LASTING EFFECTS OF ABSTRACT PRETRAINING BEYOND PERPLEXITY

Anonymous authors
Paper under double-blind review

ABSTRACT

Language models are typically pretrained from random initialization. Recent work challenges this convention, showing that a brief warm-up on abstract, algorithmically generated data can provide a better starting point for subsequent learning of natural language. In this paper, we show that in small language models, such a warm-up improves specific capabilities that are not reflected in language-modeling perplexity. Our warm-up uses an abstract stack-manipulation task that requires compositional and state-tracking capabilities. Allocating as little as 1% of pretraining tokens to this data improves multi-hop question answering by up to 3.9 F_1 points on MUSIQUE, with additional gains on HOTPOTQA and 2WIKIMULTIHOPQA despite comparable language-modeling perplexity. Controlled experiments further show that the warm-up substantially accelerates the acquisition of deeper reasoning chains. We also explore what drives this transfer. First, the precise structure of the data matters: replacing the stack task with a queue manipulation fails to produce the same gains. Second, the gains are specific: performance improves on sequential reasoning chains, with no consistent benefit on tasks that combine or compare independent facts. Third, timing matters: mixing abstract data with natural language is far less effective than an initial dedicated phase, and exposure *after* pretraining completely removes the benefits. The early advantage also persists through billions of subsequent language tokens. These results show that early training on abstract data can reliably shape the capabilities that language models later acquire.

1 INTRODUCTION

Recent studies show that brief training on abstract, algorithmically generated data can improve how transformers subsequently learn language, code, mathematics, and images (Jiang et al., 2026; Lee et al., 2026; Papadimitriou & Jurafsky, 2023; Shinnick et al., 2026). This approach, known as *procedural pretraining*, exposes models to data generated by simple algorithms such as formal languages or cellular automata, prior to standard pretraining (also called *pre-pretraining* by Hu et al. 2025). The benefits include faster convergence and improved downstream capabilities (Jiang et al., 2026; Lee et al., 2026). Yet it remains unclear which properties of the abstract data matter, which specific capabilities improve, and what drives these benefits.

This paper studies how abstract pretraining shapes the subsequent acquisition of capabilities in small language models, in particular for multi-hop reasoning. We show that brief initial training on an abstract memory-manipulation task improves multi-hop question answering and accelerates the acquisition of reasoning skills over long chains. Importantly, these benefits are not always reflected in language-modeling perplexity. This suggests that abstract pretraining shapes the inductive biases for generalization, rather than simply making the language-modeling optimization easier. It also implies that selecting abstract data based on perplexity alone can miss important differences in the capabilities they enable.

Overview of our study. We conduct controlled experiments to identify which capabilities benefit from abstract training, which properties of the abstract data drive the transfer, and how the timing of exposure affects it. We evaluate multi-hop question answering on MUSIQUE, HOTPOTQA, and 2WIKIMULTIHOPQA (Trivedi et al., 2022; Yang et al., 2018; Ho et al., 2020) and use a controlled synthetic task (DEPO, (Allen-Zhu, 2026)) to measure the acquisition of increasingly long reasoning

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

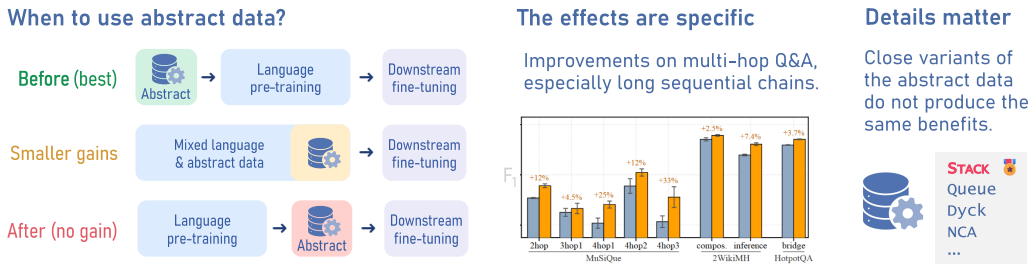


Figure 1: We identify three practical principles for using abstract data to pretrain language models. **(Left)** It is most effective in an initial dedicated phase. **(Middle)** The benefits are not always apparent in language-modeling perplexity, but they show up more clearly in evaluations of specific capabilities such as multi-hop reasoning. **(Right)** The choice of abstract data matters: the benefits depend on the model learning specific structures rather than a generic headstart on the optimization.

chains. We also vary the structure of the abstract task and its timing relative to language pretraining. This allows us to isolate the specific capabilities that transfer and determine whether the same abstract data must be presented before, during, or after language pretraining.

We focus on a specific form of abstract data, referred to as STACK, that simulates sequences of push and pop operations on a memory. Next-token prediction on this data requires tracking the state of the memory through nested, compositional sequences. Prior work found it particularly effective for procedural pretraining of LLMs for code and natural language (Jiang et al., 2026). Its simple structure also allows us to systematically manipulate the abstract computation and test which aspects of the learned inductive bias transfer to downstream capabilities.

Our contributions and main findings are summarized as follows.

1. **Identifying capabilities that improve with abstract pretraining.** We find clear improvements in multi-hop reasoning on Q&A benchmarks (MUSIQUE, HOTPOTQA, and 2WIKIMULTIHOPQA), specifically on reasoning chains in which later steps depend on earlier ones, rather than on tasks that merge or compare independent facts. Using DEPO, we further find faster acquisition of longer reasoning chains (Section 4). These effects are not consistently coupled with improvements in language-modeling perplexity.
2. **Identifying properties of the abstract data that determine transfer.** We construct variants of the STACK task that fail to reproduce its benefits (Section 5). In particular, a similar queue-manipulation task achieves similar perplexity but does not produce the same downstream improvements. Thus, the choice of abstract data can affect generalization independently of its effect on language modeling.
3. **Identifying when abstract training must occur.** We find that early exposure is critical: abstract training throughout or after language pretraining does not produce the same benefits (Section 6). The persistence of the early advantage through billions of subsequent language tokens shows that abstract training can meaningfully influence the whole training trajectory.

Our results identify three factors that affect whether abstract data produces measurable benefits: the capability being evaluated, the structure of the abstract data, and its position in the training pipeline. Our experiments are conducted at a small scale (SmolLM-135M and 360M) but they complement prior studies showing that the benefits of abstract training extend to larger models (1.3B parameters in Jiang et al. 2026) and longer training ($\sim$ 400 language tokens/parameter in Cheng et al. 2026).

2 RELATED WORK

Pretraining on abstract data. Data from formal languages is a useful tool in linguistics and NLP for controlled studies of language acquisition (Chiang & Lee, 2022; Goodale et al., 2025; McCoy & Griffiths, 2023; Papadimitriou & Jurafsky, 2023; Ri & Tsuruoka, 2022; Hu et al., 2025). Abstract data was proposed recently as a practical means to improve the training of large language models (LLMs) (Lindemann et al., 2024; Wu et al., 2022; 2021; Zhang et al., 2024; Bloem, 2025). This data is used in a short initial training phase, and the benefits include faster subsequent training (Jiang

et al., 2026; Lee et al., 2026), robustness to noisy data (Guo et al., 2026), and improvements in downstream capabilities (Cheng et al., 2026; Veitsman et al., 2026). Improvements on benchmarks are not necessarily tied to better pretraining perplexity, which is well known for LLMs in general (Liu et al., 2023; Veličković et al., 2026). We explore this in more detail, identifying properties of the data responsible for the improvements and the effect on specific capabilities such as multi-hop reasoning.

Training order in language-model pretraining. Models learn different skills and properties of the data in a predictable order Achille et al. (2019); Belrose et al. (2024); Michaelov et al. (2026) which can be controlled with the order of the data (Bengio et al., 2009). Reasoning data during pretraining provides benefits that fine-tuning does not fully reproduce (Akter et al., 2026). Most closely related to our work, Cheng et al. (2026) show that initial training on formal derivations affects subsequent skill acquisition and model representations. However, this and other studies with abstract data do not establish whether a dedicated initial training phase is necessary. We address this gap by directly comparing exposure to abstract data before, during, and after language pretraining, and testing whether the effects persist through subsequent training.

Multi-hop reasoning is the skill required to combine multiple pieces of evidence through dependent steps. It is challenging for language models and dedicated question-answering benchmarks are designed such that the answers cannot be determined by retrieving individual facts (e.g. HotpotQA, 2WikiMultiHopQA, MuSiQue; Yang et al. 2018; Ho et al. 2020; Trivedi et al. 2022). Existing solutions include context repetition (Yu et al., 2025), interleaved retrieval and reasoning (Trivedi et al., 2023), and RL fine-tuning (Kabra et al., 2026). In comparison, we keep the architecture and fine-tuning protocol fixed, and show that multi-hop reasoning can be improved by shaping the model's inductive biases through an initial training on generic abstract data.

3 METHODS

Our setup follows prior work on abstract data (see e.g. Jiang et al. 2026; Lee et al. 2026). We train a language model first on a small amount of abstract data, then proceed with standard phases such as language pretraining and supervised fine-tuning (SFT) on a target task.

		Baseline		
(1) **Training on abstract data**	→	(2) **Standard language pretraining**	→	(3) **SFT on target task**
STACK, QUEUE, NCA, etc.		FINEWEB-EDU or TINYSTORIES		HOTPOTQA, DEPO, etc.
Typically ∼80M tokens		E.g. 7.2B tokens (20 tokens/param.) for SmolLM-360M		Budget varies across datasets

The model is evaluated for language-modeling perplexity after (2) and for downstream performance on the target task after (3). The main baseline is a model that undergoes only phases (2) and (3). The first phase on abstract data can therefore be seen as an alternative to standard random initialization that instills better inductive biases in the model. We match the token budget of phases (2,3) across experiments since the additional data in the abstract phase only adds a very small amount of tokens (∼1% of the pretraining budget, see Table 2 in the appendix).

We additionally want to evaluate whether the abstract data could be used at other points in this pipeline. We therefore evaluate alternative settings (see Figure 1) where the abstract data is either interspersed with the language data, or used after language (swapping (1) and (2) above).

3.1 MODELS AND LANGUAGE PRETRAINING

Our main experiments use the SmolLM-135M and SmolLM-360M architectures (Allal et al., 2024) and FINEWEB-EDU (Penedo et al., 2024) as the source of language data, with a budget of 20 language tokens/parameter (Hoffmann et al., 2022). Our abstract data (described below) uses the same vocabulary size as the SmolLM tokenizer, the token embeddings and output head are thus trained and transferred across phases. A few of our experiments also use TINYSTORIES (Eldan & Li, 2023) as an alternative source of language data.

We use the training recipe and hyperparameters from the SmolLM codebase, except that we adapt it for the modern Muon optimizer because it performs significantly better as a baseline. We re-tune the learning rates and weight decay by grid search on the SmolLM-135M baseline with full Chinchilla-optimal runs (2.69B tokens; see details in Appendices A.2 and B.1) and use these hyperparameters throughout our experiments. We do not re-tune them when using abstract data, so further gains might be possible. Our results are typically reported over three to nine seeds (details in Appendix B.3).

3.2 ABSTRACT DATA

For abstract training, we optimize the model for next-token prediction on one of these types of data.

Memory manipulation (STACK). This data is generated as random sequences of push and pop operations on a simulated stack memory (last-in, first-out). Most operations are arbitrary (at any point, valid continuations are to either push any new token, or pop one if the stack is not empty). But each sequence ends with the actual contents of the memory according to the preceding operations (see examples in Table 1 and Appendix A.1). Successful prediction therefore requires tracking the state of the memory over the whole sequence. This data proved particularly effective in prior work (Jiang et al., 2026) and in our initial experiments. It is therefore the focus of our experiments.

Variants of STACK. In the first variant (QUEUE), we replace the stack with a queue (first-in, first-out). In the second variant (STACKRAND), we randomly permute the final contents of the stack. These allow evaluating minor variations in the underlying structure of the data while keeping the surface statistics similar.

Alternative tasks. We also evaluate others forms of abstract data proposed in prior work, generated with simple algorithms like SORTING and SET (Jiang et al., 2026), neural cellular automata (NCA, Lee et al. 2026), and formal languages (Hu et al., 2025). See details in Appendix A.1– A.3.

Table 1: Examples of abstract data. Boxes represent tokens. The loss is computed on colored ones.

Type of abstract data	Example sequences
STACK	`push(A)` `push(B)` `pop()` `push(C)` `<SEP>` `A` `C` `<EOS>`
QUEUE	`push(A)` `push(B)` `pop()` `push(C)` `<SEP>` `B` `C` `<EOS>`
STACKRAND	`push(A)` `push(B)` `pop()` `push(C)` `<SEP>` `C` `A` `<EOS>`
SORT	`Z` `C` `B` `T` `A` `<SEP>` `A` `B` `C` `T` `Z` `<EOS>`

3.3 TARGET DATA AND EVALUATION

We use three question-answering benchmarks designed to evaluate multi-hop reasoning:

- **MUSIQUE** (Trivedi et al., 2022) evaluates multi-hop reasoning through questions composed from dependent single-queries, with 2–4 reasoning steps that prevent the use of disconnected shortcuts. We use the answerable split with gold supporting paragraphs.
- **HOTPOTQA** (Yang et al., 2018) evaluates multi-hop reasoning over multiple supporting documents, including bridge and factoid-comparison questions.
- **2WIKIMULTIHOPQA** (Ho et al., 2020) evaluates multi-hop reasoning using compositional and comparison questions derived from Wikidata relations.

We follow the protocol from Trivedi et al. (2022) and fine-tune the model for one epoch on 20k training questions (details in Appendix B.3). We report the mean token-level F_1 in our main results. Other metrics are reported in Appendix C. We also use a synthetic task for controlled evaluations:

- **DEPO** (Allen-Zhu, 2026) evaluates multi-hop reasoning of controlled complexity. Each sequence describes a directed cycle of nodes with the list of its connected pairs in random order. This is followed by a "query" that specifies a node and number of hops k. The answer is then the node reached k steps down the chain. The model must therefore dereference k pairs from the context in the right order to identify the correct answer. We use the code from Allen-Zhu (2026) to generate data for fine-tuning and evaluation with a difficulty adjusted to be barely solvable by our baseline models, namely 10 nodes per cycle and $k \in \{1,2,3,4\}$ (details in Appendix B.4).

4 SPECIFIC IMPROVEMENTS BEYOND LANGUAGE MODELING

In this section, we evaluate the effect of initial training with STACK on language modeling, on QA benchmarks with aggregate and question-type performance, and finally on the synthetic DEPO task.

4.1 IMPROVEMENTS IN MULTI-HOP QUESTION ANSWERING

We evaluate SmolLM-135M and SmolLM-360M models trained on STACK then on FINEWEB-EDU against baselines trained solely on FINEWEB-EDU. These models are then fine-tuned on either

MUSIQUE, HOTPOTQA, or 2WIKIMULTIHOPQA. We run this pipeline for 3 pre-training seeds and 3 fine-tuning seeds, i.e. evaluating 9 models in total.

Figure 2 shows that our models improve the mean F_1 on MUSIQUE and HOTPOTQA at both sizes, and on 2WIKIMULTIHOPQA at 135M. The result on 2WIKIMULTIHOPQA at 360M is largely unchanged. The largest gain is 3.9 F_1 points on MUSIQUE with SmolLM-135M. These improvements occur despite comparable language-modeling performance at 135M and even slightly worse at 360M (Figure 2, bottom). The initial training on abstract data therefore shapes the capabilities in the model in a way that only becomes apparent after fine-tuning on particular tasks.

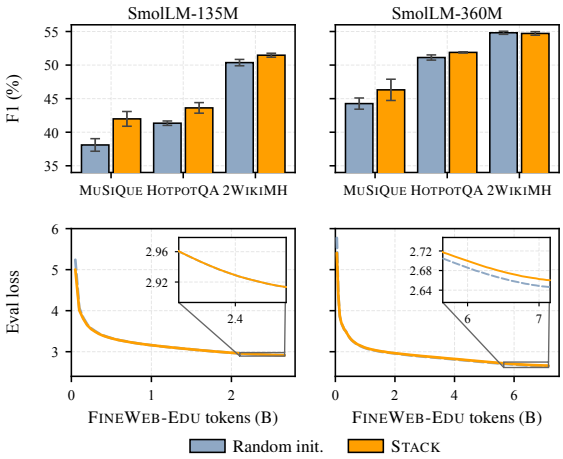


Figure 2: **STACK improves multi-hop QA without improving language modeling.** We compare language-only pretraining with STACK followed by language pretraining at two model sizes (SmolLM-135M and SmolLM-360M). **(Top)** Token-level F_1 after benchmark-specific fine-tuning. **(Bottom)** Validation loss during the pretraining phase on FINEWEB-EDU.

4.2 THE GAINS CONCENTRATE ON SEQUENTIAL COMPOSITIONS

We now run a finer-grained evaluation using the annotations of question types in the benchmarks. These categories specify the type of reasoning structure necessary to solve each question. The **chain**-dominated categories involve passing intermediate answers across sequential dependent steps, while the **merge**- and **comparison**-based categories involve combining or comparing information from separate branches. See Appendix B.3 for details on the grouping of question types.

Figure 3 reports the mean F_1 of our models for each question type. The STACK-trained models obtain consistently better performance on chain-dominated questions, for all relevant question types in the three benchmarks (left box). In contrast, there is no clear effect on the other question types in the merge- and comparison-based categories (right box), with slight degradation on one question type in MUSIQUE (close however to the inter-seed variability of the evaluation). These results indicate that the improvements obtained with the STACK data are not generic. They instead target specific capabilities in particular for sequential composition.

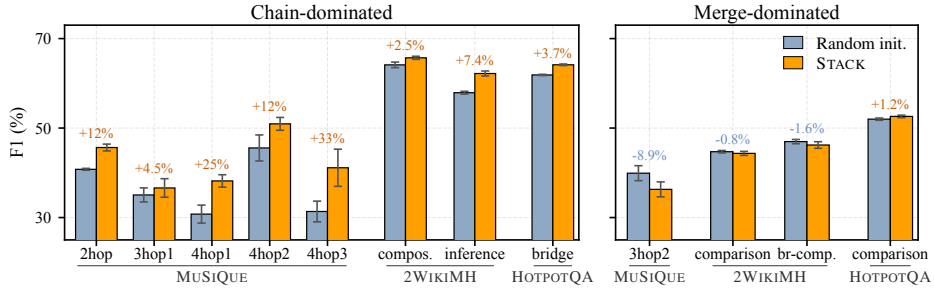


Figure 3: **Gains on QA benchmarks concentrate on sequentially composition.** Mean token-level F_1 by question type. Models trained on STACK perform consistently better in every chain-dominated category, not in merge- and comparison-based categories. Annotations indicate relative differences with the baseline trained only on language from standard random initialization (no abstract data).

4.3 Faster Learning on a Controlled Multi-Hop Task

We now use the synthetic DEPO task to precisely measure how well and how fast the model learns multi-hop operations of specific complexity. We use here a simple GPT-2-style model with 4 layers (details in Appendix B.2). We compare a baseline pretrained on language (FINEWEB or TINYSTORIES) with ours trained first on STACK before language. They are then fine-tuned for DEPO on data with 1-to-4-hop queries, and evaluated on test data of each complexity throughout training.

Figure 4 shows that models exposed to the STACK data learn DEPO faster at every complexity level. By the end of the fine-tuning, our models reach near-perfect accuracy for all hop counts, while the language-only models still do not handle the four-hop queries correctly. We show in the appendix (Figure 13) that this striking improvement on DEPO occurs even though the language modeling validation loss is similar or even a bit worse with our model on both language corpora.

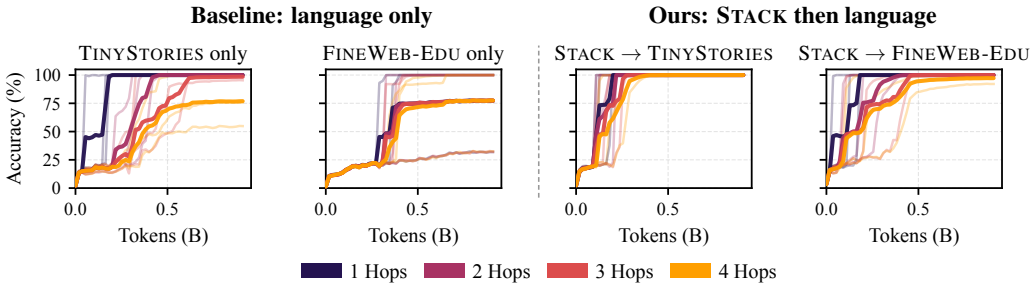


Figure 4: **(Left)** On the synthetic DEPO task, the baseline models with standard pretraining on language struggle to perform multi-hop operations. **(Right)** In comparison, our models exposed to the STACK data learn faster and reach near-perfect accuracy across all tested hop counts.

Takeaway. Initial training on STACK improves multi-hop QA, with gains concentrated on sequential composition and long chains, even when the language modeling capabilities of the models appear similar or even slightly worse than the baselines.

5 Transfer Depends on Task Structure and Precise Weights

Section 4 shows that initial exposure to abstract data can improve performance after language and task-specific training. We now try to identify the conditions necessary for this positive transfer.

5.1 Transfer Varies Across Abstract Tasks

We compare STACK with other types of abstract data from prior work (details in Appendix A.1). We use a minimal setup with direct transfer to DEPO and no intervening language pretraining (details in Appendix A.3). We observe substantial differences in Figure 5. STACK enables the fastest learning to near-perfect accuracy. Other options converge more slowly or fail to solve the 3- and 4-hop test cases. This highlights the specificity of the effects, but does not mean that STACK is universally better: prior work already showed that different types of abstract data can induce different capabilities (Shinnick et al., 2025). See Figure 14 in the appendix for a full comparison of 12 types of data.

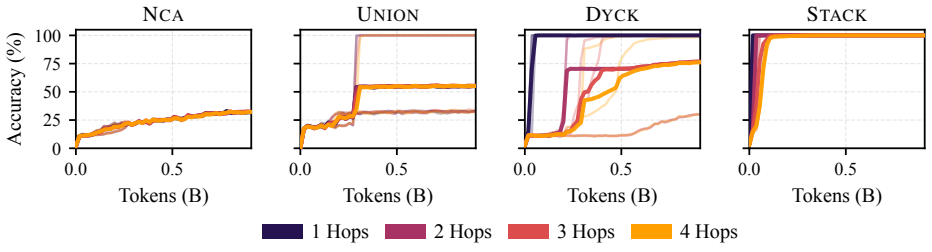


Figure 5: **Different types of abstract data are not equally effective.** The test accuracy on DEPO throughout training is best for STACK. See Figure 14 for additional options from prior work.

We now evaluate the variants of STACK introduced in Section 3.2: QUEUE and STACKRAND. We evaluate them on natural-language multi-hop QA after language pretraining, and on DEPO through direct transfer.

Figure 7 shows that QUEUE performs poorly in all situations. STACKRAND retains substantial benefits of STACK: their performance is close on HOTPOTQA and 2WIKIMULTIHOPQA, but STACKRAND is a bit worse on MUSIQUE and learns more slowly on DEPO. These differences occur again despite similar validation loss during language pretraining (Figure 6).

The precise structure of the abstract task therefore affects downstream transfer in ways that the language-modeling loss does not capture.

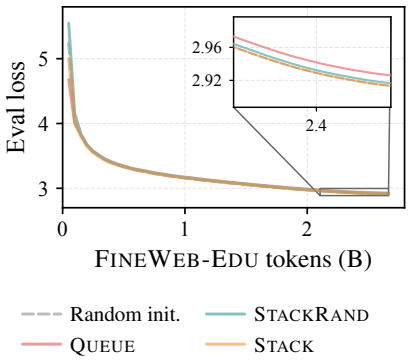


Figure 6: Similar loss during FINEWEB-EDU pretraining without and with prior exposure to variants of STACK.

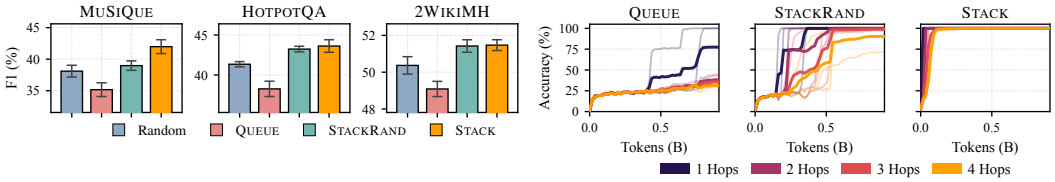


Figure 7: **Close variants of the abstract data perform much worse.** Replacing the simulated stack with a queue is worse than even a random initialization on QA benchmarks (**left**, token-level F_1 after language pretraining) and on DEPO (**right**, accuracy after direct transfer). The STACKRAND variant is closer to the original data and performs accordingly better, but still worse than STACK.

5.2 ADJUSTING WEIGHT MAGNITUDES DOES NOT REPRODUCE THE FULL BENEFITS

We test whether the benefits of training on STACK can be explained by a simple shift in weight magnitudes facilitating the optimization. We evaluate two interventions: after training on STACK, the *Gaussian init.* fills each weight tensor with Gaussian-distributed values of identical (per-tensor) mean and variance. The *Shuffled* intervention instead shuffles values within tensors, randomizing structure while preserving the exact distribution. We evaluate them on DEPO with direct transfer.

Figure 8 shows that both interventions perform significantly worse than the intact STACK-trained weights, particularly on longer chains. They retain some benefit over a random initialization, but neither reproduces STACK's rapid learning and near-perfect accuracy. This means that the information learned from abstract data is more complex than simple weight magnitudes.

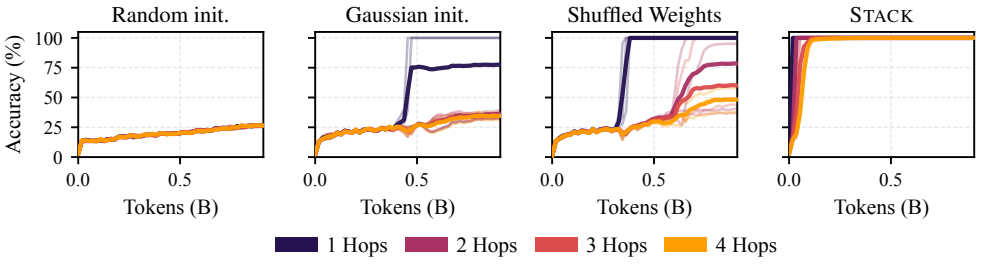


Figure 8: **Matching weight magnitudes does not reproduce STACK's benefits.** The test accuracy on DEPO improves slightly (center left/right) with random weights whose magnitude matches those trained on STACK, but they do not reach the near-perfect performance of the intact weights (right).

Takeaway. The benefits of training on abstract data depend on precise structure within the data, and the resulting weights capture more than a simple adjustment of their magnitudes.

6 EARLY EXPOSURE TO ABSTRACT DATA PRODUCES LASTING GAINS

We examine how the timing of abstract data in the training pipeline impacts its effects, comparing its use before, during, and after standard language pretraining. We then evaluate whether its benefits persist with increasing amounts of language pretraining by performing SFT from various checkpoints of the pretraining trajectory.

6.1 INITIAL EXPOSURE PRODUCES THE LARGEST GAINS

Using SmolLM-135M, we insert a STACK training phase at 0%, 25%, 50%, 75%, or 100% of a fixed FINEWEB-EDU token budget. STACK uses a separate optimizer, such that the language pretraining and its learning-rate schedule pause during intermediate insertions and then resume. We eventually fine-tune each model on one of the three QA benchmarks and report the mean change in F_1 relative to language-only pretraining.

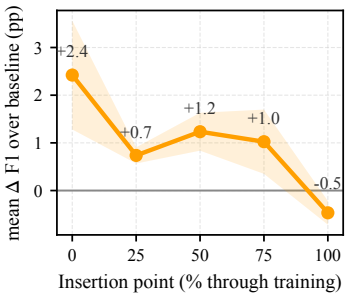


Figure 9: **The *initial* training on STACK produces the largest gain.** Mean change in F_1 over the three QA benchmarks relative to language-only pretraining.

Figure 9 shows that the *initial* STACK training produces the largest gains. Inserting STACK training during language pretraining yields smaller gains, while training afterward provides no mean QA improvement, even with the best learning rate tested (Table 6). Using STACK later in training therefore does not reproduce the gains from initial STACK training.

Controlled training order. We test whether the same timing effect holds for deep-chain reasoning capabilities using the controlled setting from Section 4.3. For both TINYSTORIES and FINEWEB-EDU, we compare STACK training before and after language pretraining, matching abstract- and language-training budgets and parameter-transfer protocols between these two settings. On both corpora, the initial STACK training accelerates the learning on four-hop DEPO. Training *afterward* on STACK instead reduces the final four-hop accuracy below the language-only baseline (Figure 10).

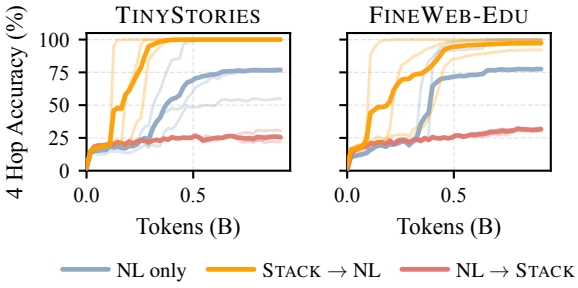


Figure 10: **Training order affects deep-chain reasoning acquisition.** Four-hop DEPO learning curves after pretraining on natural language (NL) only, STACK → language, or language → STACK.

Repeated exposure. We also test whether further training on STACK improves on initial training alone. Starting with initial STACK training, we add three further phases at 25%, 50%, and 75% of language pretraining (Figure 11). This schedule increases the amount of abstract training. Compared with the initial training alone, it improves mean F_1 on HOTPOTQA and 2WIKIMULTIHOPQA, but reduces it on MUSIQUE. More training on this abstract data is therefore not necessarily better.

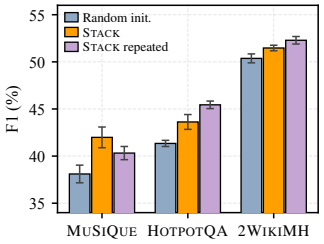


Figure 11: **Repeated exposure to abstract data has mixed effects.**

6.2 BENEFITS PERSIST THROUGHOUT LANGUAGE PRETRAINING

Finally, we test whether initial STACK training continues to benefit subsequent natural language multihop QA as language pretraining progresses. At matched language-token counts, we separately fine-tune checkpoints from STACK-initialized and randomly initialized SmolLM-135M models on each QA benchmark, using the same downstream protocol.

Figure 12 shows that models given initial STACK training achieve higher F_1 after fine-tuning at every evaluated checkpoint. Initial abstract training therefore continues to support subsequent multi-hop QA learning through billions of language-pretraining tokens.

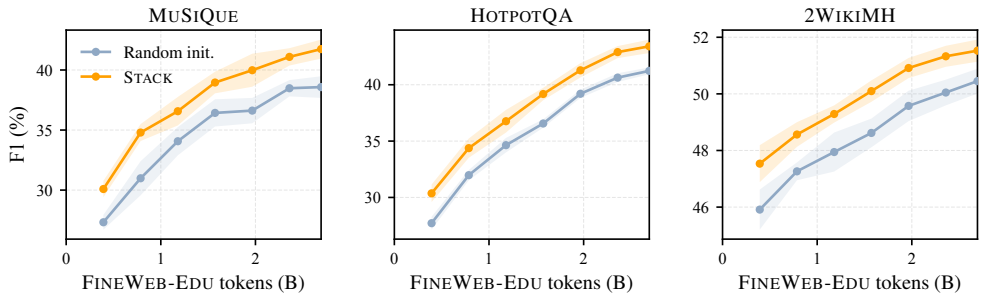


Figure 12: **The benefits of initial STACK training persist throughout language pretraining.** Mean token-level F_1 after separately fine-tuning successive SmolLM-135M checkpoints on each natural language multihop QA benchmark.

Takeaway. *Initial* training on STACK produces the largest gains among tested single-phase schedules, while doing so *after* language pretraining provides no benefit at all. The gains moreover persist through billions of tokens of language pretraining.

7 DISCUSSION

Abstract data is best used first. Prior work showed that pretraining on abstract data can improve subsequent language modeling. But it did not establish whether a dedicated initial phase is necessary. We directly compared exposure to abstract data before, during, and after language. The benefits are largest when used first, smaller when mixed with language, and disappear when introduced afterwards. The early benefits also persist through billions of language tokens. This suggests that the abstract data impacts the whole training trajectory. The fact that the benefits cannot be obtained through post-training suggests that abstract data instills capabilities not present in baseline models.

The improvements are specific. We showed that the abstract data does not merely provide a generic improvement in language modeling. In the case of our STACK task, the improvements concentrate on multi-hop reasoning and long sequential chains. The specifics of the abstract data are also critical: a matched queue-manipulation task does not produce the same benefits. Future work on abstract data should therefore seek to identify and align shared structures between abstract and target data.

Limitations. We study only one type of abstract data and relatively small models. Other forms of abstract data e.g. from cellular automata (Lee et al., 2026) or random recurrent networks (Guo et al., 2026) may have different effects. Our results highlight the importance of capability-specific evaluations: improvements may be invisible in aggregate metrics like perplexity, but substantial for particular skills. Our experiments also complement prior work that already evaluated scalability to larger models (Jiang et al., 2026) and longer training (Cheng et al., 2026).

Future directions. We focused on the data and its effects but we do not explain how it changes the model's parameters. The simplicity of the abstract data is an opportunity for a mechanistic analysis of the circuits learned during the initial phase, and how they affect subsequent language training (see e.g. Schiffman 2026).

Another question is whether the benefits of abstract training can be obtained more efficiently. We showed that the initial phase shapes the optimization trajectory by modifying the inductive biases at initialization. If we could characterize these inductive biases more directly (Teney et al., 2024; 2026) it may be possible to obtain similar benefits with new architectures or initializations, rather than an explicit abstract-training phase. This could instill useful generic structures into the weights of a model in lieu of the arbitrary, random initializations in use today.

AI USE STATEMENT

Generative AI was used solely to assist with grammar and presentation. It did not contribute to the development of the core ideas, experimental design or execution, analysis, or interpretation of results.

REPRODUCIBILITY STATEMENT

We provide detailed descriptions of our methods, experimental settings, and evaluation procedures in the appendix. We will release the code required to reproduce our experiments upon publication.

REFERENCES

Alessandro Achille, Matteo Rovere, and Stefano Soatto. Critical learning periods in deep networks. In *International Conference on Learning Representations*, 2019.

Syeda Nahida Akter, Shrimai Prabhumoye, Eric Nyberg, Mostofa Patwary, Mohammad Shoeybi, Yejin Choi, and Bryan Catanzaro. Front-loading reasoning: The synergy between pretraining and post-training data. In *International Conference on Learning Representations*, 2026.

Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Leandro von Werra, and Thomas Wolf. SmolLM - blazingly fast and remarkably powerful, 2024.

Zeyuan Allen-Zhu. Physics of language models: Part 4.1, architecture design and the magic of canon layers. *Advances in Neural Information Processing Systems (NeurIPS)*, 2026.

Nora Belrose, Quintin Pope, Lucia Quirke, Alex Mallen, and Xiaoli Fern. Neural networks learn statistics of increasing complexity. *arXiv preprint arXiv:2402.04362*, 2024.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *International Conference on Machine Learning (ICML)*, 2009.

Peter Bloem. Universal pre-training by iterated random computation. *arXiv preprint arXiv:2506.20057*, 2025.

Jo-Ku Cheng, Nikolaos Aletras, and Marco Valentino. Logic before language: Pre-pretraining on formal derivations fosters skill acquisition and compressibility. *arXiv preprint arXiv:2608.03930*, 2026.

Cheng-Han Chiang and Hung-yi Lee. On the transferability of pre-trained language models: A study from artificial datasets. In *AAAI Conference on Artificial Intelligence*, 2022.

Ronen Eldan and Yuanzhi Li. TinyStories: How small can language models be and still speak coherent english? *arXiv preprint arXiv:2305.07759*, 2023.

Michael Goodale, Salvador Mascarenhas, and Yair Lakretz. Meta-learning neural mechanisms rather than bayesian priors. *arXiv preprint arXiv:2503.16048*, 2025.

Xu Guo, Runyu Peng, Jian Tong, Yunhua Zhou, Haijun Lv, Zhihui Lu, and Qipeng Guo. Synthetic pre-pre-training improves language model robustness to noisy pre-training data. *arXiv preprint arXiv:2605.10129*, 2026.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps. In *International Conference on Computational Linguistics*, 2020.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

Michael Y. Hu, Jackson Petty, Chuan Shi, William Merrill, and Tal Linzen. Between circuits and chomsky: Pre-pretraining on formal languages imparts linguistic biases. In *Annual Meeting of the Association for Computational Linguistics (Long Papers)*, 2025.

Liangze Jiang, Zachary Shinnick, Anton van den Hengel, Hemanth Saratchandran, and Damien Teney. Procedural pretraining: Warming up language models with abstract data. In *International Conference on Machine Learning (ICML)*, 2026.

Anmol Kabra, Yilun Yin, Albert Gong, Kamilė Stankevičiūtė, Dongyoung Go, Johann Lee, Katie Luo, Carla Gomes, and Kilian Weinberger. Learning from synthetic data improves multi-hop reasoning. In *International Conference on Learning Representations*, 2026.

Dan Lee, Seungwook Han, Akarsh Kumar, and Pulkit Agrawal. Training language models via neural cellular automata. *arXiv preprint arXiv:2603.10055*, 2026.

Matthias Lindemann, Alexander Koller, and Ivan Titov. Sip: Injecting a structural inductive bias into a seq2seq model by simulation. In *Annual Meeting of the Association for Computational Linguistics*, 2024.

Hong Liu, Sang Michael Xie, Zhiyuan Li, and Tengyu Ma. Same pre-training loss, better downstream: Implicit bias matters for language models. In *International Conference on Machine Learning (ICML)*. PMLR, 2023.

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

James Michaelov, Roger Levy, and Benjamin Bergen. Language model behavioral phases are consistent across architecture, training data, and scale. *Advances in Neural Information Processing Systems (NeurIPS)*, 2026.

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

Guilherme Penedo, Hynek Kydlíček, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, Thomas Wolf, et al. The fineweb datasets: Decanting the web for the finest text data at scale. *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

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

Joshua S Schiffman. Transformers converge to invariant algorithmic cores. *arXiv preprint arXiv:2602.22600*, 2026.

Zachary Shinnick, Liangze Jiang, Hemanth Saratchandran, Damien Teney, and Anton van den Hengel. Transformers pretrained on procedural data contain modular structures for algorithmic reasoning. In *MOSS Workshop at ICML*, 2025.

Zachary Shinnick, Liangze Jiang, Hemanth Saratchandran, Damien Teney, and Anton van den Hengel. Can you learn to see without images? procedural warm-up for vision transformers. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026.

Damien Teney, Armand Mihai Nicolicioiu, Valentin Hartmann, and Ehsan Abbasnejad. Neural redshift: Random networks are not random functions. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

Damien Teney, Liangze Jiang, Hemanth Saratchandran, and Simon Lucey. Can transformers really do it all? on the compatibility of inductive biases across tasks. In *International Conference on Machine Learning (ICML)*, 2026.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. MuSiQue: Multi-hop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 2022.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. In *Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2023.

Yana Veitsman, Jonas Mayer Martins, Jonathan Lautenschlager, and Lisa Beinborn. Structural priors for data-efficient language learning. *arXiv preprint arXiv:2609.11505*, 2026.

Petar Veličković, Federico Barbero, Christos Perivolaropoulos, Simon Osindero, and Razvan Pascanu. Perplexity cannot always tell right from wrong. *arXiv preprint arXiv:2601.22950*, 2026.

Yuhuai Wu, Markus N Rabe, Wenda Li, Jimmy Ba, Roger B Grosse, and Christian Szegedy. Lime: Learning inductive bias for primitives of mathematical reasoning. In *International Conference on Machine Learning (ICML)*, 2021.

Yuhuai Wu, Felix Li, and Percy S Liang. Insights into pre-training via simpler synthetic tasks. *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

Sangwon Yu, Ik-hwan Kim, Jongyoon Song, Saehyung Lee, Junsung Park, and Sungroh Yoon. Unleashing multi-hop reasoning potential in large language models through repetition of misordered context. In *Findings of the Association for Computational Linguistics: NAACL*, 2025.

Shiyang Zhang, Aakash Patel, Syed A Rizvi, Nianchen Liu, Sizhuang He, Amin Karbasi, Emanuele Zappala, and David van Dijk. Intelligence at the edge of chaos. *arXiv preprint arXiv:2410.02536*, 2024.

A PROCEDURAL PRETRAINING DETAILS

A.1 PROCEDURAL TASKS

STACK. We adopt the STACK task from Jiang et al. (2026) and use the authors' implementation.[1] Each example consists of a sequence of push and pop operations over abstract symbols. A push operation adds a symbol to the top of the stack, while a pop operation removes the current top element. After executing the full sequence, the model predicts the remaining stack contents from top to bottom. The task therefore requires maintaining and updating an internal state under last-in-first-out (LIFO) access. The task contains no natural-language semantics: symbols are arbitrary, and the target is determined entirely by the sequence of stack operations. Training configurations for the natural-language and controlled settings are provided in Appendices A.2 and A.3, respectively. Table 1 provides an example.

Structural controls. We construct two variants using exactly the same input sequences as STACK. The FIFO control changes the interpretation of the pop token: it removes the oldest element rather than the most recently added element, and the remaining queue contents are emitted front-to-back. The shuffled-output control preserves the original LIFO state updates but independently shuffles the final stack contents for each example before emitting them. All other data-generation rules follow the original implementation. Table 1 provides examples.

Alternative procedural tasks. In the controlled setting only, we additionally evaluate a broad suite of procedural tasks spanning sequence transformations, formal languages, and cellular automata. Following Jiang et al. (2026), we use IDENTITY, which copies the input sequence; REVERSE, which outputs it in reverse order; DELETE, which removes a specified token; SET, which removes duplicate tokens; SORT, which outputs the sequence in sorted order; and UNION, which combines two sequences while removing duplicates. We use neural cellular automata (NCA) following Lee et al. (2026), with configurations of increasing complexity. Finally, following the formal-language setup of Hu et al. (2025), we use DYCK, which models nested bracket structure, and DYCK-SHUFFLE, a matched non-nested variant. Exact controlled-training configurations are provided in Appendix A.3.

A.2 NATURAL-LANGUAGE PROCEDURAL TRAINING

Data construction and objective. Stage-(1) procedural pretraining uses the STACK task described in Appendix A.1. Each example contains 128 input tokens, followed by a separator and the final stack contents read from top to bottom. We apply the language-modeling loss only to output tokens, masking the input and separator.

Vocabulary and transfer. For each architecture, we construct the procedural task using the model's native vocabulary. The full procedurally pretrained checkpoint can therefore be transferred directly into standard pretraining, including token embeddings and the output head, without resizing or reinitialization.

Optimization. We train for 5,000 steps with effective batch size 64. Hidden attention and MLP parameters are optimized with Muon using learning rate 4×10^{-3}, momentum 0.95, and weight decay 0.05. Embeddings, the output head, and remaining parameters use an auxiliary AdamW optimizer with learning rate 10^{-4}; weight decay is applied to matrix parameters but not one-dimensional parameters. We use 250 warm-up steps followed by cosine decay to 10% of the peak learning rate.

A.3 CONTROLLED PROCEDURAL TRAINING

Model and optimization. All controlled procedural experiments use a 4-layer, 4-head, 512-dimensional GPT-2 transformer ($\sim$12.7M parameters). We use AdamW throughout. Table 2 reports the exact data and optimization configurations for the procedural-task sweep.

[1] https://github.com/zlshinnick/procedural-pretraining

Task	Seq. len.	Vocab.	Optimization	Task-specific setting
Procedural tasks from Jiang et al. (2026)				
STACK	128	102	10k steps, eff. bs 64, 5×10^{-4} const.	push/pop; emit stack top-first
IDENTITY	128	102	10k steps, eff. bs 64, 5×10^{-4} const.	copy input
REVERSE	128	102	10k steps, eff. bs 64, 5×10^{-4} const.	reverse input
DELETE	128	102	10k steps, eff. bs 64, 5×10^{-4} const.	remove specified token
SET	128	102	10k steps, eff. bs 64, 5×10^{-4} const.	emit unique elements
SORT	128	102	10k steps, eff. bs 64, 5×10^{-4} const.	sort input
UNION	128	102	10k steps, eff. bs 64, 5×10^{-4} const.	deduplicated union
Neural cellular automata following Lee et al. (2026)				
NCA-Low	1024	64,000	5k steps, eff. bs 32, 10^{-4} cos.	complexity 0.10–0.25
NCA-Medium	1024	64,000	5k steps, eff. bs 32, 10^{-4} cos.	complexity 0.25–0.50
NCA-High	1024	64,000	5k steps, eff. bs 32, 10^{-4} cos.	complexity 0.50–1.00
Formal languages following Hu et al. (2025)				
DYCK	2048	129	500 steps, eff. bs 32, 5×10^{-4} const.	$k = 64$
DYCK-SHUFFLE	2048	129	500 steps, eff. bs 32, 5×10^{-4} const.	$k = 64$
Structural controls introduced in this work				
Shuffled-output STACK	128	102	10k steps, eff. bs 64, 5×10^{-4} const.	LIFO state updates; permuted read-out
FIFO	128	102	10k steps, eff. bs 64, 5×10^{-4} const.	pop oldest; emit oldest-first

Table 2: **Procedural pretraining configurations used in the controlled experiments.** All models are 4-layer, 4-head, 512-dimensional GPT-2 transformers ($\sim$12.7M parameters) trained with AdamW over three seeds. For tasks adopted from prior work, we follow the original protocols as closely as possible while adapting them to our controlled model and transfer setting. The FIFO and shuffled-output STACK controls use the same training configuration as STACK.

Training objective. For the input–output procedural tasks from Jiang et al. (2026), the language-modeling loss is applied only to output tokens. The NCA tasks follow the next-token prediction setup of Lee et al. (2026), while DYCK and DYCK-SHUFFLE use full-sequence language-modeling loss following Hu et al. (2025).

Transfer setup. Following Jiang et al. (2026); Lee et al. (2026), we transfer the transformer blocks from the procedural checkpoint into subsequent training stages while reinitializing vocabulary-specific parameters where required. For direct transfer to DEPO, token embeddings, positional embeddings, and the language-modeling head are freshly initialized. All parameters remain trainable after transfer.

B STANDARD PRETRAINING DETAILS

B.1 NATURAL-LANGUAGE STANDARD PRETRAINING

Data. We use FINEWEB-EDU (Penedo et al., 2024) with the 49,152-token SmolLM tokenizer. EOS-terminated documents are concatenated into 2,048-token blocks, discarding incomplete blocks and allowing cross-document attention. Each run uses blocks without repetition.

Models and training budgets. We train SmolLM-135M and SmolLM-360M with an effective batch size of 96 sequences and approximately Chinchilla-optimal budgets of 20 tokens per parameter. Table 3 summarizes the budgets.

Table 3: **Language-pretraining budgets.** All runs use 2,048-token sequences and an effective batch size of 96 sequences.

Model	Language tokens	Optimization steps
SmolLM-135M	2.69B	13,701
SmolLM-360M	7.20B	36,621

Optimization and hyperparameter selection. We use Muon for hidden attention and MLP matrices, with auxiliary AdamW for embeddings, the output head, and remaining parameters. Muon uses Nesterov momentum with coefficient 0.95. Auxiliary AdamW uses $\beta_1 = 0.9$, $\beta_2 = 0.95$, and $\epsilon = 10^{-10}$. To establish a tuned baseline, we perform a hyperparameter grid search on randomly initialized SmolLM-135M, training every configuration for the full Chinchilla-optimal budget of 2.69B tokens. We select the configuration with the lowest validation loss: Muon learning rate 0.016, weight decay 0.01, and auxiliary AdamW learning rate 3×10^{-3} (Table 4). We use these settings for all SmolLM language-pretraining runs. Both optimizers use linear warm-up over the first 5% of training steps, followed by cosine decay to 10% of their respective peak learning rates.

Table 4: **Muon hyperparameter search on SmolLM-135M.** Each configuration is trained for 2.69B FINEWEB-EDU tokens. The selected configuration has the lowest validation loss (in bold).

Muon LR	Weight decay	Auxiliary AdamW LR	Validation loss ↓
0.016	0.01	3×10^{-3}	**2.9144**
0.020	0.01	3×10^{-3}	2.9150
0.010	0.01	3×10^{-3}	2.9225
0.008	0.01	3×10^{-3}	2.9275
0.008	0.10	3×10^{-3}	2.9361
0.010	0.10	3×10^{-3}	2.9430
0.008	0.10	3×10^{-4}	2.9608
0.010	0.01	3×10^{-4}	2.9662
0.010	0.10	3×10^{-4}	2.9674
0.016	0.01	3×10^{-4}	2.9687
0.008	0.01	3×10^{-4}	2.9689
0.016	0.10	3×10^{-3}	2.9695
0.020	0.01	3×10^{-4}	2.9756
0.020	0.10	3×10^{-3}	2.9884
0.016	0.10	3×10^{-4}	2.9982
0.020	0.10	3×10^{-4}	3.0195

Initialization and controls. The baseline starts from random initialization, whereas the procedural condition retains the full stage-(1) model, including token embeddings and the language-modeling head. Both conditions use the same stage-(2) training data, token budget, and optimization settings.

Random seeds. For SmolLM-135M, we use three language-pretraining seeds per setting. Each procedural run starts from an independently trained STACK checkpoint paired with a distinct

Table 5: **Retained natural language QA benchmark examples after context-length filtering.** Counts are retained/original examples using the SmolLM tokenizer.

Benchmark	Training	Development
MUSIQUE	19,937/19,938	2,417/2,417
HOTPOTQA	18,786/20,000	6,813/7,405
2WIKIMULTIHOPQA	19,674/20,000	12,075/12,576

language-pretraining seed. For SmolLM-360M we use one pretraining seed due to computational cost. Downstream fine-tuning seeds are specified in Appendix B.3.

B.2 CONTROLLED STANDARD PRETRAINING

Data and model. We perform controlled language pretraining on TINYSTORIES and FINEWEB-EDU. For TINYSTORIES, we use `noanabeshima/TinyStoriesV2`. For both datasets, we use a 10,000-token BPE tokenizer. Both settings use a GPT-2 transformer with four layers, four attention heads, hidden dimension 512, dropout 0.1, and tied input and output embeddings, with approximately 12.6M non-embedding parameters.

Optimization and budget. We use AdamW with learning rate 5×10^{-4}, $\beta = (0.9, 0.999)$, $\epsilon = 10^{-8}$, and no weight decay. The effective batch size is 32 sequences and sequence length is 2048. We train for 5,731 optimization steps, using 10% linear warm-up followed by cosine decay to 5×10^{-5}. We target a Chinchilla-optimal budget of approximately 20 language tokens per parameter, using identical language-training recipes for random and procedural initializations within each corpus.

B.3 NATURAL-LANGUAGE MULTI-HOP EVALUATION

Benchmarks and context. We evaluate on MUSIQUE using gold supporting paragraphs, and on HOTPOTQA and 2WIKIMULTIHOPQA using all provided paragraphs, including distractors. Models receive the context and question and are trained to produce short answers without reasoning traces.

Fine-tuning. We fine-tune all parameters for one epoch using AdamW with learning rate 3×10^{-4}, weight decay 0.01, and effective batch size 32. We use 3% linear warm-up followed by cosine decay, with loss applied only to answer and EOS tokens.

Context-length filtering. We exclude examples exceeding 2,048 tokens, including answer and EOS tokens during training or a 64-token generation allowance at evaluation. Retained subsets are identical across model sizes and initializations (Table 5).

Evaluation. We evaluate final checkpoints on development splits using greedy decoding (up to 64 tokens), taking the first output line as the answer. We report F_1 and exact match using the official MUSIQUE evaluation code (Trivedi et al., 2022).

Random seeds. Each pretrained checkpoint is fine-tuned with three random seeds, yielding nine runs per setting for SmolLM-135M (three checkpoints) and three for SmolLM-360M (one checkpoint). We report means and standard deviations across runs, capturing both pretraining and fine-tuning variation for SmolLM-135M.

Reasoning structure. We classify official question types by the dominant (longest-path) structure of their composition graphs. Chain-dominated types pass intermediate answers between dependent hops, whereas merge-dominated types join independent lookups.

B.4 CONTROLLED MULTI-HOP EVALUATION

DEPO task. Each example contains a directed cycle over 10 distinct entities, sampled from a vocabulary of 20 single-token entities. The 10 directed edges are presented in random order, followed

by 10 queries, one for each starting entity. Each query asks for the entity reached after k successive transitions, with k sampled uniformly from $\{1, 2, 3, 4\}$ during training. Loss is applied only to answer tokens.

Example. For a cycle beginning $a \to b \to c \to d$, a query starting at a with $k = 3$ has answer d. Edges are encoded as `SEP a b`, and queries as `Q_k a ANS d`, where `Q_k` specifies the hop count. Each complete example contains 71 tokens.

Model and optimization. We use the GPT-2 transformer with four layers, four attention heads, hidden dimension 512. We train for 100,000 steps with batch size 128 using AdamW, learning rate 5×10^{-4}, $\beta = (0.9, 0.999)$, $\epsilon = 10^{-8}$, and no weight decay. We use 1,000 linear warm-up steps followed by cosine decay to zero.

Evaluation protocol. We evaluate every 2,000 steps on a fixed set of 500 examples per hop, giving 5,000 answer predictions for each $k \in \{1, 2, 3, 4\}$.

C Additional Results

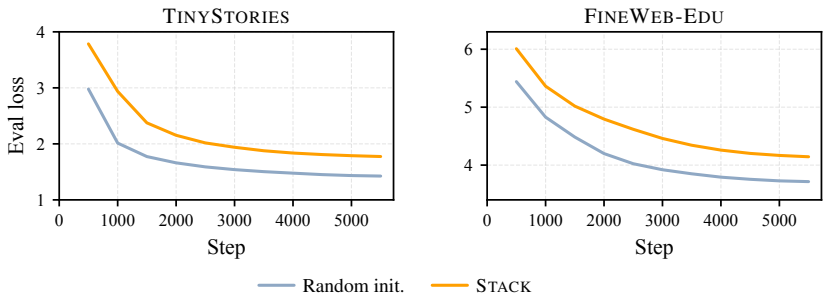


Figure 13: **Language-modeling validation loss on TinyStories and FineWeb-Edu.** Stack-initialized models exhibit higher loss than randomly initialized models throughout language pretraining.

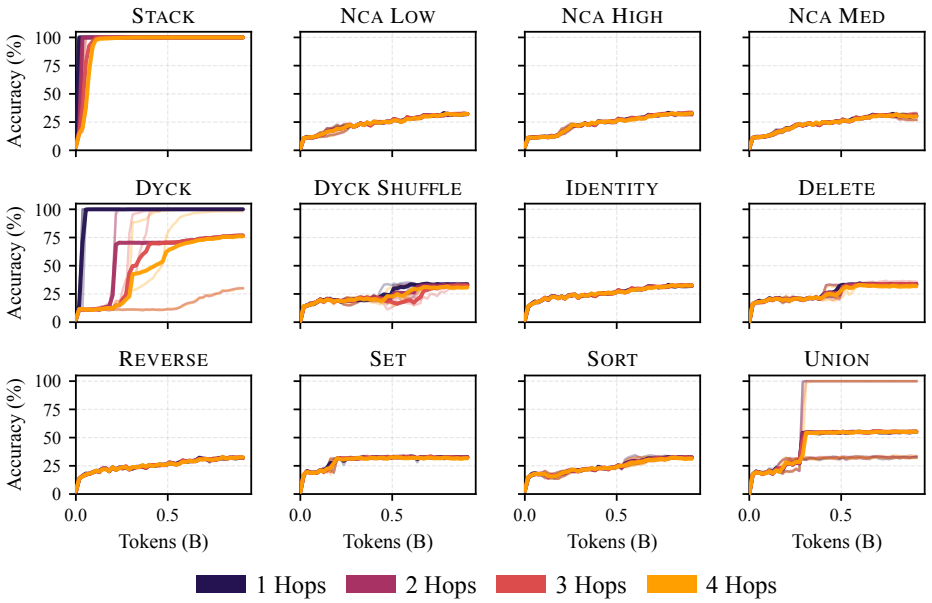


Figure 14: **Full comparison of abstract data on DEPO.** Accuracy by hop count during downstream training after initial training on abstract data, without intervening language pretraining. Stack enables the fastest learning to near-perfect accuracy across all four hop counts; other types of abstract data yield slower or less complete learning, particularly on longer chains.

Table 6: **Learning-rate sweep for post-language Stack training.** Downstream F_1 for SmolLM-135M, using three pretraining seeds per configuration. Language-only and initial Stack exposure are included as references.

Muon LR	Auxiliary LR	MuSiQue	HotpotQA	2WikiMH
4×10^{-3}	10^{-4}	23.57	25.07	44.96
1.3×10^{-3}	3.3×10^{-5}	35.97	38.66	49.26
4×10^{-4}	10^{-5}	37.93	40.57	50.01
Language-only baseline		38.10	41.34	50.37
Stack → FineWeb-Edu		41.99	43.54	51.24