Transformers Pretrained on Procedural Data Contain Modular Structures for Algorithmic Reasoning

Zachary Shinnick [1] **Liangze Jiang** [2 3] **Hemanth Saratchandran** [1] **Anton van den Hengel** [1] **Damien Teney** [3]

Abstract

Context. Pretraining on large, semantically rich datasets is key for developing language models. Surprisingly, recent studies have shown that even synthetic data, generated procedurally through simple semantic-free algorithms, can yield some of the same benefits as natural language pretraining. It is unclear *what* specific capabilities such simple synthetic data instils in a model, *where* these capabilities reside in the architecture, and *how* they manifest within its weights.

Findings. In this short paper, we identify several beneficial forms of procedural data, together with specific algorithmic reasoning skills that improve in small transformers. Our core finding is that different procedural rules instil *distinct but complementary inductive structures* in the model. With extensive ablations and partial-transfer experiments, we discover that these structures reside in different parts of the model. Attention layers often carry the most transferable information, but some pretraining rules impart useful structure to MLP blocks instead. Most interestingly, the structures induced by multiple rules can be composed to jointly reinforce multiple capabilities.

Implications. These results suggest an exciting possibility of disentangling the acquisition of knowledge from reasoning in language models, with the goal of improving their robustness and data efficiency.

1. Introduction

What properties of pretraining data enable language models to acquire both knowledge and reasoning capabilities? While the size and diversity of the data are empirically crucial for learning factual knowledge (Longpre et al., 2024), far less is understood about the structural and distributional characteristics required to acquire reasoning abilities.

The value of structured data. Empirically, it has been observed that pretraining language models on computer code has proven highly effective, likely due to the inherent compositional and recursive structure present in code (Petty et al., 2024). Recent work suggests that procedural data, generated from simple algorithm rules [4] can offer similar benefits. For instance, Hu et al. (2025) demonstrate that formal language data provides greater value per token than natural language when training a 1B-parameter model. Likewise, Zhang et al. (2024) show that data generated by cellular automata can accelerate the learning of abstract reasoning tasks and yield modest improvements in chess move prediction.

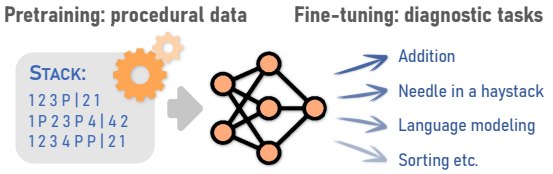
This paper. We aim to better understand the mechanisms at play when pre-training transformer-based sequence models on procedural data (see Figure 1). We characterise and locate useful structures created in such pretrained models that facilitate subsequent fine-tuning and generalisation on algorithmic reasoning and language modelling tasks. The literature contains many results on the pretraining of language models with data from formal languages and simple algorithms (see Appendix A), down to a simple identity function (Wu et al., 2022), i.e. simply learning to repeat the input. It is conceivable that such pretraining improves over a random initialisation simply by adjusting the overall magnitude of the weights (Huang et al., 2020). We design experiments to distinguish such trivial effects from the genuine learning of transferrable mechanisms.

Summary of findings. We bring elements of answers to the following questions.

[1]Australian Institute for Machine Learning (AIML), University of Adelaide, Australia [2]École Polytechnique Fédérale de Lausanne (EPFL), Switzerland [3]Idiap Research Institute, Switzerland. Correspondence to: Zachary Shinnick <zachary.shinnick@adelaide.edu.au>.

[4]We distinguish *procedural* data, generated from explicit algorithms, from *synthetic* data, from learned models e.g. other LLMs.

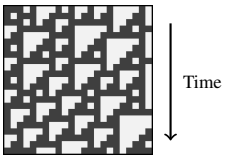
Accepted at Methods and Opportunities at Small Scale (MOSS), ICML 2025, Vancouver, Canada.



(a)

Pretraining task	Example sequence
k-DYCK	([{ }])
k-DYCK SHUFFLE	([{]) }
STACK	1 2 3 P | **2 1**
IDENTITY	1 2 3 | **1 2 3**
SET	1 2 2 | **1 2**

(b)



(c) ECA Rule 110

Figure 1: (a) We pretrain small transformers on various forms of procedural data, then fine-tune them on a series of diagnostic tasks. The data is generated from formal languages (b) or simple algorithms such as elementary cellular automata (c). In k-DYCK examples, matching brackets are color-coded. For STACK, 'P' is the *pop* operation. For STACK, IDENTITY, and SET, '|' acts as a separator between the input and the expected output, on which the loss is computed (**bold** tokens).

1. **What** is gained from procedural data? (Section 3) We pretrain small transformers on diverse forms of procedural data and fine-tune them on a series of diagnostic tasks. *We find that specific types of procedural data significantly enhance particular capabilities for algorithmic reasoning.*
2. **Where** does the useful knowledge reside in the pretrained models? (Section 4.1) We evaluate the partial transfer of pretrained weights. *We find that the beneficial effects of pretraining lie in different parts of the architecture*, depending on the task, with the most pronounced benefits often found in the attention layers.
3. **How** does the pretraining improve these downstream capabilities? (Section 4.2) We evaluate various perturbations of the pretrained weights to understand how they encode useful information. For some tasks (e.g. SORTING), even shuffled weights preserve *some* benefits, but all tasks undergo a significant performance drop with any kind of perturbation (e.g. Gaussian noise). This indicates that *pretraining creates non-trivial inductive structures* in attention and/or MLP weights. Moreover, *these structures prove remarkably modular*. The attention layers of a pretrained model can be combined with MLPs of another to construct a better initialization that improves multiple algorithmic reasoning capabilities (Section 5).

We discuss exciting potential applications of these results in Section 6. A longer version of this paper is also in preparation.

2. Methodology

For each experiment, we train a small transformer on one form of procedural data then fine-tune it on one diagnostic task.

Procedural pretraining. We pretrain the model for standard next-token prediction on data generated from formal languages and simple algorithms (Figure 1a). The following selection is motivated by prior work on procedural data. See Appendix B for details. (1) k-DYCK: a formal language of nested brackets. (2) k-DYCK SHUFFLE: a variant with matching braces that are not necessarily nested. (3) STACK, a simulation of a stack memory; the model must predict the final contents of the memory. (4) IDENTITY: the model must repeat the input. (5) SET: the model must remove repeated tokens from the input. (6) Complex elementary cellular automaton (ECA RULE 110): a binary sequence with deterministic Markovian evolution.

Fine-tuning on diagnostic tasks. Prior work with procedural data focused on language modelling (Appendix A) while we aim to identify specific improved skills. We thus fine-tune each model on tasks that span a range of algorithmic reasoning capabilities: memory recall (NEEDLE-IN-A-HAYSTACK), arithmetic (ADDITION, REVERSED ADDITION, MULTIPLICATION), sorting (SORTING), and handling of natural language (LANGUAGE MODELLING). See Appendix B.3 for details.

Architecture. Our focus is on the data, hence we use a simple GPT-2-type architecture (Radford et al., 2019), with 2 layers, 4 attention heads, and a hidden size of 16, or a larger configuration for more challenging tasks (MULTIPLICATION, LANGUAGE MODELING). See Appendix B.2 and B.4 for details on hyperparameters.

Weight transfer. We denote the weights of a transformer model as $\mathcal{T} = (\mathbf{E}, \mathbf{A}, \mathbf{F})$ where $\mathbf{E}$ corresponds to the embedding and unembedding layers (tokens and positions), $\mathbf{A} = \{\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(L)}\}$ to the attention, and $\mathbf{F} = \{\mathbf{F}^{(1)}, \ldots, \mathbf{F}^{(L)}\}$ to the MLPs across L layers. When the vocabulary of the pretraining and fine-tuning tasks cannot be aligned, we reset the token embeddings to average pretrained vector (Hewitt, 2021), as an unbiased initialisation before fine-tuning. The positional embeddings are transferred if the pretrained context length is sufficient, or randomly initialised otherwise.

3. Different Procedural Tasks Improve Distinct Reasoning Capabilities

Prior work on procedural data demonstrates its value for general language modelling (Hu et al., 2025). However it is not clear what exact skills improve in this setting. In this section, we show that different forms of procedural data improve specific capabilities for algorithmic reasoning. See Appendix B–C for additional details and results.

Setup. We first pretrain a model $\mathcal{T}_{\text{pre}} = (\mathbf{E}_{\text{pre}}, \mathbf{A}_{\text{pre}}, \mathbf{F}_{\text{pre}})$ on procedural data. We then transfer all weights from $\mathcal{T}_{\text{pre}}$ into a new model $\mathcal{T}_{\text{full}} = (\mathbf{E}_{\text{pre}}, \mathbf{A}_{\text{pre}}, \mathbf{F}_{\text{pre}})$, referred to as "full-model" transfer. $\mathcal{T}_{\text{full}}$ is then fine-tuned on a downstream diagnostic task. We compare its performance to a randomly initialised baseline $\mathcal{T}_{\text{rand}} = (\mathbf{E}_{\text{rand}}, \mathbf{A}_{\text{rand}}, \mathbf{F}_{\text{rand}})$ that undergoes the same fine-tuning. For LANGUAGE MODELLING, we increase the hidden size to 64. For MULTIPLICATION, we use the larger `gpt2-mini` architecture[5] to enable a comparison with a model pretrained on OpenWebText.

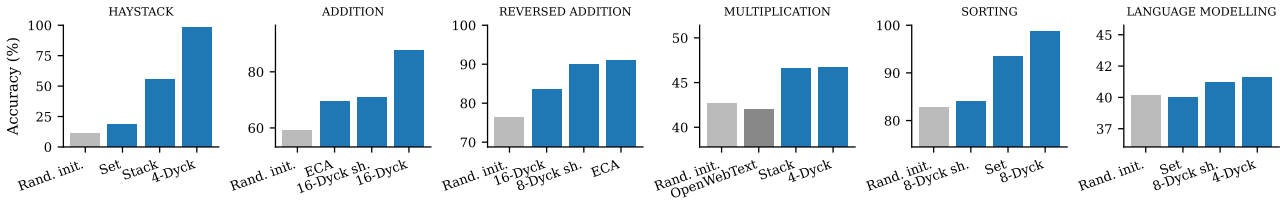


Figure 2: Models pretrained on carefully-chosen procedural data (blue) significantly improve specific downstream diagnostic tasks, compared to a random initialisation (light gray) or pretraining on natural language (dark gray, shown for MULTIPLICATION). See Appendix C for full results, including cases where procedural pretraining brings no benefit.

Results. Figure 2 shows that procedurally pretrained models can largely outperform randomly initialized ones for every downstream task, confirming that **procedural pretraining data can instil meaningful "soft inductive biases"** in transformers. Furthermore, we observe that the different forms of procedural data yield different improvements across the downstream tasks. This indicates that **each procedural task imparts distinct inductive biases** that are each relevant to different downstream tasks. For example, for HAYSTACK, pretraining on k-DYCK can inject an effective memory recall capability (in order to match the nested brackets), which cannot be provided by SET, k-DYCK SHUFFLE or ECA (see also Table 3). Notably, for MULTIPLICATION, procedural pretraining outperforms pretraining on OpenWebText.

4. Characterising the Effects of Procedural Pretraining

4.1. The Benefits from Pretraining Often Reside in Specific Architectural Components

Procedural pretraining creates useful "soft inductive biases" in a model, but where do they reside? This section shows that different forms of procedural data act on different parts of the architecture.

Setup for selective transfer. Given a procedurally-pretrained model $\mathcal{T}_{\text{pre}} = (\mathbf{E}_{\text{pre}}, \mathbf{A}_{\text{pre}}, \mathbf{F}_{\text{pre}})$ from Section 3, we initialise models for fine-tuning with selected components: attention-only transfer $\mathcal{T}_{\text{attn}} = (\mathbf{E}_{\text{rand}}, \mathbf{A}_{\text{pre}}, \mathbf{F}_{\text{rand}})$, MLP-only transfer $\mathcal{T}_{\text{mlp}} = (\mathbf{E}_{\text{rand}}, \mathbf{A}_{\text{rand}}, \mathbf{F}_{\text{pre}})$, or full-model transfer $\mathcal{T}_{\text{full}} = (\mathbf{E}_{\text{pre}}, \mathbf{A}_{\text{pre}}, \mathbf{F}_{\text{pre}})$. In all cases, the *entire* model is fine-tuned.

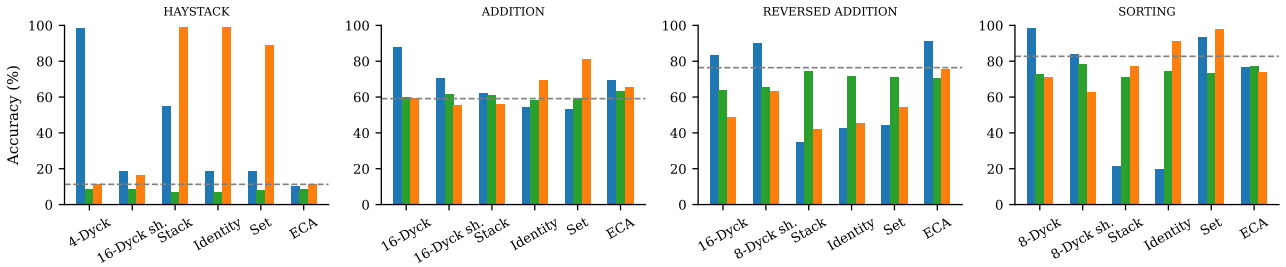


Figure 3: Results of the selective transfer of procedurally-pretrained weights. Each color is a different type of transfer: $\mathcal{T}_{\text{full}}$, $\mathcal{T}_{\text{mlp}}$, $\mathcal{T}_{\text{attn}}$. The horizontal dotted line is the baseline performance with a random initialisation. See Appendix C for full results including the variance across seeds.

Results. Figure 3 shows that selective transfer often outperforms full transfer, showing that **the useful inductive biases reside in specific parts of the model**. $\mathcal{T}_{\text{attn}}$ frequently yields the best performance, and in some cases substantially so. With IDENTITY pretraining for HAYSTACK, $\mathcal{T}_{\text{attn}}$ improves by 80 percentage points over $\mathcal{T}_{\text{full}}$ (18.8% $\rightarrow$ 99.0%). This indicates that the attention layers encode a useful inductive bias for algorithmic reasoning, while MLP weights encode mechanisms specific to the pretraining that do not transfer. Similar observations are made for STACK and SET pretraining for HAYSTACK. In contrast, on REVERSED ADDITION, $\mathcal{T}_{\text{mlp}}$ and $\mathcal{T}_{\text{full}}$ outperform $\mathcal{T}_{\text{attn}}$.

These results generally show that the inductive biases from procedural pretraining reside in different parts of the model, and that **they can be transferred independently**, with the attention layers being the most consistently transferable carrier.

[5] `https://huggingface.co/erwanf/gpt2-mini`

4.2. Pretraining Creates Soft Inductive Biases in Precisely Structured Weights

Transformers are sensitive to the initialization: simply adjusting the magnitude of random weights has a big impact (Huang et al., 2020). We aim to distinguish such trivial effects from the genuine learning of transferrable soft inductive biases.

Setup. We introduce two types of perturbations to pretrained weights and examine the resulting performance drop after fine-tuning. (1) Additive Gaussian noise checks whether inductive biases are encoded in precise weight structures; (2) Random per-layer weight shuffling checks for benefits merely from the weight magnitudes. Shuffling destroys any precise structure but preserves the distribution of weight values. We apply the perturbations on the best transfer configurations from §4.1 and report a relative improvement score (1.0 corresponds to unperturbed pretrained weights, 0.0 to a random initialisation, $\mathcal{T}_{\text{rand}}$).

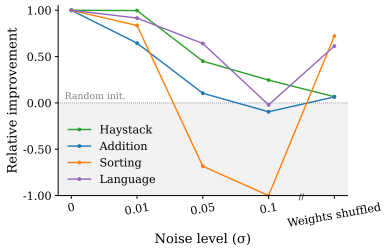


Figure 4: Relative downstream improvement using pretrained weights with perturbations (noise or shuffling). See Appendix C for full results and details.

Results. Gradually increasing noise consistently degrades performance, indicating that precise structure is crucial. HAYSTACK and ADDITION are highly sensitive to shuffling, suggesting that **pretraining encodes information beyond weight magnitudes**. In contrast, LANGUAGE MODELLING and SORTING are more robust, implying that **some cases partially benefit from adjusted magnitudes**.

5. Combining Multiple Pretraining Tasks

Since pretraining tasks act on different layers, can we combine components and benefits from several pretrained models?

Setup. We define $\mathcal{T}^1_{\text{pre}}$ and $\mathcal{T}^2_{\text{pre}}$ as two models pretrained on two types of procedural data, respectively. We then transfer specific components from the two models (e.g. the attention layers $\mathbf{A}_{\text{pre}}$ from $\mathcal{T}^1_{\text{pre}}$ and the MLP layers $\mathbf{F}_{\text{pre}}$ from $\mathcal{T}^2_{\text{pre}}$) to initialise a combined model $\mathcal{T}_{\text{comb}}$. The combined model is then fine-tuned on various downstream tasks.

Pretraining configuration	HAYSTACK	ADDITION	REVERSED ADDITION	SORTING
SET (Full transfer)	$18.9_{\pm 26.6}$	$53.4_{\pm 0.1}$	$44.6_{\pm 5.1}$	$93.5_{\pm 1.6}$
SET (Attention only)	$88.9_{\pm 27.1}$	$\mathbf{81.1}_{\pm 12.2}$	$54.4_{\pm 10.4}$	$98.1_{\pm 2.8}$
ECA (Full transfer)	$10.5_{\pm 0.5}$	$69.6_{\pm 7.9}$	$\mathbf{91.0}_{\pm 16.1}$	$76.9_{\pm 1.4}$
ECA (MLPs only)	$8.71_{\pm 1.0}$	$63.1_{\pm 14.4}$	$70.5_{\pm 31.6}$	$77.1_{\pm 8.1}$
SET (Attention) + ECA (MLPs)	$\mathbf{94.4}_{\pm 2.5}$	$\underline{80.3}_{\pm 13.9}$	$\underline{82.9}_{\pm 16.9}$	$\mathbf{99.4}_{\pm 0.2}$

Table 1: Comparison of a modularly composed model with models pretrained on individual procedural data types. Combining SET attention and ECA MLP layers performs strongly on all four tasks.

Results. In Table 1, we show an example with SET and ECA, where we combine the attention layers from SET and MLPs from ECA. Compared to other configurations that only rely on one procedural task, which fail at least on one downstream task, the combined model (last row) consistently performs reasonably well on all four tasks. This suggests that **the useful structures can be modularly composed** into a single "initialisation" that facilitates fine-tuning on multiple tasks.

6. Discussion and Open Questions

We showed that pretraining transformers on well-chosen procedural tasks creates useful structure in different parts of the architecture. We identified capabilities that significantly improve with specific forms of procedural data. We also verified that the improvements cannot be explained with trivial effects such as a better weight magnitude. These results open exciting questions and possibilities to take full advantage of procedural data.

Why are specific forms of procedural data helpful? We miss a first-principle explanation why specific forms of data help specific capabilities. E.g. why does HAYSTACK benefit from k-DYCK but not the SHUFFLE variant? How is pretraining with formal languages different from ECAs? Our positive and negative examples could support a comparative analysis.

Combining pretraining tasks. It is not clear which specific forms of procedural data could help training a generalist LLM. Data mixture optimization (Fan et al., 2023; Xie et al., 2023; Ye et al., 2024) could help balance multiple procedural rules. This procedural data could be merged with standard pretraining data, or instead be used as a "pre-pretraining" curriculum.

Closed-form initialisation. Expanding simple procedural rules into millions of training examples seems computationally wasteful. Can we characterise the resulting structure in pretrained models to directly instantiate it in initial weights?

Knowledge vs. reasoning. LLMs' difficulties to reason robustly may be rooted in entangled representations of knowledge and reasoning (Han et al., 2025). Procedural data could teach reasoning independently from specific semantic information.

References

Abnar, S., Dehghani, M., and Zuidema, W. Transferring inductive biases through knowledge distillation. *arXiv preprint arXiv:2006.00555*, 2020.

Aryabumi, V., Su, Y., Ma, R., Morisot, A., Zhang, I., Locatelli, A., Fadaee, M., Üstün, A., and Hooker, S. To code, or not to code? exploring impact of code in pre-training. *arXiv preprint arXiv:2408.10914*, 2024.

Balestriero, R. and Huang, H. For perception tasks: The cost of llm pretraining by next-token prediction outweigh its benefits. In *NeurIPS Workshop: Self-Supervised Learning-Theory and Practice*, 2024.

Baradad, M., Wulff, J., Wang, T., Isola, P., and Torralba, A. Learning to see by looking at noise. *arXiv preprint arXiv:2106.05963*, 2021.

Baradad, M., Chen, C.-F., Wulff, J., Wang, T., Feris, R., Torralba, A., and Isola, P. Procedural image programs for representation learning. *arXiv preprint arXiv:2211.16412*, 2022.

Chan, S., Santoro, A., Lampinen, A., Wang, J., Singh, A., Richemond, P., McClelland, J., and Hill, F. Data distributional properties drive emergent in-context learning in transformers. *Advances in neural information processing systems*, 35: 18878–18891, 2022.

Charton, F. and Kempe, J. Emergent properties with repeated examples. *arXiv preprint arXiv:2410.07041*, 2024.

Chiang, C.-H. and Lee, H.-y. On the transferability of pre-trained language models: A study from artificial datasets. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 10518–10525, 2022.

Eldan, R. and Li, Y. Tinystories: How small can language models be and still speak coherent english?, 2023.

Fan, S., Pagliardini, M., and Jaggi, M. Doge: Domain reweighting with generalization estimation. *arXiv preprint arXiv:2310.15393*, 2023.

Goodale, M., Mascarenhas, S., and Lakretz, Y. Meta-learning neural mechanisms rather than bayesian priors. *arXiv preprint arXiv:2503.16048*, 2025.

Grau-Moya, J., Genewein, T., Hutter, M., Orseau, L., Delétang, G., Catt, E., Ruoss, A., Wenliang, L. K., Mattern, C., Aitchison, M., et al. Learning universal predictors. *arXiv preprint arXiv:2401.14953*, 2024.

Han, S., Pari, J., Gershman, S. J., and Agrawal, P. General reasoning requires learning to reason from the get-go. *arXiv preprint arXiv:2502.19402*, 2025.

He, Z., Blackwood, G., Panda, R., McAuley, J., and Feris, R. Synthetic pre-training tasks for neural machine translation. In *Findings of the Association for Computational Linguistics*, pp. 8080–8098. Association for Computational Linguistics, 2023.

Hewitt, J. Initializing new word embeddings for pretrained language models, 2021. URL `https://www.cs.columbia.edu/~johnhew/vocab-expansion.html`.

Hu, M. Y., Petty, J., Shi, C., Merrill, W., and Linzen, T. Between circuits and chomsky: Pre-pretraining on formal languages imparts linguistic biases. *arXiv preprint arXiv:2502.19249*, 2025.

Huang, X. S., Perez, F., Ba, J., and Volkovs, M. Improving transformer optimization through better initialization. In *International Conference on Machine Learning*, pp. 4475–4483. PMLR, 2020.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Krishna, K., Garg, S., Bigham, J., and Lipton, Z. Downstream datasets make surprisingly good pretraining corpora. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*, pp. 12207–12222. Association for Computational Linguistics, 2023.

Lindemann, M., Koller, A., and Titov, I. Sip: Injecting a structural inductive bias into a seq2seq model by simulation. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, pp. 6570–6587. Association for Computational Linguistics, 2024.

Longpre, S., Yauney, G., Reif, E., Lee, K., Roberts, A., Zoph, B., Zhou, D., Wei, J., Robinson, K., Mimno, D., et al. A pretrainer's guide to training data: Measuring the effects of data age, domain coverage, quality, & toxicity. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 3245–3276, 2024.

McCoy, R. T. and Griffiths, T. L. Modeling rapid language learning by distilling bayesian priors into artificial neural networks. *arXiv preprint arXiv:2305.14701*, 2023.

Müller, S., Hollmann, N., Arango, S. P., Grabocka, J., and Hutter, F. Transformers can do bayesian inference. *arXiv preprint arXiv:2112.10510*, 2021.

Nakamura, R., Tadokoro, R., Yamada, R., Asano, Y. M., Laina, I., Rupprecht, C., Inoue, N., Yokota, R., and Kataoka, H. Scaling backwards: Minimal synthetic pre-training? *arXiv preprint arXiv:2408.00677*, 2024.

Papadimitriou, I. and Jurafsky, D. Injecting structural hints: Using language models to study inductive biases in language learning. *arXiv preprint arXiv:2304.13060*, 2023.

Petty, J., van Steenkiste, S., and Linzen, T. How does code pretraining affect language model task performance? *arXiv preprint arXiv:2409.04556*, 2024.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog*, 2019.

Ri, R. and Tsuruoka, Y. Pretraining with artificial language: Studying transferable knowledge in language models. *arXiv preprint arXiv:2203.10326*, 2022.

Ruis, L., Mozes, M., Bae, J., Kamalakara, S. R., Talupuru, D., Locatelli, A., Kirk, R., Rocktäschel, T., Grefenstette, E., and Bartolo, M. Procedural knowledge in pretraining drives reasoning in large language models. *arXiv preprint arXiv:2411.12580*, 2024.

Teney, D., Nicolicioiu, A. M., Hartmann, V., and Abbasnejad, E. Neural redshift: Random networks are not random functions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

Teney, D., Jiang, L., Gogianu, F., and Abbasnejad, E. Do we always need the simplicity bias? looking for optimal inductive biases in the wild. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025.

Trockman, A. and Kolter, J. Z. Mimetic initialization of self-attention layers. *arXiv preprint arXiv:2305.09828*, 2023.

Wang, Y., Ko, C.-Y., and Agrawal, P. Visual pre-training for navigation: What can we learn from noise? *arXiv preprint arXiv:2207.00052*, 2022.

Wang, Z., Wang, C., Dong, Z., and Ross, K. Pre-training with synthetic data helps offline reinforcement learning. *arXiv preprint arXiv:2310.00771*, 2023.

Wu, Y., Li, F., and Liang, P. S. Insights into pre-training via simpler synthetic tasks. *Advances in Neural Information Processing Systems*, 35:21844–21857, 2022.

Xie, S. M., Pham, H., Dong, X., Du, N., Liu, H., Lu, Y., Liang, P. S., Le, Q. V., Ma, T., and Yu, A. W. Doremi: Optimizing data mixtures speeds up language model pretraining. *Advances in Neural Information Processing Systems*, 36:69798–69818, 2023.

Xu, Z., Chen, Y., Vishniakov, K., Yin, Y., Shen, Z., Darrell, T., Liu, L., and Liu, Z. Initializing models with larger ones. *arXiv preprint arXiv:2311.18823*, 2023.

Ye, J., Liu, P., Sun, T., Zhan, J., Zhou, Y., and Qiu, X. Data mixing laws: Optimizing data mixtures by predicting language modeling performance. *arXiv preprint arXiv:2403.16952*, 2024.

Zhang, E., Lepori, M. A., and Pavlick, E. Instilling inductive biases with subnetworks. *arXiv preprint arXiv:2310.10899*, 2023.

Zhang, S., Patel, A., Rizvi, S. A., Liu, N., He, S., Karbasi, A., Zappala, E., and van Dijk, D. Intelligence at the edge of chaos. *arXiv preprint arXiv:2410.02536*, 2024.

Zhong, Z. and Andreas, J. Algorithmic capabilities of random transformers. In *Proceedings of the 38th Conference on Neural Information Processing Systems (NeurIPS)*, 2024.

A. Related Work

What is learned by pretraining language models. The quantity (Kaplan et al., 2020) and quality (Longpre et al., 2024) of pretraining data are empirically critical for the performance of large language models. But recent results also question the value of the data, showing that some benefits of pretraining are attributable to the optimization objective more than the actual data. Balestriero & Huang (2024) compared models trained for text classification from random initialization with fine-tuning from a pretrained checkpoint. They found that pretraining provides little benefit for tasks that do not involve text generation. Krishna et al. (2023) showed success in re-using the same data for pretraining and fine-tuning, showing also that the pretraining objective matters more than the data being used. The same conclusion follows from results of pretraining on synthetic data devoid of semantic meaning, e.g. for machine translation (He et al., 2023), computer vision (Baradad et al., 2021), visual navigation (Wang et al., 2022), and reinforcement learning (Baradad et al., 2022). This paper examines such purely synthetic pretraining to understand the exact capabilities that can be obtained from procedurally-generated data.

What matters in pretraining data. The selection of data to pretrain frontier models mostly relies on experimentation (Longpre et al., 2024). However, several key distributional and structural properties of the data have also been identified, such as data repetition to foster generalization (Charton & Kempe, 2024) and burstiness to enable in-context learning (Chan et al., 2022). Computer code is empirically very effective as pretraining data for LLMs, as it improves their abilities for compositional generalization and math-related tasks (Aryabumi et al., 2024; Petty et al., 2024). This presumably results from the abundant compositional and recursive patterns in computer code, but a better understanding of the mechanisms at play is lacking to reap the full benefits of structure in pretraining data. In this paper, we replicate the positive effects of structured pretraining data in controlled settings, and study how such data imparts useful inductive biases to the model.

Pretraining on procedural data. Most attempts to train language models with synthetic data follow a linguistic perspective, using formal languages to imitate properties of natural language (Chiang & Lee, 2022; Goodale et al., 2025; McCoy & Griffiths, 2023; Papadimitriou & Jurafsky, 2023; Ri & Tsuruoka, 2022). Recent work considers increasingly simpler forms of synthetic data such as input/outputs of simple algorithms (Lindemann et al., 2024; Wu et al., 2022). In these papers, specific forms of synthetic pretraining data prove helpful to subsequent fine-tuning on natural language tasks. Hu et al. (2025) provide strong empirical benefits, showing that data generated from formal languages is more valuable token-per-token than natural language for training a 1B-parameter language model. Zhang et al. (2024) pretrain on traces of cellular automata and show marginal but consistent improvements on simple reasoning tasks. Our study complements this line of work by examining more closely the pretrained models on diagnostic tasks, rather than evaluating their general handling of natural language. We identify specific capabilities imparted by specific types of procedural tasks, and locate useful structure in different parts of the architecture. We also investigate methods to combine the benefits from multiple complementary tasks.

Procedural data in vision and RL. Vision transformers (ViTs) have been trained on synthetic data of increasingly simple nature (Baradad et al., 2021). Nakamura et al. (2024) pretrained ViTs on a single fractal image with augmentations that remarkably match the performance of ImageNet-pretrained models after fine-tuning. This indicates that structural properties of the data matter more than its semantic contents. Similar results exist in reinforcement learning with models pretrained on data generated from random Markov chains (Wang et al., 2023) and noise-based images (Baradad et al., 2022).

Partial transfer from pretrained transformers. Zhang et al. (2023) and (Xu et al., 2023) showed that copying subsets of pretrained weights could transfer specific capabilities. Abnar et al. (2020) used knowledge distillation to transfer the inductive biases of one architecture into another. The "mimetic initialization" of self-attention (Trockman & Kolter, 2023) is a procedure handcrafted to imitate the locality bias of pretrained models. We also evaluate the partial transfer of pretrained weights, which reveals that different pretraining tasks create useful structure in different parts of the architecture.

Pretraining as an inductive bias. Pretraining transformers on synthetic data has been used to mimic the inductive biases of Bayesian inference (Müller et al., 2021) and Solomonoff Induction (Grau-Moya et al., 2024). Goodale et al. (2025) showed that well-chosen formal languages can teach complex mechanisms (e.g. counters) to a sequence model. Pretraining can generally be seen as a *soft* inductive bias for subsequent fine-tuning. But there is a large gap in our understanding of its effects compared to those of *hard* inductive biases of neural architectures (Teney et al., 2024; 2025). Han et al. (2025) argue that the difficulties of LLMs to reason robustly is due to their entangled representation of knowledge and reasoning. Much remains to be understood about how both are learned from the same data (Ruis et al., 2024). Our results suggest that procedural data could be one way to acquire reasoning mechanisms independently from specific pieces of knowledge.

B. Experimental Details

B.1. Procedural Data Generation

k-DYCK. We generate sequences consisting of correctly formed, nested parentheses using k distinct bracket pairs, where each bracket is treated as an individual token. Thus the vocabulary size is $2k$. For our experiments, we evaluate models with $k \in 4, 8, 16$, and fix all training sequences to a length of 128 tokens. Each sequence is constructed incrementally using a stack-based approach that enforces syntactic validity. At each step, the generator samples either an opening or a closing bracket based on a predefined probability ($p_{\text{open}} = 0.49$) following Papadimitriou & Jurafsky (2023). We ensure that every opened bracket is eventually closed. Specifically, if the number of remaining tokens equals the number of currently open brackets, the generator switches to exclusively emitting the appropriate closing brackets to guarantee a balanced sequence.

k-DYCK SHUFFLE. Shares the same vocabulary of opening and closing parentheses as k-DYCK, but relaxes the structural constraint of well-nestedness. Every open token has a corresponding close but they do not need to be properly nested. We use the implementation provided by Hu et al. (2025). Like k-DYCK we always use a sequence length of 128 tokens and evaluate $k \in 4, 8, 16$. Our opening token probability is ($p_{\text{open}} = 0.50$). As stated by Hu et al. (2025), the final sequence may be invalid due to truncation, but we also did not see any negative consequences of this.

STACK. Consists of sequences that simulate stack-based operations, where the first part of the input encodes a sequence of `push` and `pop` operations and the second part represents the resulting stack contents. Tokens are pushed onto a stack with a 75% probability in the first two-thrids of the sequence and popped with 75% probability in the later one third portion. Each push inserts a unique token, and pops remove the top of the stack. We ensure only a single occurance of a token can be present on the stack at any point in time. The input sequence is followed by a separator token and the remaining stack contents (in top-to-bottom order). The model is trained to autoregressively predict the stack tokens after the separator. We train using curriculum learning: starting with input sequences of length 4, we increase the length by 2 once the model achieves 99% accuracy on the current sequence length. The curriculum progresses until reaching a maximum sequence length of 20. This gradually exposes the model to increasingly complex stack manipulations, allowing it to build algorithmic competence over time. The vocabulary size is set to 103, comprising 100 pushable tokens, a dedicated `pop` token, a separator token, and a padding token.

IDENTITY. The setup mirrors that of the STACK, also employing curriculum learning. Each input sequence consists of randomly sampled tokens, which are concatenated with a separator token. The target output is an exact copy of the input sequence. The model is trained to autoregressively reproduce the input tokens following the separator, requiring it to replicate the input structure. The vocabulary is set to 102 tokens, comprising 100 valid input elements, along with dedicated tokens for padding and separation.

SET. Also uses curriculum learning to progressively increase input length. Each input sequence consists of randomly sampled tokens and is followed by a separator token. The target output is a de-duplicated version of the input sequence, preserving the original order of first occurrence for each token. This requires the model to remember which elements have already appeared while autoregressively generating the output. The vocabulary size is 102, with 100 tokens representing valid input elements and two special tokens for the separator and padding.

ECA RULE 110. We adopt the training setup and codebase released by Zhang et al. (2024), where data is procedurally generated from Elementary Cellular Automata (ECA) using Rule 110, a Class IV rule known for its complex, Turing-complete behavior. To enable next-token prediction over binary state sequences, their approach modifies the standard GPT-2 architecture by replacing the token embedding layer with a linear projection that maps binary vectors directly into the model's embedding space. Similarly, the output softmax is replaced by a linear projection back to binary space. This design allows the model to process raw binary sequences and remain deterministic, aligning with the deterministic nature of ECA dynamics. For transfer into our downstream tasks, we compute and extract the average of the learned input embeddings over the ECA pretraining data. These averaged embeddings are then used to initialise the embedding layers of our target transformer models, enabling effective transfer of structure without relying on a pretrained embedding layer.

B.2. Procedural Pretraining

We list in Table 2 the hyperparameters for each type of procedural data other than ECA.

Task	SEQ. LENGTH	BATCH SIZE	LEARNING RATE	WARMUP STEPS	VOCAB. SIZE	EARLY STOPPING	MAX STEPS
k-DYCK	128	256	1×10^{-4}	100,000	$2 \times k$	—	1,000,000
K DYCK SHUFFLE	128	256	1×10^{-4}	100,000	$2 \times k$	—	1,000,000
STACK	4–20	256	5×10^{-4}	1,000	103	100 VALIDATION CHECKS	1,000,000
SET	2–20	256	5×10^{-4}	1,000	102	100 VALIDATION CHECKS	1,000,000
IDENTITY	4–20	256	5×10^{-4}	1,000	102	100 VALIDATION CHECKS	1,000,000

Table 2: Pretraining hyperparameters for each procedural task. All use AdamW and a weight decay of 0.01.

ECA RULE 110. We adopt the training configuration from Zhang et al. (2024). Models are pretrained for next-token prediction using data generated from ECA rule 110. In each epoch, a fresh dataset is generated from a new random initial state. This setup effectively simulates infinite data. Training proceeds for up to 10,000 epochs with early stopping based on validation loss. We use a batch size of 64 (60 time steps, 100 spatial dimensions), the Adam optimiser with a learning rate of 2×10^{-6}, weight decay of 0.01, and gradient clipping (norm ≤ 1.0). A learning rate warm-up over the first 10% of training steps is followed by cosine annealing.

B.3. Description of Downstream Diagnostic Tasks

HAYSTACK. We adopt the task as implemented by Zhong & Andreas (2024) that is publicly available. This task assesses a model's ability to perform retrieval over long sequences. Each input consists of a series of key-value pairs in the format $[m_1, c_1, m_2, c_2, \ldots, m_k, c_k, m_u]$, where each m_i is a distinct marker, and each c_i is the corresponding value. The sequence concludes with a query marker m_u, and the model is required to locate its earlier occurrence and return the associated value c_u. For all experiments we set $k = 30$. Accuracy is computed on c_u.

ADDITION. This task requires the model to learn the structure of arithmetic addition presented in forward (non-reversed) notation. This is seen as a harder task than the reversed addition for transformers, as the least significant digits, which are critical to determining carry operations, appear later in the sequence. This forces the model propagate carry information backward through the input, which is misaligned with the auto retrogressive training procedure. Each input sequence takes the form `a+b=` where a and b are randomly sampled integers with a digit length n and the output is their sum. Inputs and outputs are tokenised at the digit level, with symbol tokens (+, =) assigned unique indices. Models are trained to predict only the result digits, with cross-entropy loss computed exclusively on those output positions. For all experiments we set n = 5. Accuracy is computed at the token level on the result digits.

REVERSED ADDITION. We again use the implementation by Zhong & Andreas (2024). This task evaluates a model's ability to perform multi-step arithmetic by adding two length n integers represented as sequences of digits. To simplify the positional dependencies, both the inputs and the output are reversed: for instance, the sum $ab + cd = efg$ is encoded with input `b a d c` and output `g f e`. The model must predict the digit-wise sum in left-to-right order, reflecting carry propagation across digit positions. We set $n = 10$ and accuracy is computed at the token level.

MULTIPLICATION. We evaluate the model's ability to perform multi-digit multiplication. Each input sequence represents an equation of the form $a \times b =$, where a and b are randomly sampled n-digit integers. The model is required to output the digits of their product. Inputs and outputs are tokenised at the digit level, with the multiplication operator × and the equals sign = assigned special token IDs. For all experiments, we set $n = 5$. Loss and accuracy are computed only over the output portion of the sequence corresponding to the product digits.

SORTING. This task evaluates a model's ability to perform algorithmic reasoning by sorting a sequence of integers. The input consists of a list of n integers sampled uniformly from the range $[0, P - 1]$, where P is the size of the symbol vocabulary. We set $n = 10$ and $P = 100$. The model receives the input sequence followed by a separator token and is trained to output the sorted version of the input immediately after the separator. For example, given an input sequence `6 3 5` and separator `|`, the expected output is `3 5 6`. The model is trained autoregressivley and is evaluated only on the tokens following the separator, where we calculate accuracy at the token level.

LANGUAGE MODELLING. This task uses the *TinyStories* dataset (Eldan & Li, 2023), a collection of short, synthetically generated English-language narratives. Each story consists of simple sentences intended for early readers. We frame this as a last-token prediction task. The model is given the first 63 tokens of a 64-token sequence sampled from a passage and is

trained to predict the 64$^{\text{th}}$ token. To simplify the vocabulary, we restrict the model to the top 2,000 most frequent tokens from the dataset. Accuracy is computed based on whether the predicted token matches the correct token at position 64, testing the model's ability to use contextual information for natural language completion. We adapt code from the `Pluto` repository [6] for this task.

B.4. Downstream Training

HAYSTACK, FORWARD ADDITION, REVERSED ADDITION, and SORTING. We trained models for 10^4 steps with a batch size of 1,000. The training data is generated dynamically. We used the AdamW optimizer with a learning rate of 10^{-3} and weight decay of 10^{-3}. We always use an architecture consisting of 2 layers, 4 attention heads, and 16-dimensional embeddings. We report mean and standard deviation over 10 seeds. The main body of the paper reports only mean accuracy for clarity; the full statistics including standard deviations are presented in Appendix C.

MULTIPLICATION. These experiments employed a larger model with 4 layers, 8 attention heads, and 512-dimensional embeddings. Thus, we use a smaller training batch size (64 vs. 1,000), resulting in approximately 156k update steps compared to 10k steps for the afforementioned reasoning tasks, despite using the same number of training examples. We optimize with AdamW using a learning rate of 10^{-3}, weight decay of 10^{-3}, and 500 warmup steps. We run this over 3 seeds, and report standard deviations in Appendix C.

LANGUAGE MODELLING. We used a larger architecture with 2 layers, 4 attention heads, and 64-dimensional embeddings. Models are trained for 1 epoch on 1.2 million *TinyStories* sequences of length 64 using a batch size of 64 and vocabulary size of 2,000. The learning rate is set to 2×10^{-3} with 10% linear warmup steps and cosine decay. We also run this over 10 seeds, and report standard deviations in Appendix C.

[6] `https://github.com/tanaydesai/pluto`

C. Additional Results

Pretraining task	HAYSTACK	ADDITION	REVERSED ADDITION	MULTIPLICATION	SORTING	LANGUAGE MODELLING
RAND INIT.	11.3 ± 0.4	59.1 ± 7.0	76.4 ± 23.2	42.7 ± 5.3	82.7 ± 11.6	40.2 ± 0.2
4-DYCK	98.3 ± 1.1	52.7 ± 0.3	35.7 ± 2.5	46.7 ± 4.6	56.3 ± 19.2	41.6 ± 0.2
8-DYCK	93.6 ± 1.3	53.4 ± 0.3	48.9 ± 4.9	44.5 ± 0.9	98.7 ± 0.3	41.2 ± 0.2
16-DYCK	96.9 ± 1.0	87.8 ± 4.2	83.5 ± 0.6	39.4 ± 3.3	95.5 ± 1.0	41.0 ± 0.5
4-DYCK SHUFFLE	7.3 ± 0.6	54.5 ± 0.2	87.8 ± 12.9	41.8 ± 3.7	61.0 ± 1.4	41.0 ± 0.3
8-DYCK SHUFFLE	9.6 ± 0.3	67.7 ± 0.8	90.1 ± 5.9	37.4 ± 0.1	84.1 ± 5.7	41.2 ± 0.3
16-DYCK SHUFFLE	18.6 ± 26.3	70.8 ± 5.5	87.0 ± 12.8	44.0 ± 0.1	71.1 ± 5.4	41.2 ± 0.3
STACK	55.2 ± 39.3	62.3 ± 5.3	34.9 ± 0.2	46.6 ± 2.0	21.3 ± 0.6	38.4 ± 0.3
IDENTITY	18.8 ± 14.3	54.7 ± 0.2	42.7 ± 0.9	46.6 ± 2.7	19.9 ± 0.5	37.9 ± 0.2
SET	18.9 ± 26.6	53.4 ± 0.1	44.6 ± 5.1	43.5 ± 8.4	93.5 ± 1.6	40.05 ± 0.5
ECA	10.5 ± 0.5	69.6 ± 7.9	91.1 ± 16.1	—	76.9 ± 1.4	—

Table 3: Full results across all pretraining tasks and downstream tasks. Each cell reports the mean accuracy $\pm$ standard deviation over 10 random seeds, except for MULTIPLICATION and LANGUAGE MODELLING, which is over 3 seeds. A subset of these results is visualised in Figure 2.

Pretraining task	FULL TRANSFER	MLP ONLY	ATTENTION ONLY
4-DYCK	98.3 ± 1.1	8.7 ± 0.5	11.6 ± 0.5
16-DYCK SHUFFLE	18.6 ± 26.3	8.9 ± 0.9	16.5 ± 10.6
STACK	55.2 ± 39.3	7.1 ± 0.6	98.9 ± 0.8
IDENTITY	18.8 ± 14.3	7.0 ± 0.9	99.0 ± 1.7
SET	18.9 ± 26.6	8.3 ± 0.7	88.9 ± 27.1
ECA	10.5 ± 0.5	8.7 ± 1.0	11.6 ± 1.0

Table 4: HAYSTACK task accuracy (mean $\pm$ standard deviation over 10 seeds) for models initialised with weights from different pretraining tasks. We report results for full model transfer, MLP-only transfer, and attention-only transfer.

Pretraining task	FULL TRANSFER	MLP ONLY	ATTENTION ONLY
16-DYCK	87.8 ± 4.2	60.0 ± 6.6	59.2 ± 10.4
16-DYCK SHUFFLE	70.8 ± 5.5	61.7 ± 6.9	55.3 ± 4.9
STACK	62.3 ± 5.3	61.1 ± 9.4	56.2 ± 5.0
IDENTITY	54.7 ± 0.2	58.3 ± 7.2	69.7 ± 13.1
SET	53.4 ± 0.1	59.6 ± 6.4	81.1 ± 12.2
ECA	69.6 ± 7.9	63.1 ± 14.4	65.8 ± 12.8

Table 5: ADDITION task accuracy (mean $\pm$ standard deviation over 10 seeds) for models initialised with weights from different pretraining tasks. We report results for full model transfer, MLP-only transfer, and attention-only transfer.

Pretraining task	FULL TRANSFER	MLP ONLY	ATTENTION ONLY
16-DYCK	83.5 ± 0.6	64.0 ± 26.4	49.1 ± 20.3
8-DYCK SHUFFLE	90.1 ± 5.9	65.8 ± 24.8	63.3 ± 18.1
STACK	34.9 ± 0.2	74.4 ± 24.7	42.1 ± 8.1
IDENTITY	42.7 ± 0.9	71.7 ± 29.2	45.2 ± 3.7
SET	44.6 ± 5.1	71.2 ± 23.7	54.4 ± 10.4
ECA	91.1 ± 16.1	70.5 ± 31.6	75.5 ± 27.2

Table 6: REVERSED ADDITION task accuracy (mean $\pm$ standard deviation over 10 seeds) for models initialised with weights from different pretraining tasks. We report results for full model transfer, MLP-only transfer, and attention-only transfer.

Pretraining task	FULL TRANSFER	MLP ONLY	ATTENTION ONLY
8-DYCK	98.7±0.3	72.8±3.1	71.4±5.7
8-DYCK SHUFFLE	84.1±5.7	78.2±8.6	62.9±6.7
STACK	21.3±0.6	71.0±2.2	77.5±12.2
IDENTITY	19.9±0.5	74.5±8.1	91.3±10.1
SET	93.5±1.6	73.5±1.5	98.1±2.8
ECA	76.9 ± 1.4	77.1±8.1	73.9±3.2

Table 7: SORTING task accuracy (mean ± standard deviation over 10 seeds) for models initialised with weights from different pretraining tasks. We report results for full model transfer, MLP-only transfer, and attention-only transfer.

Perturbation	HAYSTACK	ADDITION	REVERSED ADDITION	SORTING	LANGUAGE MODELLING
Pretrained	98.9 ± 0.8	87.8 ± 4.2	90.1 ± 5.9	98.7 ± 0.3	41.6 ± 0.2
Shuffled	17.2 ± 12.7	61.0 ± 9.1	82.9 ± 23.5	94.2 ± 4.2	41.1 ± 0.4
0.01 noise	98.6 ± 1.7	77.6 ± 20.1	74.0 ± 21.0	96.0 ± 7.6	41.5 ± 0.5
0.05 noise	50.8 ± 30.5	62.1 ± 13.3	91.0 ± 15.7	71.9 ± 26.1	41.1 ± 0.3
0.10 noise	32.9 ± 6.1	56.4 ± 7.4	83.6 ± 21.5	37.9 ± 5.8	40.2 ± 0.1
Random init	11.3 ± 0.4	59.1 ± 7.0	76.4 ± 23.2	82.7 ± 11.6	40.2 ± 0.2

Table 8: Mean accuracy (± standard deviation over 10 seeds) across five downstream tasks under different perturbation conditions. Pretrained models were selected based on best individual performance per task: STACK (attention-only) for HAYSTACK, 16-DYCK for ADDITION, 8-DYCK SHUFFLE for REVERSED ADDITION, 8-DYCK for SORTING, and 4-DYCK for LANGUAGE MODELLING.