halt probability set to the geometric-prior parameter λ_p.

B. Extended Related Work

Compositional and length generalization. Compositional generalization asks whether neural models learn reusable rules rather than correlations tied to a training distribution (Lake & Baroni, 2018; Hupkes et al., 2020). Length generalization is a direct version of this question because the model must apply the same computation more times. Formal and empirical work studies which algorithms transformers can learn under length shift, and how failures relate to position, sparsity, or formal-language structure (Zhou et al., 2023; Huang et al., 2024; Golowich et al., 2025). In arithmetic, this becomes the question of whether a model trained on short additions has learned the digit-wise rule productively enough to apply it to longer numbers.

Task coordinates for length extrapolation. Many successful length-generalization methods can be viewed as supplying or improving a coordinate system. General-purpose approaches modify, randomize, or remove positional encodings through ALiBi, randomized positional encodings, NoPE, or temperature and architectural changes to positional extrapolation (Press et al., 2021; Ruoss et al., 2023; Kazemnejad et al., 2023; Wang et al., 2024; Zhao et al., 2024). Arithmetic-specific approaches expose more task structure. Abacus embeddings provide digit-position information (McLeish et al., 2024); position coupling shares IDs among same-significance digits (Cho et al., 2024); task hinting supplies auxiliary subproblems (Awasthi & Gupta, 2023); and symmetry-based encodings build arithmetic structure into the representation (Sabbaghi et al., 2024). These methods can be very effective, including for extrapolating over both operand length and operand count (Cho et al., 2025). NoPE is used here in this literal sense, as the absence of positional embeddings, not as an additional coordinate. Our experiment removes explicit positional encodings, digit-index hints, scratchpad targets, carry labels, input injection, and long-length priming, and asks whether a useful coordinate can emerge from recurrence. Table 1 summarizes representative addition settings.

Algorithmic formats and program simplicity. Another line of work explains length generalization through the simplicity of the algorithm a transformer must implement. Zhou et al. (2023) relate length generalization to short RASP-L programs and show that scratchpad formats can help or hurt depending on whether they simplify the underlying program. Task hinting similarly makes intermediate structure more available to the model (Awasthi & Gupta, 2023). Looped transformers provide a different way to create intermediate computation. Fan et al. (2025) show strong length generalization on tasks with n-RASP-L solutions, including binary addition, using looped transformers with step-dependent training and adaptive inference. Our setting differs by using autoregressive decimal addition with a learned tokenwise halting policy and without explicit arithmetic coordinates. The computation frontier can therefore be viewed as a latent, recurrent analogue of an ordered scratchpad. Intermediate structure is not written as tokens, but appears over loop iterations.

Recurrence, halting, and latent computation. Adaptive Computation Time, Universal Transformers, and PonderNet introduced or developed learned recurrence and halting mechanisms (Graves, 2016; Dehghani et al., 2019; Banino et al., 2021). PonderNet is not merely another halting head in this lineage: Banino et al. explicitly present it as a probabilistic reformulation of ACT, motivated by ACT's instability, hyperparameter sensitivity, and biased computation-cost gradient. Its expected-loss objective and KL prior over halting depth give us a simple way to train tokenwise stopping distributions while keeping the recurrent block itself generic. Prior work gives evidence that recurrence, weight sharing, and other architectural details can help systematic or length generalization, while also showing that details matter (Dehghani et al., 2019; Csordas et al., 2021; Ontañón et al., 2022; Fan et al., 2025; Soulos et al., 2024). Recent looped or recurrent models further motivate latent depth as a reasoning axis (Saunshi et al., 2025; Geiping et al., 2025). At the same time, dynamic-compute analyses caution that spending more computation on difficult tokens is not sufficient evidence for algorithmic extrapolation (Moosa et al., 2026). Our contribution is to connect recurrent depth to a specific internal schedule over digit positions.

Mechanistic analysis of arithmetic and recurrent models. Prior mechanistic work has found structured digit-wise mechanisms in fixed-depth addition transformers (Quirke & Barez, 2024). TunedLens asks when intermediate states become readable by a model's own output head (Belrose et al., 2023). Recent work probes recurrent language models for latent reasoning and loop dynamics (Lu et al., 2025;

Blayney et al., 2026). Our analysis follows this spirit in a controlled arithmetic setting where the central object is not only final accuracy, but the emergence of a readable computation frontier in recurrent time.

C. Representative Addition Baselines

Table 1 contextualizes the behavioral result against representative addition baselines. The cited systems differ in architecture, data format, supervision, positional information, and evaluation protocol. The main point is that higher extrapolation factors in prior arithmetic work often come with stronger task-specific structure, whereas our setting removes explicit arithmetic coordinates and uses the baseline table to make those assumptions visible.

D. Across Training Regimes

Table 2 summarizes the length-generalization result. The table reports operand lengths; answer lengths can be one digit longer because of a final carry. All three PonderNet regimes extrapolate beyond the maximum trained operand length, and the largest trained regime gives the largest absolute extrapolation range. The short- and long-regime training curves in Appendix F show the same qualitative OOD learning pattern. The result is moderate compared with highly engineered arithmetic systems, but strong for a model trained from scratch without explicit digit-alignment aids.

E. Additional Behavioral Figures

Figures 4 and 5 show the mid-regime diagnostics omitted from the main text: digit accuracy, selected halt steps, answer-length accuracy, and carry-stratified accuracy. Figures 6 and 7 show the corresponding behavioral evaluations for the short and long regimes. Together, these figures support the same qualitative conclusion as the mid-regime exact-match heatmap in Figure 1: accuracy remains high beyond the training boundary, selected halt steps increase with answer length, and carry-stratified accuracy declines near the same failure region rather than collapsing immediately out of distribution.

F. Additional Training Curves

Figure 8 shows the training dynamics for the short and long PonderNet regimes. Together with the mid-regime curves in Figure 2, these runs show the same qualitative pattern across operand-length ranges: in-distribution accuracy rises first, OOD accuracy on the held-out length band follows, and the trained PonderNet models generalize out of distribution. The fixed-loop no-halting ablation in Figure 2 is the exception among these training-curve evaluations.

G. Frontier Analysis Details

For the frontier analysis we collect residual states at the predictor position for each answer digit, namely the token position immediately before that digit. We use 500 probe-train examples and 400 disjoint probe-evaluation examples, sampled uniformly. Answer-position probes predict k with multinomial logistic regression, and carry probes predict the carry-in bit c_k computed from the ground-truth operand digits. Classifiers are trained separately for each recurrent step, using class-balanced subsampling. For probe target z_i and step n, the fitted classifier is

$$W_n^* = \arg\min_W \sum_i \mathrm{CE}(\mathrm{softmax}(W x_i^n), z_i)\,. \tag{5}$$

For a scalar observable $m(t, k)$, such as correct-token probability or halt probability, we define a frontier time $\tau(k)$ as the first loop iteration at which the observable crosses a threshold or reaches a fixed fraction of its dynamic range. A roughly linear relation

$$\tau(k) \approx \alpha k + \beta_0 \tag{6}$$

means that later answer digits become active later in recurrent time. The important point is not the exact fitted slope, but the agreement among independent observables. A single shared answer-position probe trained on states pooled across all recurrent steps did not yield a useful position decoder, so the reported probe frontiers should be read as phase-specific decodability rather than as a stable absolute position code.

H. Configuration and Evaluation Summary

Table 3 lists the active checkpoint settings and paper-evaluation caps for the three PonderNet length-generalization regimes. The behavioral grids, answer-length diagnostics, and frontier plots use one final checkpoint per regime; the training-curve figures aggregate the three seeds. All runs use answer-span-only loss, pad sequences to the configured block size, use cumulative halting for the paper evaluations with threshold 0.2, and train with gradient accumulation 1, AdamW weight decay 0.1, $\beta_2 = 0.99$, 500 warmup iterations, and cosine decay to $\eta_{\min} = \eta/10$.

Table 1. **Representative arithmetic length-generalization settings.** Reported extrapolation factors are not directly comparable because the tasks, formats, architectures, reporting conventions, and externally supplied task information differ. The table is intended to contextualize our stricter no-explicit-coordinate setting rather than to define a leaderboard.

Work	Setup	Reported addition result	Coordinate / side information
Lee et al. (2023)	Decoder-only small transformers trained from scratch with next-token prediction; studies plain, reversed-output, and scratchpad formats.	Reverse and scratchpad formats improve in-distribution learning, but the paper reports failures to generalize to unseen digit lengths, such as training on 1–7 digit addition and failing on 8-digit addition.	Formatting and scratchpads.
Jelassi et al. (2023)	Encoder-only transformer and Universal Transformer models trained from scratch as supervised output-position classifiers; uses padded operands and outputs with relative position embeddings.	Training on up to 5-digit addition reaches high accuracy at 15 digits, but performance drops strongly by 20 digits.	Relative position embeddings; encoder-style output classification.
Zhou et al. (2024)	Decoder-only transformer with reversed format, index hints, FIRE position encodings, randomized positional encodings, and best-of-10 reporting.	Training on 1–40 digit addition reaches more than 98% exact match at 100 digits, a $2.5\times$ factor, but with high seed and data-order sensitivity.	Index hints and positional recipe.
McLeish et al. (2024)	Decoder-only models with Abacus digit-position embeddings; experiments also use input injection and recurrent or looped variants.	Training on up to 20-digit operands reaches 99% exact match on 100-digit addition, and the paper reports generalization to 120 digits in some settings.	Digit-position embeddings; often input injection and recurrence.
Cho et al. (2024)	Decoder-only transformers with position coupling, where same-significance digits receive shared position IDs; uses reversed format and zero padding.	Training on 1–30 digit addition reaches more than 95% exact match up to 200 digits, a $6.67\times$ factor.	Same-significance position IDs.
Sabbaghi et al. (2024)	Transformer with modified number formatting and custom positional encodings designed to encode arithmetic symmetries.	Training on at most 5-digit numbers generalizes up to 50-digit addition and multiplication.	Custom structural encodings.
Cho et al. (2025)	Multi-operand addition and multiplication with task-specific scratchpads and multi-level position coupling.	Approximately 2–3$\times$ length generalization over operand length and operand count tasks.	Scratchpads and multi-level position coupling.

Table 2. **Behavioral length generalization.** Each model is trained only on the operand-length range shown. We report the longest max-operand-length ring for which all rings up to that length retain greater than 99% mean exact-match accuracy.

Regime	Train	OOD	Factor	Mean EM
Short	1–10	13	$1.3\times$	100.0%
Mid	1–20	34	$1.7\times$	99.7%
Long	1–40	72	$1.8\times$	99.5%

Table 3. **Configuration summary.** All models are one-layer adaptive looped transformers with 2 attention heads, embedding dimension 128, dropout 0.1, NoPE attention, no input injection, no loop embeddings, zero-initialized residual projections, and tokenwise PonderNet halting.

Setting	Short	Mid	Long
Train operands	1–10	1–20	1–40
Context size	70	100	160
Batch size	512	512	256
Max iters	80k	80k	150k
Checkpoint loop cap	50	40	55
Behavioral eval loop cap	60	120	200
Probe trace loop cap	30	60	120
Learning rate η	3×10^{-4}	3×10^{-4}	3×10^{-4}
Min LR $\eta_{\min}$	3×10^{-5}	3×10^{-5}	3×10^{-5}
PonderNet λ_p	0.5	0.25	0.125
PonderNet β	0.01	0.01	0.01

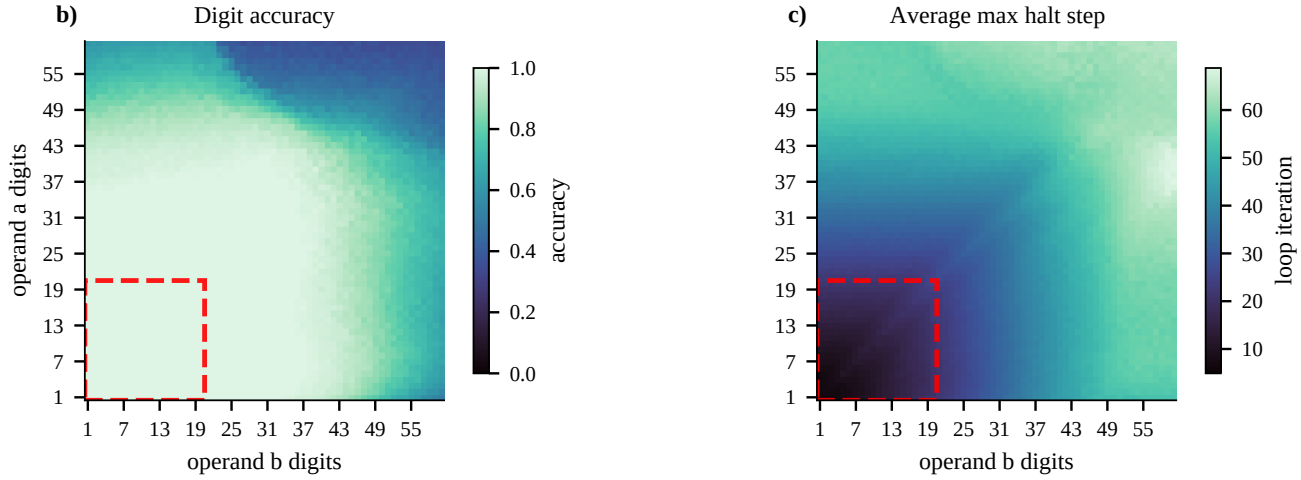

Figure 4. **Mid-regime supplemental operand-length diagnostics.** Panels (b) and (c) continue Figure 1: digit accuracy and average maximum selected halt step over the same operand-length grid. The red guides mark the training boundary.

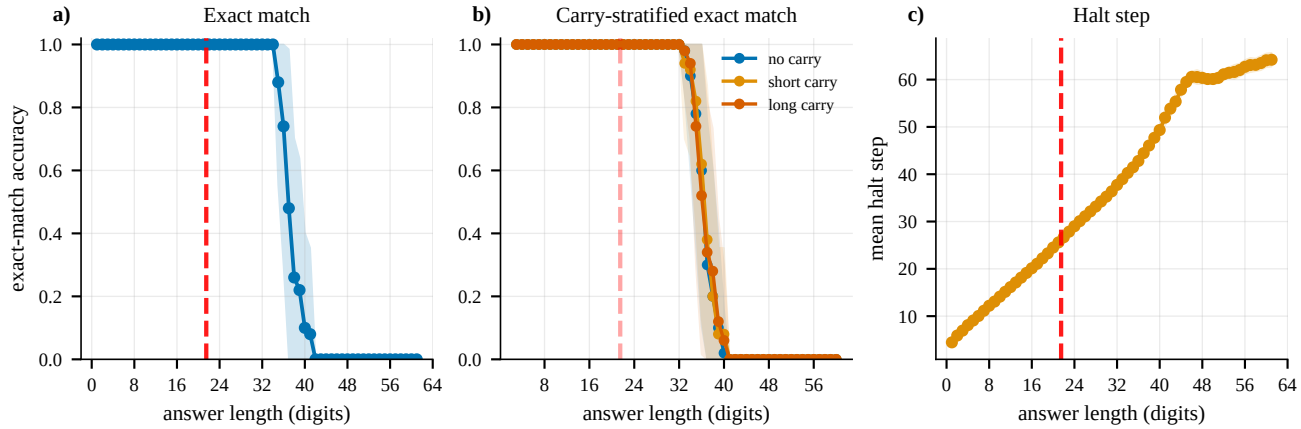

Figure 5. **Mid-regime answer-length accuracy, carry robustness, and computation growth.** Panels show exact-match accuracy by answer length, carry-stratified exact-match accuracy, and mean selected halt step. Long-carry examples remain close to the other carry groups until the model approaches its failure boundary. The halt-step curve shows that the model uses more recurrent computation on longer generated answers.

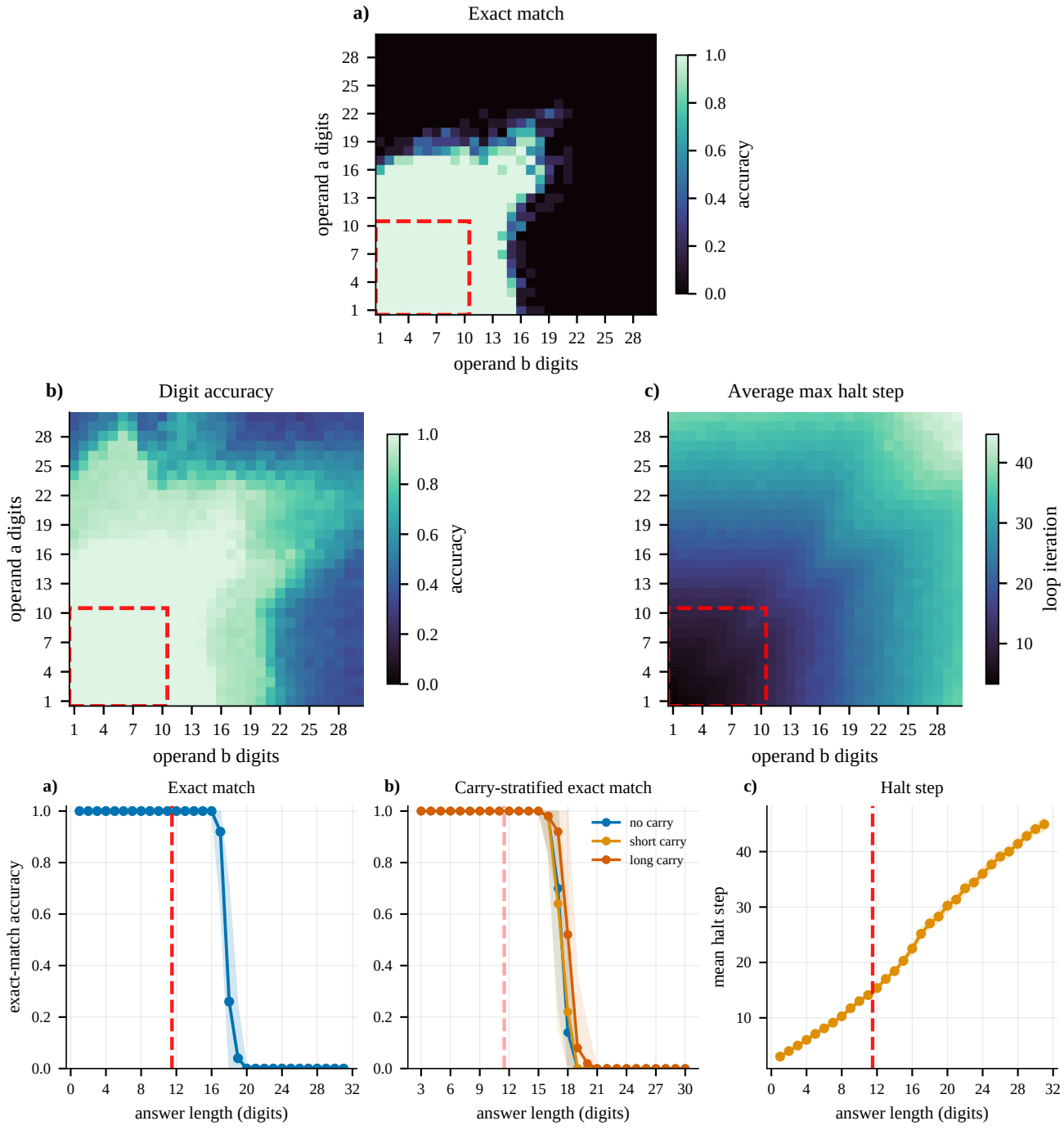

Figure 6. **Short-regime behavioral evaluation.** Top: exact-match accuracy over operand lengths. Middle: digit accuracy and average maximum selected halt step. Bottom: answer-length and carry-stratified diagnostics. The model is trained on operands up to 10 digits.

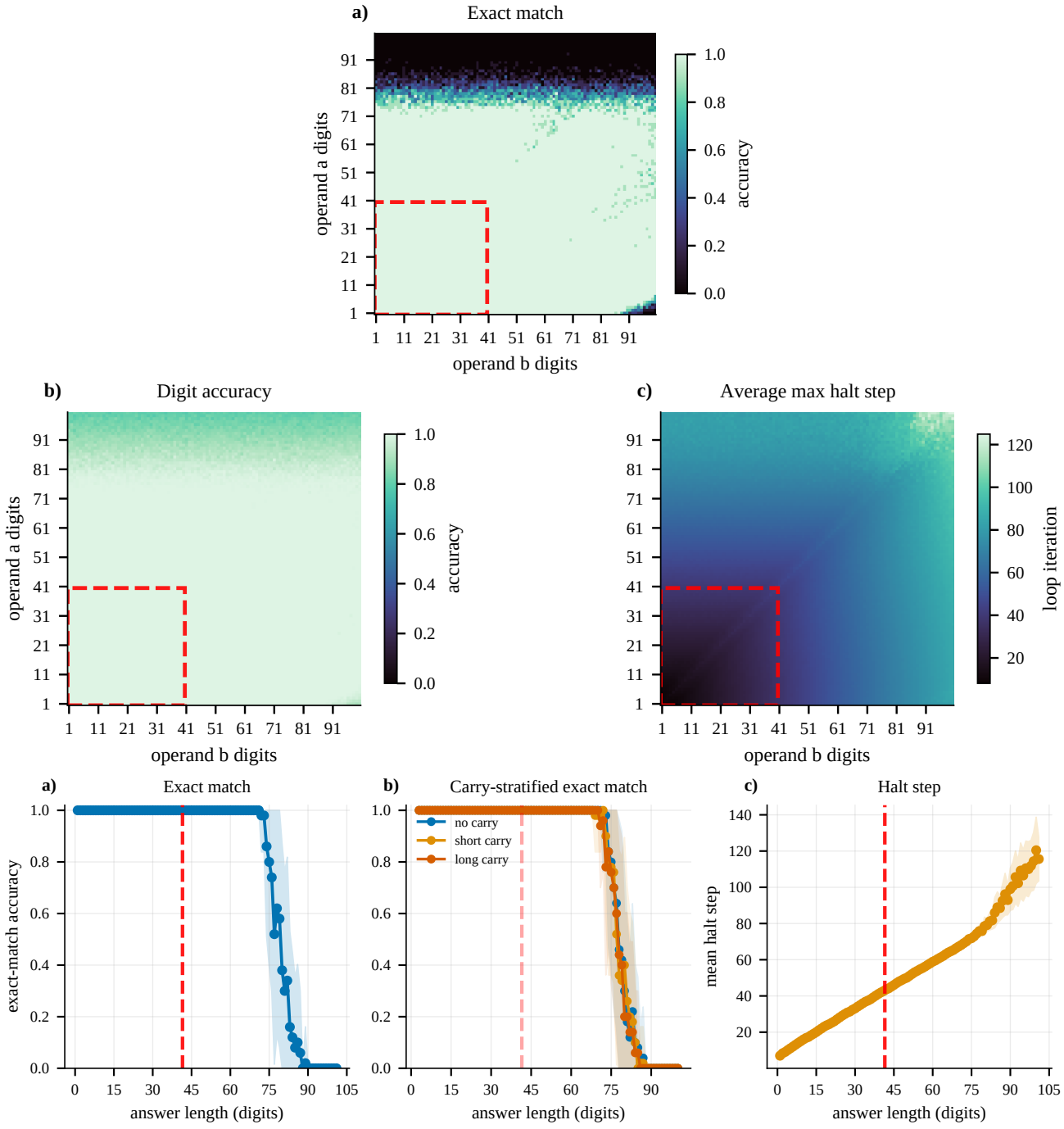

Figure 7. **Long-regime behavioral evaluation.** Top: exact-match accuracy over operand lengths. Middle: digit accuracy and average maximum selected halt step. Bottom: answer-length and carry-stratified diagnostics. The model is trained on operands up to 40 digits.

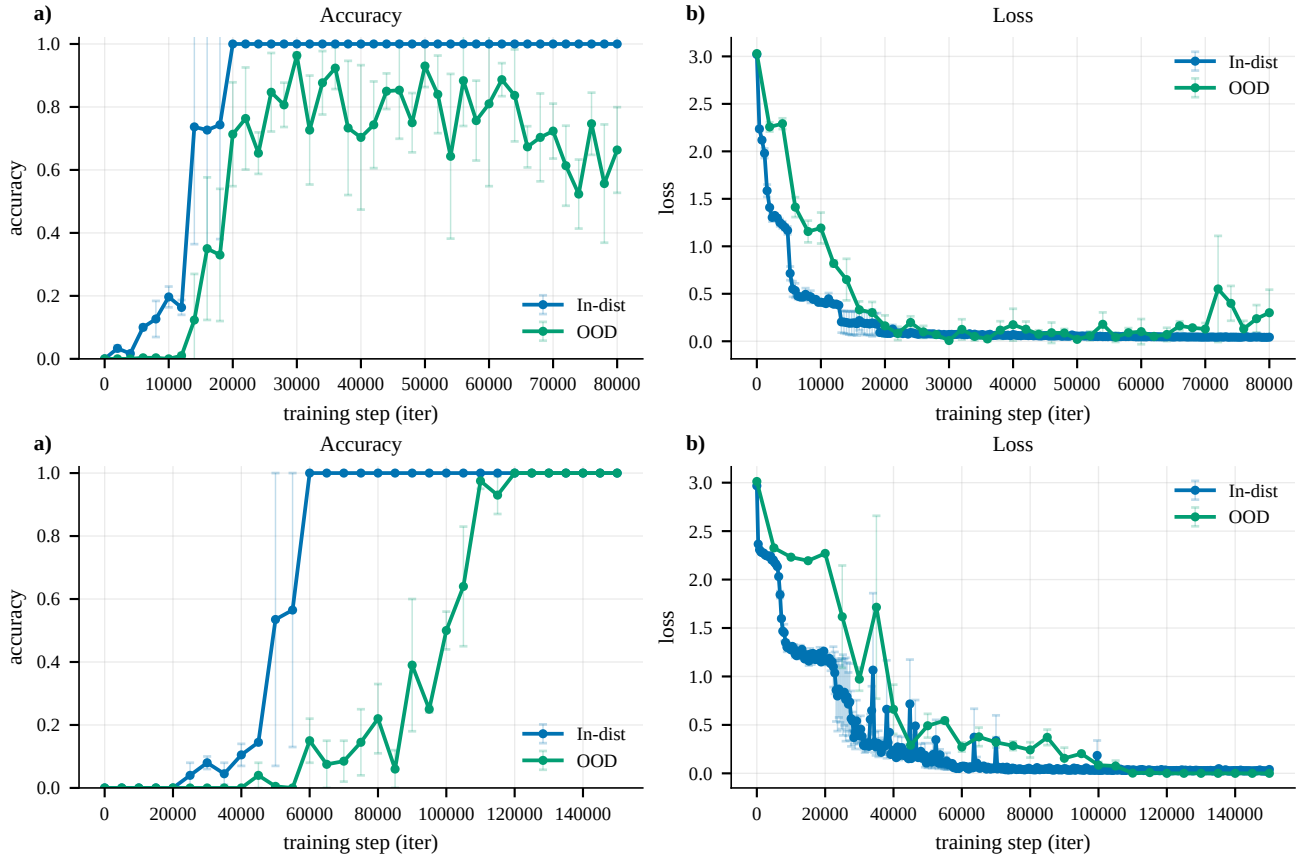

Figure 8. **Short- and long-regime training curves.** The panels use the same accuracy and loss metrics as Figure 2; only PonderNet runs are shown here. Each curve is the mean over three PonderNet seeds and error bars show one standard deviation across seeds. The OOD validation stream samples the ten held-out operand lengths immediately above training: 11–20 digits for the short regime and 41–50 digits for the long regime. Both regimes acquire OOD accuracy during training; Table 2 reports the broader operand-grid evaluation used for the final length-generalization summary.